Universidad Carlos III de Madrid
Computer Science Department
Ph.D. Programme in Computer Science and Technology

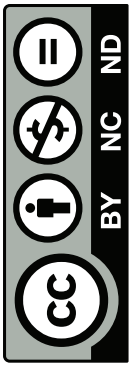
Ph.D. Dissertation

Evolutionary Design of Deep Neural Networks

Author: Alejandro Baldominos Gómez

Supervisors: Dr. Pedro Isasi Viñuela
Dr. Yago Sáez Achaerandio

Leganés. June, 2018



This page has been intentionally left blank.

This Ph.D. dissertation has been partially supported by the Spanish Ministry of Education, Culture and Sports under FPU fellowship with identifier FPU13/03917.

Part of this Ph.D. dissertation was written during a research stay in the AnyScale Learning For All group of the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology.

This research stay has been partially co-funded by the Spanish Ministry of Education, Culture and Sports under FPU short stay grant with identifier EST15/00260.

Please feel free to reach the author at the following electronic address: abaldomi@inf.uc3m.es

Ph.D. Dissertation

Evolutionary Design of Deep Neural Networks

Author: Alejandro Baldominos Gómez

Supervisors: Dr. Pedro Isasi Viñuela
Dr. Yago Sáez Achaerandio

THE EXAMINING BOARD

President: Dr. Araceli Sanchís de Miguel
Secretary: Dr. Inés Galván León
Intl. Member: Dr. Simon Lucas

After the defense of this **Ph.D. thesis**, taking place in **Escuela Politécnica Superior** of **Universidad Carlos III de Madrid** (Leganés) on September _____, 2018, the examining board agrees to confer the next **GRADE**: _____

This page has been intentionally left blank.

"Model building is the art of selecting those aspects of a process that are relevant to the question being asked. As with any art, this selection is guided by taste, elegance, and metaphor; it is a matter of induction, rather than deduction. High science depends on this art."

John Henry Holland (1929–2015)

"You don't understand anything until you learn it more than one way."

"No computer has ever been designed that is ever aware of what it's doing; but most of the time, we aren't either."

Marvin Lee Minsky (1927–2016)

This page has been intentionally left blank.

Agradecimientos

To those non-Spanish speakers reading these lines, I beg pardon for writing them in my mother tongue. I hope by the moment you are reading this, deep learning-powered machine translation has become proficient enough as to be able to output an accurate text in your language that manages to properly preserve all the passion I tried to put in each of these words.

Eran las once de la noche del primer domingo de julio. Nuestro protagonista se encontraba en un modesto pero bien cuidado hotel rural a las afueras de Pravia, un pequeño concejo de Asturias no muy lejos de la costa. La noche no era aún muy cerrada, si bien lo que unos minutos atrás se había dejado vislumbrar como un frondoso bosque en toda su extensión, ahora se tornaba en unas oscuras siluetas apenas reconocibles. La oscuridad había vencido a la poca bruma que antes se dejaba llevar por los aires asturianos, y el frescor de la noche empezaba a trasladar una ligera sensación de frío. Nuestro protagonista habría agradecido una copita de güisqui, incluso habría estado dispuesto a pagarla gustosamente de haber habido un mini-bar en su habitación. No era el caso, y el solo pensamiento de tener que moverse del sitio parecía cortar de raiz cualquier otro deseo que pudiera aparecer.

La noche era tranquila, quizás menos silenciosa de lo que debiera haber sido. A lo lejos se podía escuchar lo que a oídos de nuestro inexperto protagonista parecía ser un grillo, o algo similar. Más cerca se oían voces, que él trataba de acallar en su cabeza.

Acabar la tesis no había sido tarea fácil. Sin embargo, jamás habría anticipado ese final. Era, desde luego, mejor de lo que hubiera imaginado tan solo unas horas atrás. La perspectiva de vestir toga y birrete le resultaba atractiva, y la imagen que apareció en su cabeza chocaba frontalmente con el aspecto desaliñado que tenía en ese momento, con unos pantalones cortos raídos de un color a medio camino entre el blanco hueso y el blanco roto, una camiseta que difícilmente podría llevar a juego con nada y un peinado, eso sí, bastante más cuidado de lo que acostumbraba.

Miró al cielo. Un puñado de nubes se movía con bastante desgana. Detrás, en un firmamento en el que apenas destacaba alguna estrella solitaria, pudo llegar a sentir más estrellas a las que sin duda le habría gustado saludar y dar gracias. Ya no podría hacerlo. *Los valientes también lloran*.

Bajó la mirada, trasladándose de golpe al mundo de los vivos. En él había mucho que agradecer. Hacía cuatro años que decidió adentrarse en aquel tunel del que ya se veía la luz al final, y eran muchas las personas que habían conseguido iluminar su interior antes de que el final estuviera ni remotamente cerca.

Iluminar y dar calor. Se acordó de Vortex, Lava y Magma, quienes se habían visto abocados al averno ya tiempo ha, y se esforzaban en compartir la sensación con todo el

que se les acercaba. "*Les habría venido bien este clima*"–pensó, pues la simbiosis parecía de lo más acertada. Pero la libertad no estaba pensada para ellos.

Miró un momento hacia abajo. No muy lejos se encontraba aparcado un turismo de color blanco, que permanecía impasible ante la noche y el frío. Si pudiera manifestar algún tipo de emoción, esta con seguridad sería una preocupación ante los caminos que le tocarían recorrer en ese viaje. *Hemos llegado juntos hasta aquí*.

Por un momento se ladeó y echó un vistazo a la puerta por la que había salido, tan solo un instante antes de volver a centrar su mirada al frente. Su familia le había llevado hasta allí. A pesar de la preocupación de los últimos días, parecía haber sido una buena decisión. Tal vez haya que agradecérselo. "*A ellos y al gato, por las cicatrices*".

Recordaba bien el comienzo de todo. Tuvo que renunciar a un contrato digno para aceptar un trofeo cargado con el veneno de la burocracia. "*Pagan mal y no paran de buscar problemas*". No pudo dejar de pensar que todas aquellas coletillas en forma de agradecimiento no eran más que la hipocresía de turno, imprescindible para poder ajustarse a las normas que había prometido cumplir. *Que ningún cántaro de leche vuelva a desparramarse por el rellano*.

En realidad, el camino no había sido tan difícil. Había sido muy entretenido, divertido casi siempre. Ahí estaban los amigos de siempre para cervecear cuando hacía falta, porque *Fanta rules*. Nicás, Elena, Adri, Villar... siguen ahí de vez en cuando. *Haciendo que los días se digieran mejor que una tosta con tomate*.

También estaba la gente del Formol, que lo mismo organiza unas cenas que se anima a un partido de baloncesto mal jugado. Sergio, Víctor, Elena, Dani, Jorge, Álvaro, Isma, Nacho, Nega, Honduco, Lucía, y Almu, cuando aparece. *Que Langosta no te quite la sonrisa*. También Karina, que no se ha unido al grupo del Éter, pero debería.

Un insecto se posó en su pantalla. Sopló un poco, para hacerlo marchar. Se acordó de la gente de T3chFest, y de los buenos ratos, y de las celebraciones de los éxitos. Mario, Axel, Nerea. *Pesadilla después de Navidad*, pero una pesadilla muy agradecida.

Siguió pensando. Los caminos se hacen más fáciles cuando buenos caballos tiran de la carga. Pedro y Yago lo hacían, y si les sobraba tiempo quizás pudieran colocar un chuletón, como el que le coloca una zanahoria a un burro, salvo por ser esta menos alcanzable. *Y el chupito del final*.

Se sentía cuidado. Estaba en un grupo donde todo el mundo se quería y estaba dispuesto a ayudarse si hacía falta. Había mucha buena gente: David, Chema, Ricardo, Inés, Alejandro, Gustavo. Con todos había tenido contacto en algún momento. Todos le habían ayudado. También Cris y Espe, que hacía años que se habían bajado del carro. *Pero allí seguían*.

"*No solo en el grupo, tambien fuera*". Belén y Alex Calderón siempre dispuestos a echar una mano. JMAW, siempre preguntando qué tal las GPUs. *¿Siguen vivas?* Manu había tenido la cortesía de donar temporalmente un equipo para poder acelerar experimentos. A Javi y a Loli les debía mucho, pues estaba allí en parte gracias a ellos. *Ya hacía ocho años que todo había cambiado*. Había muchos con los que se podía intercambiar un saludo amable o una sonrisa por los pasillos. Subió la mirada un segundo, y luego volvió a mirar a la espesura del horizonte, donde lo que antaño había sido un bosque ahora empezaba a fundirse con la oscuridad del cielo.

También pareciera que el camino se ilumina más cuando ves a otros llevar tu misma carga. *Mal de muchos*. En ese grupo estaban Víctor Suárez, con quien se podía mantener una conversación sobre deep learning, Japón, o lo que surgiera; y Álvaro Montero, que de alguna forma se había terminado convirtiendo en el chico de las mudanzas.

También se acordó de Óscar y Víctor por la compañía diaria durante casi dos años. Confía en que las sesiones de *mentoring* improvisadas con Víctor hayan sido útiles.

Se acordó de todo el apoyo que tuvo cuando estuvo lejos de casa. Teresa, Imani, Alba y Dani le habían acogido como uno más. Los amiguis le habían acogido fuera de casa y alguna vez le invitaron a una buena barbacoa. Se acordó de Carlos, Mateo y Olivia. Le vino a la mente una queimada. También Una-May, que desde el principio le consideró un estudiante. Erik siempre estuvo para solucionar alguna duda técnica, y Catherine y Nicole se mostraron siempre dispuestas a arreglar los trámites administrativos. Lorette seguía ahí, mostrando su confianza en él. Siempre fue divertido tomar un café e intercambiar impresiones.

También había personas que convirtieron el camino en caminos, por los que desviarse, haciendo la experiencia más interesante. La gente de Blockverse: José, Mario y Diego, por aventurarse en lo desconocido y luchar por el triunfo. *Nunca pierdes hasta que lo dejas*. Carlos Afonso, que siempre tenía hueco para plantear algún negocio que acabara de surgir.

Llevaba un rato mirando al horizonte y no sabía muy bien a dónde más mirar. Miró al pasado y recordó ese momento: "*De qué va tu tesis*", le acababan de preguntar. "*Te lo digo en un par de meses*". Hizo falta más de un año para poder dar respuesta a esa pregunta. Quizás fue la curiosidad lo que lo inició todo. Quizás la confusión. Pero se había convertido en mucho más, y así pudieron recorrer parte del camino juntos durante casi tres años. *Tres años es casi toda la tesis*. A Clara le debía mucho, y ella debía saberlo.

Su mente volvió al presente. La noche ya era cerrada. Había pensado demasiado. *Habría pensado mejor con esa copita*. La temperatura había bajado lo suficiente como para invitar a nuestro protagonista a abandonar el balcón. La oscuridad sumía el paisaje y se fundía con las siluetas. No se oía nada.

Volvió a mirar al cielo. Algunos astros nuevos habían decidido manifestarse, y le devolvían el saludo. Era hora de despedirse. *Buenas noches*.

Gracias.

This page has been intentionally left blank.

Abstract

For three decades, neuroevolution has applied evolutionary computation to the optimization of the topology of artificial neural networks, with most works focusing on very simple architectures. However, times have changed, and nowadays convolutional neural networks are the industry and academia standard for solving a variety of problems, many of which remained unsolved before the discovery of this kind of networks.

Convolutional neural networks involve complex topologies, and the manual design of these topologies for solving a problem at hand is expensive and inefficient. In this thesis, our aim is to use neuroevolution in order to evolve the architecture of convolutional neural networks.

To do so, we have decided to try two different techniques: genetic algorithms and grammatical evolution. We have implemented a niching scheme for preserving the genetic diversity, in order to ease the construction of ensembles of neural networks. These techniques have been validated against the MNIST database for handwritten digit recognition, achieving a test error rate of 0.28%, and the OPPORTUNITY data set for human activity recognition, attaining an F1 score of 0.9275. Both results have proven very competitive when compared with the state of the art. Also, in all cases, ensembles have proven to perform better than individual models.

Later, the topologies learned for MNIST were tested on EMNIST, a database recently introduced in 2017, which includes more samples and a set of letters for character recognition. Results have shown that the topologies optimized for MNIST perform well on EMNIST, proving that architectures can be reused across domains with similar characteristics.

In summary, neuroevolution is an effective approach for automatically designing topologies for convolutional neural networks. However, it still remains as an unexplored field due to hardware limitations. Current advances, however, should constitute the fuel that empowers the emergence of this field, and further research should start as of today.

Keywords: topology design, neural architecture search, neuroevolution, evolutionary computation, convolutional neural networks, genetic algorithms, grammatical evolution, representation learning, supervised learning, deep learning, machine learning, artificial intelligence

This page has been intentionally left blank.

Contents

This page has been intentionally left blank.

List of Figures

This page has been intentionally left blank.

List of Tables

Chapter 1

Introduction

In this chapter we will introduce the work that will be performed during this Ph.D. thesis. To begin, in section 1.1 we will describe the context in which this work takes place, by locating it within the right "spot" in the field of computer science. Understanding the context is important to have a better understanding of the disciplines surrounding this work.

Secondly, in section 1.2 we will explain the motivation for this thesis by describing a problem we have found and for which we will propose a solution. To do so, we will work under certain hypotheses which are described in section 1.3, in order to achieve the objectives we aim to fulfill with the completion of the current research work, which are listed in section 1.4.

To achieve these objectives, we will follow a well-established methodological approach in order to proceed with this work. This methodology will be introduced and described in section 1.5. After the completion of this research work, we expect our results to have a scientific impact. Though it is not possible to advance whether this impact will eventually take place, our expectations are elaborated to some extent in section 1.6.

Finally, section 1.7 describes how the current document is structured.

1.1 Context

This thesis involves a research work in computer science, which is by itself a very broad field encompassing all aspects regarding the study of computers, including but not constrained to hardware design, networking, software engineering, computer security, theory of computation, etc. To be more specific, when placing this work within an area of knowledge we will state that this thesis involves research in the field of artificial intelligence (AI). A more in-depth analysis of why we have considered this work to belong to artificial intelligence, including some philosophical background and considerations, is provided in section 2.1.

Despite being much more specific than computer science, the field of artificial intelligence is still rather big. However, in this thesis we will put the focus in two main subfields, namely: machine learning and search and optimization.

In a broad sense, machine learning (ML) is a field of study that aims at making computers learn. A more elaborated definition and a historical overview of machine learning are provided in section 2.1. While machine learning comprises many different types of problems, in this work we will focus in supervised learning, representation learning and deep learning:

- Supervised learning: this problem involves, given a set of data composed of features and a label, learning a model from the features in order to be able to guess the label. The fact that input data is labelled is why this problem is known as "supervised" learning. A more formal definition and extensive explanation of supervised learning will be provided in section 2.1.
- Representation learning: in some machine learning problems, features may be known in advance or be trivial to extract from the data. However, in some cases, extracting features from the data might be a challenging task, especially when raw data is available in complex formats (multidimensional waveforms, images, video, etc). This process is called "feature engineering", and can be performed manually. Representation learning is a set of machine learning techniques aiming at extracting a convenient set of features given a raw input, so they can be effectively exploited in further machine learning tasks (e.g., supervised learning). This discipline is also known as "feature learning".
- Deep learning: while there is no simple definition for this relatively new concept, we could state that it comprises a set of techniques which are composed of a sequence of several layers in order to extract features from raw data and to learn a model from such features. A more exhaustive study of deep learning will be provided in section 2.3.

The relationship between these three fields is as follows: supervised learning is a type of problem which can be solved by many different types of techniques, and which requires a set of features for the learning phase. These features can be extracted in different ways, one of them being representation learning. Of course, representation learning can also be used for automatic feature engineering for problems different than supervised learning. Deep learning can be applied for supervised learning or for other types of problems, and often involves representation learning in order to automatically extract features from raw data. However, not all representation learning techniques involve deep learning.

The intersection of these three disciplines conforms the boundaries in which this work takes place. In particular, one of the most representative techniques of deep learning are "deep neural networks". This term is often used to refer to neural networks with more than one hidden layer (as opposed to "shallow" neural networks which would only contain one hidden layer). Another relevant concept for this work is convolutional neural networks (CNNs), which include a series of convolutional layers for representation learning and subsequent dense or recurrent layers for concept learning. We will delve into all these concepts in chapter 2.

In the last years, convolutional neural networks have been applied to very diverse problems, including character recognition (OCR), image classification, natural language processing, signal processing and classification, etc.

Besides machine learning, this work also involves search and optimization, another important discipline within the field of artificial intelligence. In search problems, we want to find a goal state within a large set of states (search space), and to do so, we are given a start state and a transition function to move across states. Many search algorithms are based on heuristics, which are domain-specific functions guiding the search. Optimization problems are a special case of search problems, in which a cost function is computed from a set of values, and the objective is to find the optimal set of values in order to minimize the cost function. It is worth noting that the cost function can be stochastic, non-trivial or even unknown (a *black box*).

Some search and optimization techniques are known as metaheuristics. These algorithms are heuristic-guided, but the heuristic function is domain-independent; instead, they make very few assumptions about this function. Metaheuristics are suitable for combinatorial optimization when a specific heuristic is not available, and they will often find suboptimal solutions. Some of these techniques are known as biologically-inspired (or nature-inspired) metaheuristics, because they rely on some biological foundations in order to drive the optimization process.

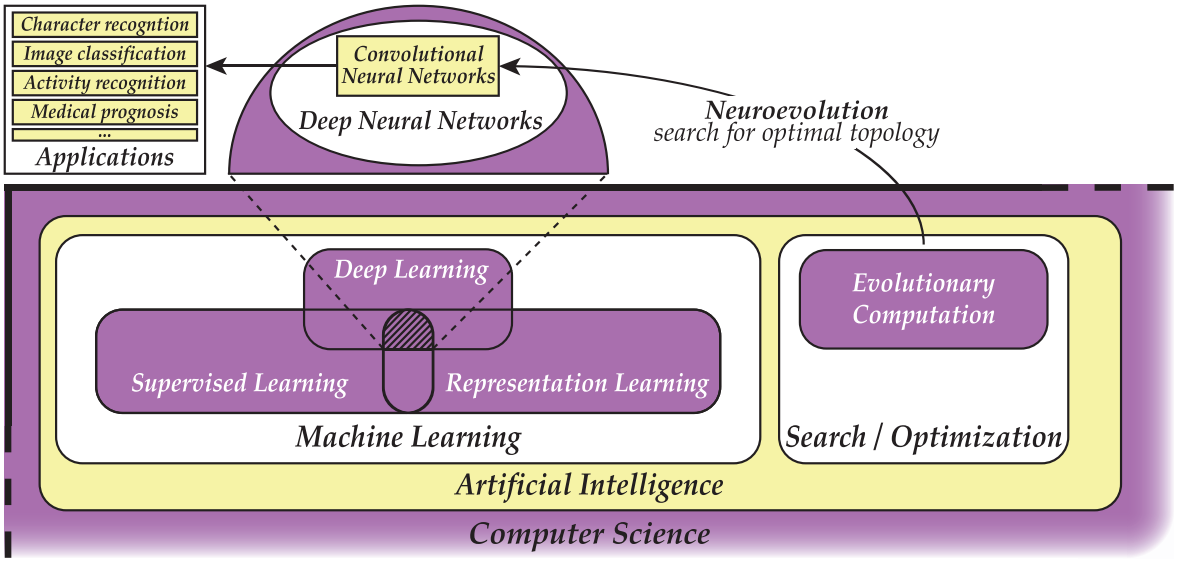


Figure 1.1: *Schematic context of this Ph.D. dissertation.*

A set of techniques located within biologically-inspired metaheuristics, and whose governing principles are based on Darwin's theory of evolution and natural selection, is known as evolutionary computation. Examples of evolutionary computation techniques include, but are not limited to, genetic algorithms, evolutionary strategies, genetic programming or grammatical evolution.

Figure 1.1 provides a schematic summary of the context surrounding this research work in the form of a Venn diagram. It should be noted that the picture is only illustrative and the sets are not in scale (e.g., machine learning and search/optimization are not the only fields within artificial intelligence). A specific contextualization of this work involves convolutional neural networks and evolutionary computation.

Finally, in order to provide a more accurate placement of this research, we will categorize it in the 2012 ACM Computing Classification System (CCS) [8], a hierarchical ontology for works in computer science and engineering. This work would belong to the following categories[1]:

- **Computing methodologies**
  - *Artificial intelligence*
    - Search methodologies
      - [Discrete space search]
    - Computer vision
  - *Machine learning*
    - Learning paradigms
      - Supervised learning
    - Machine learning approaches
      - [Neural networks]
      - Learning latent representations
    - Machine learning algorithms
      - Ensemble methods
- **Mathematics of computing**
  - *Discrete mathematics*
    - Combinatorics
      - [Combinatorial optimization]
- **Applied computing**
  - *Life and medical sciences*
    - [Health informatics]

[1]Categories displayed within square brackets are terminal nodes in the ACM CCS ontology.

1.2 Motivation

Artificial neural networks have been used for decades in order to tackle supervised learning problems. Some historical background of the history of neural networks will be provided in section 2.1, and section 2.2 will summarize the foundations and working mechanisms of neural networks. One of the neural networks models most widely used for classification and regression is the multilayer perceptron (MLP), which is an implementation of a feed-forward network.

A multilayer perceptron is characterized by a topology (also called architecture) and weights. The topology comprises one or more hidden layers, each of which will contain a number of hidden neurons. Neurons from one layer are connected to all neurons in the following layers, and each of these connections is assigned a weight. Also, neurons have another input parameter called bias, which is a special kind of weight not connected to any neuron from the previous layer.

Popularity of artificial neural networks, and of the multilayer perceptron in particular, grew with the discovery of the backpropagation process, first published in Nature in 1986. This process uses gradient descent in order to optimize the weights of the neural network. Since then, several optimization functions have arisen in order to automatically search for the optimal configuration of weights and biases in order to minimize the classification or regression error of the neural network.

However, regarding the topology, there is not a rule-of-thumb for determining the most suitable architecture for a given problem, and estimating it often involves trial-and-error; i.e., manually evaluating the performance of different architectures in order to determine the best one.

In the latest years, convolutional neural networks (CNNs), which lie within the field of deep learning, have been extensively used to solve a large variety of problems (e.g., computer vision, natural language processing, signal classification, etc). The main advantage of convolutional neural networks is that they contain some convolutional layers which are able to perform representation learning; i.e., to automatically learn features from raw data. On the other hand, CNNs involve a more complex topology than those of traditional neural networks, and thus require more hyperparameters to be specified, to mention a few: number of convolutional layers, number of kernels (or patches) per convolutional layer, size of the convolutional patch, activation function, pooling, padding, etc. The meaning of these hyperparameters will be explained in section 2.5.

Finally, the topology gets even more complicated if we consider that the neural network can have recurrent layers, where connections between neurons not only happen from one layer to the following one, but also to the same layer. This solution is an interesting approach when tackling the classification of time series or other information with a temporal component (handwritten recognition, speech recognition, etc). There are different types of implementations for recurrent topologies, some of which will be reviewed in section 2.6, and the decision on which to use adds further hyperparameters to the network architecture.

Determining the optimal topology of a convolutional neural network for a given problem is a hard task. Also, in some domains some topologies may not work at all, either because they are too simple, too complex, or they do not fit the data. So far, most researchers come up with a manual design of a convolutional neural network that satisfies their expectations, but this topology will likely be far from the optimal solution.

Finally, some works have evaluated the performance of ensembles or committees of convolutional neural networks; however, this area still remains quite unexplored. When using committees, several models with a good performance are put together in order to create a meta-classifier which performs better than any of the models from which it is composed. If, in the process of finding a good CNN architecture, other good architectures are found as well (even if they perform worse than the best found), they could be used to conform an ensemble.

In this thesis, we are motivated by the hardness of finding the optimal topology of a convolutional neural network for solving a given problem.

1.3 Working Hypotheses

In the previous section, we described the facts serving as motivation for the current work. In addition to these facts, we begin this research line under the following working hypotheses:

- Manual design of convolutional neural networks is a time-consuming task and leads to topologies which are far from the optimal.
- Topology design of CNNs can be approached as an optimization problem.
- Metaheuristics, and in particular evolutionary computation techniques, can be used for automatic optimization of CNN topologies in an affordable amount of time.
- Evolved topologies will in average outperform manually-designed topologies or, in the worst case, lead to similar results requiring less effort.
- Committees of convolutional neural networks will often perform better than a single neural network, and thus if several good solutions are found they can be used to build a committee.
- A system can be designed to automatically optimize the topology of CNNs given a supervised learning problem in an effective and efficient manner.

1.4 Objectives

Given the above motivations and working hypotheses, the main aim of this thesis is the following:

To explore the automatic design of optimal topologies of convolutional neural networks using evolutionary computation techniques, and explore the performance of the optimized topologies in order to validate that they are better than most, if not all, manually-engineered CNN architectures.

To achieve such aim, we propose the achievement of the following objectives:

1. Proposal of one or more domain-agnostic encodings that model the topology of a CNN in a way that can be evolved using specific evolutionary computation techniques.
2. Proposal of a fitness function, which may be domain-dependent, to be able to compare the performance of different CNN topologies.
3. Design and development of a system able to optimize the topology of a convolutional neural network, given the fulfillment of objectives (1) and (2).
4. Improvement of the system designed in objective (3) in order to reduce the computational cost in terms of time, by enabling parallel fitness computations.
5. Selection of different domains to evaluate the performance of automatically evolved CNNs. These domains should cover a wide spectrum of applications: handwritten character recognition, signal classification, etc.

6. Exhaustive review of the state of the art of the domains chosen in objective (5), explicitly discriminating best results attained using convolutional neural networks and other techniques.
7. Comparative evaluation of the best CNN obtained for each domain against the state of the art ranking elaborated in objective (6).
8. Construction of committees / ensembles using more than one of the best found CNN topologies for each domain.
9. Evaluation of the performance of the committees built in objective (9), comparing it against the state of the art and the best individual CNN.
10. Analysis of the results and validation, if applicable, of the working hypotheses.

1.5 Methodology

To achieve the aforementioned objectives, we will adhere to a strict and systematic research methodology, which involves the following steps:

1. Extensive review of the state of the art, in order to find related work. These works will be carefully analyzed in order to find and outline flaws or improvements that can be solved within the current research work.
2. Design of a proposal which is able to satisfyingly achieve the objectives given the working hypotheses. The proposal will be described with enough level of details as to enable reproducibility by the scientific community.
3. Provision of the hardware and implementation of the software required to successfully materialize the proposal.
4. Completion of an exhaustive evaluation of the results against the state of the art, in order to validate the competitiveness of the proposal.
5. Validation of the working hypotheses and of the achievement of the research objectives. If some of the hypotheses do not hold or some objectives are not achieved, this fact should be carefully analyzed and the underlying hypotheses must be corrected.
6. Extraction of conclusive remarks about the contributions of the work and proposal of future lines of work to guide research within this field.

1.6 Contribution

Successful research must have a scientific impact. In this thesis, we estimate the contribution to be the following:

- Learning how evolutionary computation techniques perform when optimizing the topology of convolutional neural networks, as compared to manual topology design.
- Availability of an efficient system to perform automatic optimization of the topology of convolutional neural networks given the definition of a supervised learning problem; i.e., data and an objective function.

- For each domain taking part in the evaluation, the result of the best CNN topology, or the best committee of CNNs will be contributed to the state of the art.

Because of these contributions, we expect this thesis to have a positive impact in future research in the application of convolutional neural networks to different domains. Because the system will be freely available and will enable parallel fitness computation, design of new topologies can be accelerated, especially if the system is deployed in a multi-GPU cluster. If the optimization system resulting from this thesis serves for advancing research in key areas, such as healthcare or medicine, significant positive social impact could be achieved.

1.7 Structure of the Document

The remainder of this document is structured as follows:

Chapter 2 provides a historical overview of machine learning and neural networks and describes some key concepts relevant to this research work, including convolutional neural networks, recurrent networks or evolutionary computation. A reader already familiarized with all these concepts can safely ignore this chapter, though an interesting philosophical dissertation on why we state this thesis to belong to the field of artificial intelligence is provided by the beginning of the chapter.

Chapter 3 reviews the state of the art for the current research area. We will explore related works carrying out the automatic optimization of neural network topologies, focusing on convolutional neural networks but including key works on non-convolutional neural networks. We will consider all relevant works, regardless of whether they use or not evolutionary computation.

Chapter 4 describes the proposal of this thesis, starting by a formal definition of the problem to be solved and followed by an analysis of some of the challenges this problem poses and providing the main features of the system to be developed. Later, we study different alternatives for solving the problem at hand, finally providing an exhaustive design of the proposal.

Chapter 5 provides a systematic evaluation of the proposal. The chapter begins describing the experimentation environment in terms of hardware and software, with the aim of enabling full reproducibility of the experiments by the scientific community. Then, for each of the chosen domains, we will explain the data and outline any possible pre-processing performed over it, exhaustively review the state of the art, describe the experimentation setup and enumerate the results obtained along with a discussion. The end of the chapter summarizes the results of the evaluation.

Chapter 6 provides conclusive remarks for the current work, establishes whether the working hypothesis hold and determines if the research objectives have been achieved. This chapter also suggests potential future works in order to keep advancing this research area.

After the work conclusion, we have included some appendices in order to provide further relevant information without the scope of the thesis, for those readers who might be interested:

Appendix A describes the architecture of the graphical processing units (GPUs) used to improve the speed of the experiments.

Appendix B provides an analysis of time performance within the experiments carried out in this work. In particular, this appendix first compares the performance of training and running convolutional neural networks in CPU versus GPU hardware, and then observes some aspects involving the speed of different software setups.

Appendix C describes the scientific proposal of DeepNE, a project request presented to the Spanish Ministry of Economy, Industry and Competitiveness to carry out some future work presented in this thesis, in order to develop a distributed system for parallelizing the fitness computation, accelerating the evolutionary process.

Appendix D accounts for all the tasks performed during the development of this Ph.D. dissertation, which is partially supported with public funds. In particular, the chapter includes a list of publications in journal and top-tier conferences, summarizes teaching activity and explains the whole accountability process followed during these years.

Appendix E is a space where I try to share my vision of how artificial intelligence is evolving, from a rather philosophical and societal point of view. I strongly advice readers who expect to find scientific and rigourous claims to stay away from this appendix.

Chapter 2

Theoretical Background

In this chapter we will introduce the foundations required to understand the scope of this work. First, a brief introduction about the concept known as "artificial intelligence" is provided in section 2.1 along with a deeper description of machine learning, followed by a focus in the rise and growth of artificial neural networks, a broad set of techniques for training machine learning models, described in section 2.2.

Later, in section 2.3, we will discuss "deep learning", a name recently given to some techniques aiming at learning complex models from large amounts of data, often using deep neural networks. This is the fundamental technology underlying this work, and thus it is explained in detail in this chapter along with important related concepts. Because most implementations of deep learning rely on tensors for data representation, we will explain this concept, describing in section 2.4 how different types of data (e.g. digital signals, images, etc.) can be defined in terms of tensors. In general terms, the deep neural networks that we will train and exploit in this thesis comprise two key elements: convolutional layers, which are used for automatic feature extraction from the data; and dense layers, which are able to learn complex models to identify patterns in the data. This chapter delves in the theoretical foundations of these two key elements in sections 2.5 and 2.6 respectively, and provides a methodological and mathematically sound description on how they are applied over data.

Then, we enumerate different hyperparameters that affect how a deep neural network runs and how it is trained. In particular, neurons can implement different activation functions, the most relevant of them being described in section 2.7. Also, other hyperparameters known as "regularization" are often used to prevent the model from overfitting the training data, so we explain different regularization techniques in section 2.8. For training the model we need to establish two different elements: first, a loss function must be defined which computes how "good" the model is when fitting the training data; and second an update rule must be chosen to modify the deep deep neural network parameters in order to optimize the loss function. Most common loss functions and update rules (or optimizers) are enumerated and described in section 2.9.

Finally, we will cover the concept behind evolutionary computation, which is a set of biologically-inspired techniques aiming at solving certain optimization problems. In this thesis, we will use two of these techniques in order to automatically design the architecture of deep neural networks: genetic algorithms and grammatical evolution. By the end of this chapter, in section 2.11, we will delve into these techniques and explain their working mechanism.

2.1 Brief Introduction to Machine Learning

2.1.1 Towards Artificial Intelligence

It was John McCarthy who first coined the term "artificial intelligence" (AI) back in 1955 [261]. Even now, this term has no easy definition, other than *the ability of computers or machines to show an intelligent behavior*. Of course, the scope of what can be considered such an intelligent behavior is truly difficult to be established. For instance, some people would argue that a human who can perform thousands of complicated arithmetic operations per second is really intelligent; however, that is a common denominator of most current computers and can hardly be considered an intelligent feature in computational terms.

In 1956, McCarthy organized the Dartmouth Summer Research Project on Artificial Intelligence, along with Marvin Minsky, Nathaniel Rochester and Claude Shannon. This project was an extended brainstorming session to discuss and develop ideas regarding artificial intelligence, and to establish some common terminology, since by that time there were few consensus about how to name the field. This event has been considered by many researchers as the seed of artificial intelligence [256,336], yet the term and some early developments in AI existed before the event took place.

A few years before passing away, McCarthy himself defined artificial intelligence to be "*the science and engineering of making intelligent machines, especially intelligent computer programs*", pointing out that "*it is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable*" [233]. Again, this definition poses some philosophical questions regarding the nature of what is to be considered intelligent.

From a less philosophical but rather practical approach we can define such intelligence to comprise those abilities or behaviors often associated with humans but that are "difficult" for machines to perform. Examples of these abilities include but are not limited to: parsing logical statements, recognizing objects in an image or being able to provide a detailed description of it, understanding a complex sentence or being able to engage in a conversation, resolving ambiguity in a sentence, detecting emotions or even showing them, or learning from experience. Of course these abilities can be performed to a greater or lesser extent, and a machine could show these intelligent behaviors with different degrees of expertise, thus being possible that these tasks are performed even better than a human would do.

Prior to the use of the term "artificial intelligence", the ability of a machine to engage in a human-like conversation was already widely considered as an equivalent of its ability to "think". This equivalence was established after Alan Turing's work in 1950 on a modified version of the imitation game, which was later known as the Turing test [367]. In this work, Turing asks the question "Can machines think?" and proposes a game for determining machine intelligence based on the ability of a machine to imitate human behavior. In particular, a machine is considered to be intelligent if, under certain conditions, a human interviewer is not able to elucidate whether the interviewee is a human or a machine. Such conditions are enforced to avoid bias, such as resolving the test by physical appearance (it is simple to physically distinguish a human from a machine) or tone of voice (thus Turing suggests to use typewritten answers to the questions).

While extensively considered a condition for artificial intelligence, the Turing test poses some limitations, in part due to the anthropocentric approach of the test. An example of these limitations is that a machine could be distinguished from a human because of its ability to do faster calculations and thus provide very fast responses to complex arithmetical operations, a fact that can hardly be considered as a reason towards making the machine less intelligent or no intelligent at all.

In 1980, John Searle established a distinction between what he called "weak AI" and "strong AI" in his famous paper "Minds, brains, and programs" [325]. In strong AI, machines are required

to have consciousness or "a mind"; whereas weak AI machines are focused in narrow, more specific tasks. In that work, Searle proposes the Chinese room experiment, where he states that there is no practical difference between a computer program able to pass the Turing test which receives an input text in Chinese, processes it and returns an output in the same language; and himself in a closed room manually running the code of that same program along with some books, paper, pencils, etc. He claims that in this scenario, he is not able to understand the conversation, and neither is the computer in the first case, thus the machine is not really "thinking".

As a result, strong AI would be false. Still, there has been significant attempts in the history of computer science to develop software and hardware that resembles the human brain, some of which will be later described in this thesis when we discuss artificial neural networks and deep learning. It is a philosophical rather than technical discussion whether to consider that a "mind" requires biological mechanisms, or whether a computational device can display a consciousness.

In this thesis, we will remain agnostic to these philosophical considerations. However, we will doubtlessly state that this work contributes to research in AI, specifically to machine learning.

2.1.2 What is Machine Learning?

When talking of intelligence in anthropocentric terms, it is often closely related to the concept of learning. Learning is an ability mostly associated to humans (yet not exclusive to this specie) by which subjects are able to acquire new knowledge from different sources and through different methods: inductive, from experience, by reinforcement, etc.

"Machine learning" (ML) is a concept that arose in the late 1950s as a field within artificial intelligence. It was defined in 1959 by Arthur Samuel as the "field of study that gives computers the ability to learn without being explicitly programmed" [315]. Years later, in 1997, Tom M. Mitchell would provide a more formal and practical definition: "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E" [253].

While machine learning is a broad field that comprises many different techniques, most of them can be classified into three broad categories:

- Supervised learning: these techniques are fed with labelled data and they aim at learning a model which is able to assign a label to new data based on what is known about it. In general, supervised learning comprises the problems of classification and regression. In classification, the label is a discrete value, also known as "class". In regression, the label is a continuous value, and is often known as "*output*".
- Unsupervised learning: these techniques aim at learning a model which is able to identify patterns which are common to some data. This problem is known as "clustering", and it involves the segmentation of the input data into clusters, so that data within one cluster is "closer" (more similar) than data between different clusters, according to certain criteria.
- Reinforcement learning: these techniques aim at learning some policy which interacts with a dynamic environment (often by performing specific actions) in order to eventually achieve some goals. This policy is learned by reinforcement, i.e., some feedback (either positive or negative) is provided to the intelligent agent as it traverses the environment.

In this thesis, we will focus on supervised learning. These techniques are able to learn by induction (i.e., from experience) a model which is able to generalize that experience in order to discriminate some specific property of the data. This model can be used to do further prediction

with future information. To better understand how supervised learning works, three fundamental concepts must be known in advance: *instances*, *attributes* and *labels*:

- An *instance* is a particular example of the data. To illustrate this definition, let us use a classical example: a credit card issuer wants to decide whether a credit transaction is valid or fraudulent in order to accept or deny that transaction. In this case, an instance would be a transaction (the element for which the prediction is done), and the experience used to train the model would be conformed by a set of historical transactions.
- An *attribute*, also known as "feature", is some unitary piece of information belonging to an instance. For example, credit card transactions may comprise some of the next attributes: value, currency, concept, location, timestamp, etc. These attributes constitute the information that the machine learning technique will use in order to train a model.
- The *label* is a special attribute that the machine learning model will aim at "predicting"; i.e., guessing its value from the other attributes. The label is also known as "class" in classification problems or "output" in regression problems. In the credit card transactions scenario, the label or class would be whether the transaction is valid or fraudulent: this is an example of a binary classification problem, since there are only two classes to predict. When more classes exist, then it is known as "multi-class" or "multinomial" classification.

With these basic concepts already defined, we can now provide a more formal definition of the supervised learning problem. These problems receive as input m instances of the form $(\mathbf{x^{(1)}}, y^{(1)}), (\mathbf{x^{(2)}}, y^{(2)}), \ldots, (\mathbf{x^{(m)}}, y^{(m)})$, where $\mathbf{x^{(i)}}$ is the vector of attributes (or feature vector) for the i-th instance and $y^{(i)}$ is that instance's label. A supervised learning algorithm aims at learning a function $f : X \rightarrow Y$, where X is the input space and Y is the output space.

In many cases, such a function f will not exist. For this reason, supervised learning algorithms often use loss functions or some other quality metrics that allow stating that a function f_1 is better than f_2, even if $f_1(\mathbf{x^{(i)}}) \neq y^{(i)}$ for some instances. Given a loss function L, a supervised learning algorithm should aim at finding the function $f' \in F$, where F is the set of all functions $f : X \rightarrow Y$, such that it minimizes the sum of losses (equation 2.1) of all instances.

$$\underset{f'}{\operatorname{argmin}} \left(\sum_i L \left(y^{(i)}, f' \left(\mathbf{x^{(i)}} \right) \right) \right) \tag{2.1}$$

While function f can be of any type, supervised learning models are often classified within three categories according to their main intuition [102]:

- Probabilistic models: these models aim at learning the probability distribution of the input data. More specifically, they infer a conditional probability function to estimate the probability distribution of the output given the vector of attributes: $f(\mathbf{x}) = P(y|\mathbf{x})$. Some examples of probabilistic supervised learning models are Bayesian models, Markovian models, linear discriminant analysis or logistic regression.
- Geometric models: these models place the input data into an n-dimensional hyperspace, where n is the length of the feature vector. Models such as the k-nearest neighbors [332] infer an instance's output based on the outputs of its spatial neighborhood given a distance function, or metric (e.g., Euclidean distance, Manhattan distance or cosine distance), which defines how far a pair of instances are located in the hyperspace. Other algorithms learn a linear separator (a hyperplane) to discriminate between different classes in the hyperspace. Examples of these algorithms are support vector machines (SVM) [76] or artificial neural networks, which will be described later in further detail.

- Logical models: these models provide an instance's output by following a sequence of conditional rules. These models are often represented in the form of classification and regression trees (CART) [44], also known as decision trees. Most representative algorithms for learning decision trees are Quinlan's ID3 [291] and C4.5 [292]. These models are often part of *ensembles*, which combine the power of several decision trees in order to provide more robust predictions. Examples of commonly used ensemble models built out of decision trees are random forests [43] or extremely randomized trees [110].

2.1.3 A Historical Overview

The ancient, prehistoric days of machine learning can be established centuries before the term itself was coined. In 1763, Reverend Thomas Bayes' work on the mathematical theory of probability [19] was published two years after his death in Philosophical Transactions, the world's first scientific journal. Sixty years later, Pierre Simon Laplace extended Bayes' work and established what is now known as the Bayes' theorem [194]. Along with Andrey Markov's discovery in the early 1910s of the technique later known as *Markov chains*, these works constitute the mathematical foundations of most of the widely used probabilistic ML models, such as naive Bayes or Bayesian networks.

Also, in the early 1800s, Adrien-Marie Legendre described the least square method[1] which is still used nowadays to compute a linear regression model which fits input data placed in a plane or n-dimensional hyperspace.

In the 1950s, both the concept of *artificial intelligence* and of *machine learning* arose. By that time, after the Turing test was defined, there was a highly anthropocentric consideration of artificial intelligence. Maybe due to this reason, many approaches to machine learning in the 1950s and 1960s tried to resemble the mechanics of the human brain based on the principles provided by Hebb's connectionism: it was the birth of artificial neural networks.

In 1943, Warren McCulloch and Walter Pitts provided the first mathematical model of the behavior of an artificial neuron based on the idea that neurons operate using binary electric impulses [234]. However, it is often considered that the first implementation of an artificial neural network was Marvin Minsky and Dean Edmonds' SNARC (Stochastic Neural Analog Reinforcement Calculator) [249] in 1951, built in hardware using vacuum tubes. A significant achievement in AI arrived later in 1957 with Frank Rosenblatt's invention: the Perceptron [307]. The Perceptron was a neural network-based algorithm for learning a linear classifier to discriminate data in a binary classification problem. Its first implementation was in software for the IBM 704, yet it was later implemented in hardware known as the Mark 1 Perceptron. Despite the relative simplicity of the Perceptron algorithm, the project raised significant attention from the media, and the New York Times described it as "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence" [266]. By that time, Arthur L. Samuel was applying minimax, a game theory concept, along with an original tree search algorithm, namely "alfa-beta pruning", to develop a computer software able to beat a human in the game of checkers [315].

In the following years, the high expectations in the Perceptron led to a increase of funding for artificial intelligence research. However, results turned out to be disappointing. One of the most desired applications of artificial intelligence was machine translation, due to the interest of the US government during the Cold War in translating Russian documents in a fast and automatic way. However, by the mid-1960s, the US National Research Council had spent more than 20 million dollars without remarkable results, concluding that human translation was cheaper and more efficient.

[1] The invention of this method has been claimed by other authors, being Gauss one of the most relevant ones, yet it is Legendre who retains the priority of publication [348, 349].

Also, in 1969, Marvin Minsky and Seymour Papert explained some of the limitations of artificial neurons and the Perceptron, such as the inability to learn a linear classifier to discriminate simple functions such as the exclusive OR (XOR), in their book *Perceptrons* [250]. All this sequence of events resulted in what is now known as the "AI winter", which froze the research and the interest in artificial intelligence during the 1970s decade.

In 1980, Kunihiko Fukushima introduced the Neocognitron [108], a hierarchical artificial neural network that would serve years later as an inspiration for the invention of convolutional neural networks. In 1982, John Hopfield described one of the earliest developments of recurrent neural networks, known as the Hopfield network [155].

One of the most relevant discoveries; however, was the process of backpropagation in 1986 by David Rumelhart, Geoffrey Hinton and Ronald Williams [308], published in the top-tier journal *Nature*. This work was built on top of the work published a decade earlier by Seppo Linnainmaa, who described a method for automatic differentiation of nested differentiable functions [220]. Because backpropagation enables fast learning of artificial neural networks and the ability to optimize more complex architectures, in the 1980s the AI winter came to an end and research in AI put the focus back in these techniques. In 1989, Dean A. Pomerleau introduced ALVINN [285], a precursor of today's self-driving cars consisting in a neural network able to follow a road.

Also in 1986, Ross Quinlan described the Iterative Dichotomiser 3 (ID3), its popular entropy-based algorithm to build decision trees [291]. Since that year, many machine learning techniques based on logical models have been explored and developed, and these techniques are still widely used nowadays. Besides supervised learning, in 1989 Christopher Watkins introduced Q-learning, a popular algorithm for reinforcement learning, as the main contribution of his Ph.D. thesis [382].

The early 90s also brought some remarkable advances to AI, and constituted the renaissance of logical models. In 1990, Robert Schapire proved that a set of weak classifiers can be coupled together to create a single classifier that performs better, a technique known as "*boosting*" [320]. In 1993, Ross Quinlan described C4.5 [292], an improved version of ID3 tackling some of its limitations. A year later, Leo Breiman had introduced the technique known as "*bagging*" (bootstrap aggregating), which allows to create ensembles of different ML classifiers, and which is extensively used along with decision trees [41, 42]. The next year, in 1995, random forests were described for the first time by Tin Kam Ho [152], an ensemble technique which is still widely used in 2017.

Despite the great advances in logical models, the 90s also brought many innovations to the field of artificial neural networks. In 1995, Gerald Tesauro introduced TD-Gammon, a backgammon software powered by artificial neural networks trained with temporal-difference learning which showed similar abilities than those of top human players [358]. Also that year, support vector machines were described by Corinna Cortes and Vladimir Vapnik [76], providing a substantial contribution in the field of geometric models. In 1997, a significant advancement in recurrent neural networks was achieved when Sepp Hochreiter and Jürgen Schmidhuber introduced long short-term memory (LSTM) [153], improving the efficiency of the recurrent component of neural networks, a technology that is still widely used nowadays. In 1998, the MNIST database was presented [204], which has become a standard benchmark for evaluating handwriting recognition, and by that year LeCun et al. had proposed one of the earliest developments of convolutional neural networks with an application to handwritten characters recognition [203], outperforming all other techniques.

On May 1997, IBM's Deep Blue beat world chess champion Garri Kaspárov [384]. This is not really a machine learning achievement, as Deep Blue was indeed a supercomputer performing very fast tree searches along with a minimax heuristic, yet it drew public attention on the status of AI.

In the early 2000s, machine learning techniques became solid enough and some libraries and projects arose for developers to easily implement ML techniques, such as Weka [137] in 2000 or Torch [74] in 2002. In the mid-2000s, the pervasiveness of e-commerce brought a new field of inter-

est: recommender systems. These are intelligent systems that aim at recommending to customers products they might be interested in according to their preferences, history of purchases or profile. Because of the interest in achieving high performance in recommender systems, Netflix launched in 2006 the Netflix Prize [265], by which they would award 1 million US dollars to the team who designed an algorithm able to improve the performance of the movie rating prediction problem (predicting the rating a user would give to a movie given his/her history of ratings) by a 10 % compared to their previous algorithm. The Netflix Prize was awarded on 2009. In 2010, Kaggle [169] is founded, providing a collaborative platform for machine learning competitions.

In the 2010s, we can find more developments and achievements that show a desire to mimic human intelligence. In 2011, IBM's Watson was able to beat two human champions in a Jeopardy! competition [230]. In 2010, ImageNet released the first annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a competition to classify and detect objects in images. By this moment, the availability of graphical processing units (GPUs) enabling for faster computation and training of neural networks led to the appearance of the field known as "*deep learning*". As a result, companies and teams willing to research in deep learning and transfer that research to specific applications were founded, such as DeepMind in 2010 [86] (which would be acquired by Google in 2014) or Google Brain in 2011 [121], led by Jeffrey Dean, Andrew Ng and Geoffrey Hinton.

In 2012, Krizhevsky, Sutskever and Hinton introduced AlexNet [190], a convolutional neural network that not only won the ILSVRC competition, but also improved the second best result by more than 10 percentual points, an astonishing achievement that; however, has been outperformed in subsequent editions. Also in 2012, the Google Brain team developed an algorithm able to recognize cat faces and human bodies in YouTube videos using deep learning [200]. The fact that deep learning was being used so extensively led to the appearance of specific deep learning libraries, such as Theano in 2010 [28], Caffe in 2014 [167] or TensorFlow in 2016 [1], and GPU vendors have developed specific primitives for deep learning, such as NVIDIA's cuDNN [62].

As of 2018, artificial intelligence and machine learning are producing remarkable achievements: self-driving vehicles are serving passengers in Pittsburgh, Pennsylvania [337], and automatic pilots are integrated in some cars manufacturers (e.g. Tesla). Deep learning techniques and the large availability of data are providing solutions for machine translation, speech recognition and automatic captioning, or image recognition and description. Machines are learning to play Atari games achieving the performance of human players [254], and have beaten Go champions Fan Hui [20,330] and Lee Sedol, just before learning from scratch how to beat itself [331].

While promising, the future of artificial intelligence and machine learning is yet to be written.

2.2 *Artificial Neural Networks*

In the previous section we have already mentioned artificial neural networks (ANNs). In this section, we will explain the biological foundations supporting artificial neural networks and then describe some of the most relevant neural networks models, including the Perceptron. The backpropagation algorithm for training a neural network will also be discussed in this section.

2.2.1 *History: A Note on Connectionism*

The interest of developing artificial neural networks in the 1950s emerged as a response to a new paradigm known as "*connectionism*", first mentioned by Edward Lee Thorndike in 1932 [361]. After conducting psychology experiments with animals, Thorndike showed that animals were able to

learn by trial and error, and the way of learning was consistent between different species, just at different speeds [360]. He concluded that learning occurred due to associations (or connections) that were formed between acts, situations and responses.

Despite of Thorndike coining the term, some of the ideas behind his work had been previously studied by Santiago Ramón y Cajal, who was awarded with the Nobel Laureate in 1906 in recognition of his work studying the structure of the nervous system [294,295]. Even before, in the 19th century, some scientists had already proposed theories that could be considered connectionists, such as Herbert Spencer in 1872 [339] or Sigmund Freud in 1895 [107].

However, it was Donald Hebb who later developed a formal mathematical model on learning based on the ideas of connectionism, popularizing them to the extent that some are known as "Hebbian theory". In his most relevant work, *The Organization of Behavior* [146], published in 1949, Hebb explained the relationship between the biological function of the brain with the functioning of the mind and the human behavior. His model for learning, which is sometimes known as Hebb's rule, was stated by himself as follows:

> *"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."*

In other words, when one neuron takes part in firing a different neuron, the weight of the connection between both neurons is strengthened. This rule serves as the building block for most of the implementations of artificial neural networks, as we will see throughout this section.

2.2.2 Biological Foundations

As the name suggests, artificial neural networks are inspired by the actual behavior of biological neural networks. The nervous system of animals, including humans, is the responsible of coordinating perception and actions, by transmitting signals from and to different parts of the body. It can be divided into two main parts: the central nervous system, which is composed of the brain and the spinal cord, and the peripheral nervous system, which comprises the nerves that carry impulses from and to the different parts of the body.

In the extreme of these nerves are receptors and effectors. Receptors are sensory structures in charge of perception, receiving input stimuli (either from the environment or from the body itself) and sending information about these through the nerves to the central nervous system, which will process this information and send corresponding impulses to the effectors, which can interpret them in the form of actions (either motor, hormonal, etc.)

The basic functional unit and building block of the nervous system is the neuron. Neurons are electrically excitable cells that receive information, and can transmit it to other neurons via electrochemical impulses. Because of these connections, neurons are grouped forming neural networks, which can store and process information. A diagram with the structure of a biological neuron cell can be found in figure 2.1.

In high level terms, the mechanism by which a biological neuron is able to receive, process and transmit information works as follow:

1. Input information is received from other neurons or receptors through the dendrites in the form of electrochemical impulses.

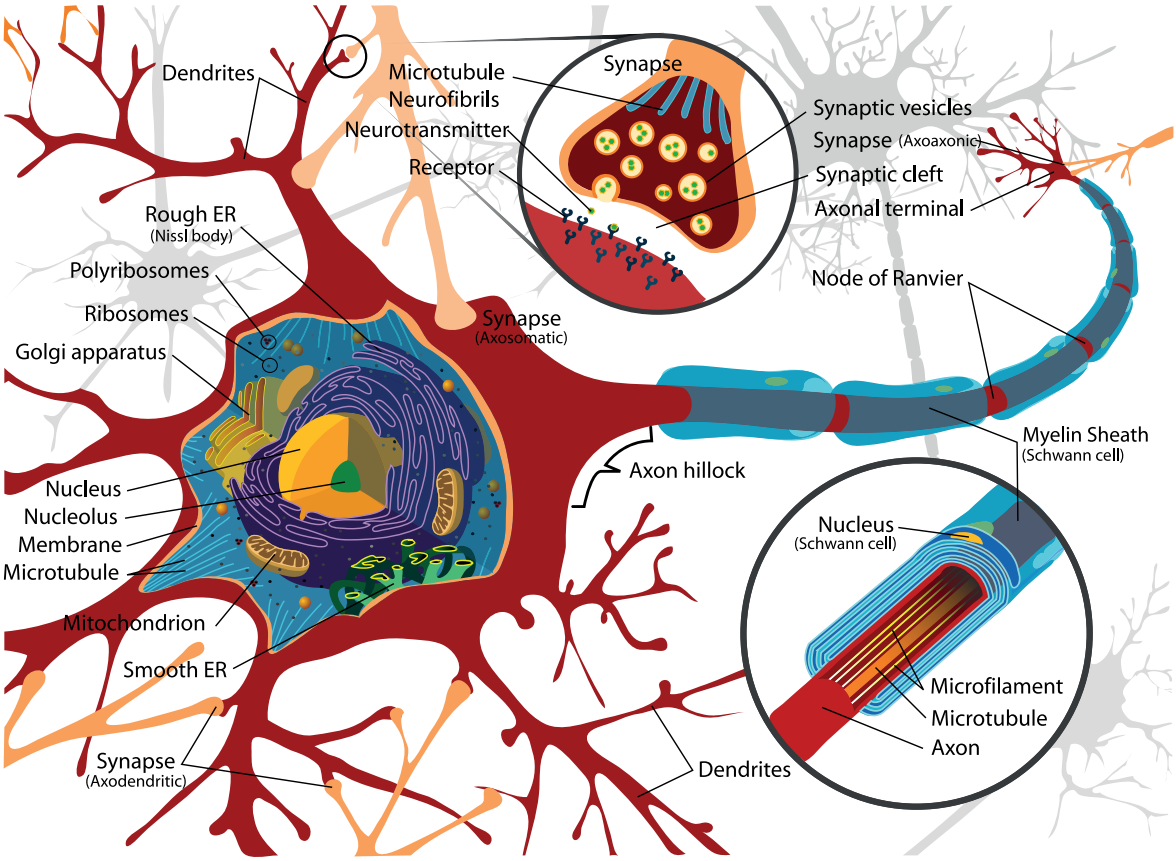


Figure 2.1: *Diagram of a neuron cell.*

2. If the neuron is activated, then an output response is generated, and the impulse flows through the neuron membrane until the axon.
3. The output response is propagated through the axon, a long connection (which can be as long as 1 meter in humans) which eventually branches at its end, establishing synaptic connections to dendrites in other neurons.

Synapses are the processes by which a connection between the axon of the neuron transmitting a signal and the dendrites of the neurons receiving that signal are interconnected. Synapses can belong to two main broad categories: chemical synapses or electrical synapses, the former kind being the predominant in mammals. At a chemical synapse, the arrival of an impulse from the axon triggers the release of neurotrasmitters, which interact with the receptors in the dendrites of the target neuron. The neuron membrane is enclosed by a plasma where different concentrations of ionic elements are present, such as potassium (K^+), sodium (Na^+) or calcium (Ca^+) ions, which can enter or leave the cell, producing an excitatory or inhibitory potential respectively. The axon is surrounded by the myelin sheath, a fatty substance forming an electrically insulating layer.

Since neuron cells are interconnected, they behave as a large network or mesh that propagates electrochemical impulses, while at the same time modifying the ionic concentration of neurons. Whether a neuron will activate or not depends both on the input signals received by the dendrites and the ionic concentration of the neuron, thus resembling a threshold function.

Despite artificial neural networks being inspired by biological neural networks, this inspiration is indeed vague. One remarkable difference between artificial and biological systems can be found

in their size: it is estimated that an average human brain has about 100 billion neurons [147, 390]. To make a rough comparison, this number is estimated to be in the same order of magnitude than the number of stars in the Milky Way [97]. Moreover, the number of neural connection is estimated in the order of 100 trillions [410]. On the other hand, the largest artificial neural network to date, presented by Digital Reasoning, involves about 160 billion connections [156], therefore being about 1,000 times smaller than an actual human neural system.

Moreover, the process by which neurons and neural connections evolve over time is also different in both cases. Despite the details of the learning process in the human brain still remain uncertain, it is likely a continuous process [350], whereas in current neural networks the process is discretized. More importantly, backpropagation, which is the most common process for learning parameters of artificial neural networks in a supervised learning scheme, relies on the availability of labeled data, so that a loss function of the output computed by the network and the real labels is computed and needs to be minimized. This process does not mimic the actual learning process carried out by humans, where labelled data may not be available.

2.2.3 The Artificial Neuron and Simple Neural Models

The artificial neuron is the building block, and simplest unit of computing of an artificial neural network. The basic structure of an artificial neuron is depicted in figure 2.2. In this diagram, $x_1, x_2, \ldots, x_n$ represent the n features of an instance, $w_1, w_2, \ldots, w_n$ represent weights assigned to the input connections, b is an additional weight (known as *bias*), f is the activation function of the neuron, and a is the activation value of the neuron.

The way in which the artificial neuron computes the a is as follows: First, all input values x_i are multiplied by the weight of the corresponding connection w_i. The neuron then computes the summation of all these products and adds the bias b. Finally, a certain activation function f is computed over the result. The computation of an artificial neuron is shown in equation 2.2:

$$a = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.2}$$

This computation can be expressed in terms of algebraic operations. To do so, let $\mathbf{x}$ and $\mathbf{w}$ be vectors of dimension $n \times 1$, as described in equation 2.3:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \tag{2.3}$$

Then, the summation of products $x_i w_i$ can be computed as the product of the transposition of the vector of weights by the vector of inputs; resulting in equation 2.4. A dimensionality analysis will prove that the product $\mathbf{w}^\intercal \mathbf{x}$ is a scalar:

$$a = f\left(\mathbf{w}^\intercal \mathbf{x} + b\right) \tag{2.4}$$

Despite equations 2.2 and 2.4 being equivalent, a vectorized implementation can often be computed much faster in modern computers than an iterative implementation of the summation operator, since vector and matrix multiplications can compute the product of different terms in parallel.

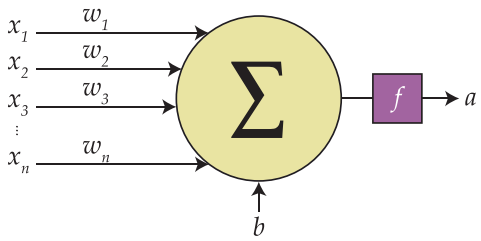


Figure 2.2: *Diagram of an artificial neuron.*

Whereas neural networks can be very different to each other in size, connectivity patterns, activation functions, etc.; this structure of an artificial neuron remains mostly invariant. In its simplest form, a neural network will only comprise one neuron. In this case, the only parameters of the network will be the weights, the bias, and the activation function. Weights and biases are parameters that are learned through a process that will be described later in this chapter. On the other hand, the activation function is a hyperparameter that is determined beforehand. Depending on the activation function, we can find several different types of single-neuron networks; e.g., the McCulloch-Pitts cell, the Perceptron, the Adaline, or logistic regression.

2.2.3.1 McCulloch-Pitts Cell

One of the earliest computational approximations to an artificial neuron was described by McCulloch and Pitts in 1943 [234]. To model the behavior of neurons, they described a threshold activation function, like the one described in equation 2.5:

$$a = \begin{cases} 1, & \text{if } \mathbf{w}^\intercal \mathbf{x} + b > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

In the original McCulloch-Pitts cell, both the inputs and the output are binary. Interestingly, the cell can perfectly model the NOT, AND and OR logical functions. In particular, NOT can be represented with $w_1 = -1, b = +1$, AND with $w_1 = +1, w_2 = +1, b = -1$ and OR with $w_1 = +1, w_2 = +1, b = 0$. Details on how the inputs are mapped to an output for representing these functions are described in table 2.1.

Because logical functions can be represented with a McCulloch-Pitts cell, we can guarantee that a network of interconnected cells can potentially represent any digital circuit, and therefore can represent a universal Turing machine. However, the number of cells could be very large, leading to a very large number of parameters (weights and biases) and making the implementation infeasible.

x_1	$\mathbf{w}^\intercal \mathbf{x} + b$	a
0	1	1
1	0	0

(a) *NOT function*

x_1	x_2	$\mathbf{w}^\intercal \mathbf{x} + b$	a
0	0	-1	0
0	1	0	0
1	0	0	0
1	1	1	1

(b) *AND function*

x_1	x_2	$\mathbf{w}^\intercal \mathbf{x} + b$	a
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	1

(c) *OR function*

Table 2.1: *Basic logical functions computed with one McCulloch-Pitts cell.*

2.2.3.2 Perceptron

The Perceptron was introduced in the late 1950s by Rosenblatt [307], and was aimed at, given a set of examples belonging to different classes, automatically learning the boundaries between those classes. Like in McCulloch-Pitts cells, the Perceptron activation function is also a threshold function, described in equation 2.6:

$$a = \begin{cases} 1, & \text{if } \mathbf{w}^\intercal \mathbf{x} + b > 0 \\ -1, & \text{otherwise} \end{cases} \tag{2.6}$$

Given an input instance $\mathbf{x}$ and a binary classification problem, we consider that $\mathbf{x}$ belong to the first class if the output is 1, and to the second class if the output is -1. As in the McCulloch-Pitts cell, the Perceptron can learn the basic logical functions NOT, AND, and OR. Notice that because of the shape of the neuron activation function, learning the weights of a Perceptron is equivalent to learning the coefficients of a hyperplane of n dimensions. Because it represents a hyperplane, for the Perceptron to work, data of different classes must be linearly separable.

While this simple model raised significant attention in the 1950s and 1960s, its inability to learn complex models discriminating non-linearly separable instances, such as the exclusive OR problem (XOR), led to the AI Winter in the 1970s.

2.2.3.3 Adaline

One limitation of the Perceptron is that it was unable to approximate real outputs and therefore cannot be used for solving regression problems. In 1960, only three years after the Perceptron was introduced, Bernard Widrow presented the Adaptive Linear Neuron, or Adaline for short [389]. The Adaline has a linear activation function, shown in equation 2.7, thus enabling a real output:

$$a = \mathbf{w}^\intercal \mathbf{x} + b \tag{2.7}$$

Despite this system being able of approximating real functions, as in the case of the Perceptron it is unable to learn a model to approximate the XOR function.

2.2.3.4 Logistic Regression

Despite the name, logistic regression is a technique that is often used for classification; i.e., assigning one instance to one class from a discrete set. The structure of a basic logistic regression architecture with a single neuron is equivalent to the previous models; however, in this case the sigmoid function is used, which is shown in equation 2.8:

$$a = \sigma\left(\mathbf{w}^\intercal \mathbf{x} + b\right) \qquad \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.8}$$

The sigmoid function is also known as logistic, therefore the name. As we will see later in this chapter, the domain of this function is $(-\infty, \infty)$ and its range is $(0, 1)$; so that when $z = 0$, then $\sigma(z) = 0.5$. Because of this, in a binary classification problem with classes $c = \{0, 1\}$, the output of the sigmoid function can be interpreted as the probability of the input instance i belonging to

class 1. Thus, we could conclude that if $\sigma(\mathbf{w}^\intercal \mathbf{x}^{(\mathbf{i})} + b) > 0.5$, then instance i has a higher probability of belonging to class 1; whereas if $\sigma(\cdot) < 0.5$, then i is more likely to belong to class 0.

2.2.4 *A Note on m Input Instances*

Until now, we have described the process of having only one input instance. However, in the general case of a supervised learning problem, we will have m instances, each one with the form $(\mathbf{x}^{(\mathbf{i})}, y^{(i)})$, with $\mathbf{x}^{(\mathbf{i})}$ being the vector of features and $y^{(i)}$ being the output or class.

So far, we have agreed on the fact that the activation value of an input instance i can be computed as $a^{(i)} = f(\mathbf{w}^\intercal \mathbf{x}^{(\mathbf{i})} + b)$. Now, let us assume that we build a matrix $\mathbf{X}$ containing all vectors $\mathbf{x}^{(\mathbf{i})}$ disposed in columns, as shown in equation 2.9:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}^{(\mathbf{1})} & \mathbf{x}^{(\mathbf{2})} & \cdots & \mathbf{x}^{(\mathbf{m})} \\ | & | & & | \end{bmatrix} \tag{2.9}$$

Here, $\mathbf{X}$ would be an $n \times m$ matrix, where n is the number of features of each input instance. At this point, it is worth remembering that $\mathbf{w}$ is the vector of weights for the input connections, and has size $n \times 1$. Now, let us compute the matrix product $\mathbf{w}^\intercal \mathbf{X}$, shown in equation 2.10:

$$\mathbf{w}^\intercal \mathbf{X} = \begin{bmatrix} w_1 & \cdots & w_n \end{bmatrix} \begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(m)} \\ \vdots & \ddots & \vdots \\ x_n^{(1)} & \cdots & x_n^{(m)} \end{bmatrix} = \begin{bmatrix} \mathbf{w}^\intercal \mathbf{x}^{(\mathbf{1})} & \cdots & \mathbf{w}^\intercal \mathbf{x}^{(\mathbf{m})} \end{bmatrix} \tag{2.10}$$

Of course, a dimensionality analysis allows us to conclude that the product of a $1 \times n$ vector by an $n \times m$ matrix is a $1 \times m$ vector. Now, if we sum the bias b to each element in the output vector and compute function f in an element-wise manner, then the final result would be a $1 \times m$ vector where position i represents the activation value for $\mathbf{x}^{(\mathbf{i})}$, as shown in equation 2.11:

$$\mathbf{a} = f\left(\mathbf{w}^\intercal \mathbf{X} + \mathbf{b}\right) = \begin{bmatrix} f\left(\mathbf{w}^\intercal \mathbf{x}^{(\mathbf{1})} + \mathbf{b}\right) & \cdots & f\left(\mathbf{w}^\intercal \mathbf{x}^{(\mathbf{m})} + \mathbf{b}\right) \end{bmatrix} = \begin{bmatrix} a^{(1)} & \cdots & a^{(m)} \end{bmatrix} \tag{2.11}$$

At this point, two things are worth noting: First, the computation of $\mathbf{w}^\intercal \mathbf{X} + b$ involves the addition of a vector with a scalar, which cannot be performed. Instead, we have used $\mathbf{b}$ to represent a vector of identical dimension than $\mathbf{w}^\intercal \mathbf{X}$ full of b elements. Some programming languages and libraries call this expansion "*broadcasting*" and perform it by default. Second, the computation of the activation values for each instance could be computed one-by-one; however, it is more efficient in most modern computers to compute the vectorized implementation.

2.2.5 *Gradient Descent for Learning Parameters*

So far we have described how to compute the activation value a given a vector of features $\mathbf{x}$ and the neuron parameters $\mathbf{w}$ and b. However, it is worth recalling that an instance has an output

value y, which we want to approximate with the neural computation. To do so, we must learn the parameters of the neural network in order to reduce a given loss function, which determines how much "right" or "wrong" the computed output is with respect to the real output. This process is also called "training" of the neural network.

Formally, let us call $\hat{y}^{(i)}$ the output of the neural network for input $\mathbf{x^{(i)}}$, while $y^{(i)}$ is the real output. It shall be noticed that $\hat{y}^{(i)} = a^{(i)}$. Given a set of instances, we want $\hat{y}^{(i)} \simeq y^{(i)}$. The loss function will be of the form $\mathcal{L}(\hat{y}, y)$ and will determine how *wrong* $\hat{y}$ is with respect to y.

The loss function $\mathcal{L}$ could potentially be any function measuring the error, such as the squared error. However, one of the most common loss functions used is the cross entropy, described in equation 2.12. This function results in an optimization problem which is convex, preventing the training process to fall in local minima:

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log (1 - \hat{y})) \tag{2.12}$$

Now, from the perspective of a binary classification problem, we can explain the behavior of this loss function. Let us simplify the previous definition of the loss function for the two scenarios, resulting in the expression shown in equation 2.13:

$$\mathcal{L}(\hat{y}, y) = \begin{cases} -\log \hat{y}, & \text{if } y = 1 \\ -\log (1 - \hat{y}), & \text{if } y = 0 \end{cases} \tag{2.13}$$

It can be seen that, because the loss function is to be minimized, when $y = 1$ we want $\hat{y}$ to be as large as possible, and when $y = 0$ then $\hat{y}$ will have to be as small as possible. In the case of logistic regression, for example, $\hat{y}$ will always be a number between 0 and 1.

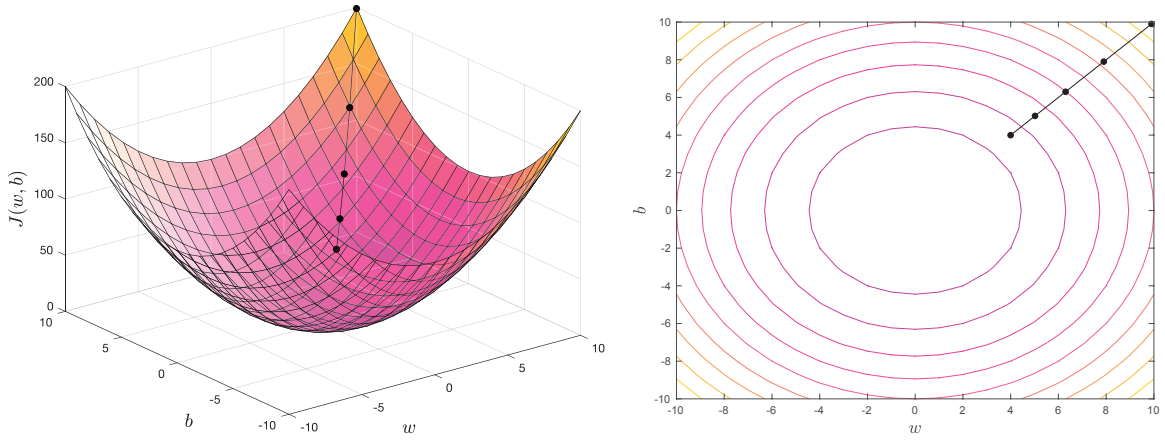
While the loss function determines the error for a single instance, an additional concept arises if we want to compute the error over the whole training set: the cost function. This cost function is essentially the average of the loss function across all instances, as shown in equation 2.14:

$$\mathcal{J}(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \log \left(1 - \hat{y}^{(i)}\right)\right] \tag{2.14}$$

Now, once $\hat{\mathbf{y}} = \mathbf{a}$ is obtained, we can compute the cost value following equation 2.14. Then, we need to adjust the weights $\mathbf{w}$ and the bias b in order to reduce this cost value.

This process of adjusting the network parameters in order to minimize the cost is known as "gradient descent". The principle behind gradient descent is as follows: in each iteration, the parameters are moved in the direction of the gradient of the cost function, thus reducing the cost value. An example of the gradient descent process after five iterations is shown in figure 2.3. The figure in the left shows the surface of a sample loss function with only two parameters (w and b), and the dots represent the changes in the loss value over time. The image on the right is an equivalent representation using a contour diagram.

To update the value of a parameter w_j, we will compute the gradient, which is the derivative of the cost function with respect to w_j. Then, we will move that parameter in the negative direction of the gradient, in order to decrease the value of the cost function. To do so, we will subtract the gradient from the previous value of the parameter w_j, as shown in equation 2.15:



(a) *Surface.* **(b)** *Contour.*

Figure 2.3: *Illustration of gradient descent over a 2-dimensional function.*

$$w_j \leftarrow w_j - \eta \frac{\partial \mathcal{J}(\mathbf{w}, b)}{\partial w_j} \tag{2.15}$$

The intuition behind the previous equation is the following: the derivative is equivalent to the slope of the function at the current value of the parameter w_j. If the slope is negative, that means that increasing the value of w_j will decrease the value of the loss function. On the other hand, if the slope is positive that means that the value of w_j must decrease for the cost function to decrease as well. Additionally, a hyperparameter $\eta > 0$ called the "learning rate" is introduced to control how much the parameter will change. The larger the value of η, the more the parameter will shift towards the minimum. However, very large values of the learning rate could prevent the gradient descent algorithm from converging.

Now, given a single training instance and assuming a logistic regression model, let us describe the formulation for upgrading a certain parameter w_j. It is worth noting that w_j may be a weight in $\mathbf{w}$, but could also be the bias b. First, we can compute the derivative of the loss function with respect to the output of the neural network, as shown in equation 2.16:

$$\frac{\partial \mathcal{L}(a, y)}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a} \tag{2.16}$$

Then, we can compute the derivative of a with respect to the value $\mathbf{w}^\intercal \mathbf{x} + b$, which we will call z for the sake of simplicity, as shown in equation 2.17:

$$\frac{\partial a}{\partial z} = \frac{\partial \sigma(z)}{\partial z} = \frac{\partial \frac{1}{1+e^{-z}}}{\partial z} = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \frac{e^{-z}}{1+e^{-z}} = a(1-a) \tag{2.17}$$

Finally, we can compute the derivative of z with respect to the weight w_j, which results in the simple differentiation shown in equation 2.18:

$$\frac{\partial z}{\partial w_j} = \frac{\partial \left(\mathbf{w}^\intercal \mathbf{x} + b\right)}{\partial w_j} = \frac{\partial \left(w_1 x_1 + w_2 x_2 + \cdots + w_j x_j + \cdots + w_n x_n + b\right)}{\partial w_j} = x_j \tag{2.18}$$

It is worth noting that if $w_j = b$, then the derivative would the constant value 1.

To end with the computation of the derivative required for gradient descent as specified in equation 2.15, we can apply the chain rule to compute the derivative of the loss function with respect to the individual weight w_j, which after introducing the values previously obtained in equations 2.16, 2.17 and 2.18 results in the expression shown in equation 2.19:

$$\frac{\partial \mathcal{L}(a,y)}{\partial w_j} = \frac{\partial \mathcal{L}(a,y)}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w_j} = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right)(a(1-a))\,x_j = (a-y)x_j \tag{2.19}$$

By plugging this derivative into the gradient descent formula shown in equation 2.15, we conclude that the rule for updating weights is the one shown in equation 2.20, which separates the case for a weight in $\mathbf{w}$ and for the bias b:

$$\begin{aligned} w_j &\leftarrow w_j - \eta(a-y)x_j \\ b &\leftarrow b - \eta(a-y) \end{aligned} \tag{2.20}$$

So far, we have explained the process for a single training instance, thus computing the derivative of the loss function with respect to each parameter. If we had several training examples, then the derivative of the cost function $J(\mathbf{w}, b)$ should be computed instead. From equation 2.14 we know that the cost function is the average loss for all the training instances. In consequence, the derivative can be computed as expressed in equation 2.21:

$$\frac{\partial \mathcal{J}(\mathbf{w},b)}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}\mathcal{L}(a^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)})x_j^{(i)} \tag{2.21}$$

To recap, this whole computation could be expressed in terms of vectors and matrices operations following the next steps. First, we will compute the activation values for all the instances of the training set, following equation 2.22:

$$\mathbf{a} = \sigma(\mathbf{w}^\intercal \mathbf{X} + \mathbf{b}) \tag{2.22}$$

Then, the derivative of the cost function with respect to all the weights and to the bias can be computed by following equation 2.23:

$$\begin{aligned} \frac{\partial \mathcal{J}(\mathbf{w},b)}{\partial \mathbf{w}} &= \frac{1}{m}\mathbf{X}\,(\mathbf{a}-\mathbf{y})^\intercal \\ \frac{\partial \mathcal{J}(\mathbf{w},b)}{\partial b} &= \frac{1}{m}\mathbf{1}\,(\mathbf{a}-\mathbf{y})^\intercal \end{aligned} \tag{2.23}$$

In this expression $\mathbf{X}$ is an $n \times m$ matrix, both $\mathbf{a}$ and $\mathbf{y}$ are $1 \times m$ vectors, and $\mathbf{1}$ represents a $1 \times m$ vector of ones. As a result, the derivative with respect to $\mathbf{w}$ will be an $n \times 1$ vector and the derivative with respect to b will be a scalar. A vectorized implementation of the parameters upgrade consistent with equation 2.15 is shown in equation 2.24:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta \frac{\partial \mathcal{J}(\mathbf{w}, b)}{\partial w} \\ b &\leftarrow b - \eta \frac{\partial \mathcal{J}(\mathbf{w}, b)}{\partial b} \end{aligned} \tag{2.24}$$

2.2.6 The Multi-Layer Perceptron

The multilayer perceptron (MLP) is one of the most widely used artificial neural networks. It is a generalization of the Perceptron that can comprise several hidden layers (i.e., layers different from the input layer) with a variable number of neurons each, and with a fully connected topology, where there are connections between all the neurons in two consecutive layers, each of these connection having an associated weight. The multilayer perceptron, under certain conditions, can act as a universal function approximator [78], meaning that it can approximate any function.

When describing the structure of an MLP, we will use the following convention: L is the number of layers of the MLP, whereas $n^{[l]}$ is the number of neurons in the l-th layer. L is also known as the "depth" of the neural network, thus deeper networks are those involving a larger number of layers, and shallower networks are those with less layers.

A sample structure of a multilayer perceptron containing two hidden layers ($L = 2$), with four neurons in the first layer ($n^{[1]} = 4$) and one neuron in the second layer ($n^{[2]} = 1$) is shown in figure 2.4. In the figure, $x_j^{(i)}$ refers to the j-th feature of the i-th input instance, $a_j^{[l]}$ is the activation value for the j-th neuron of the l-th layer, and $\hat{y}^{(i)}$ is the output value for the i-th input instance.

Given a multilayer perceptron and an input matrix $\mathbf{X}$, we can compute the output matrix $\hat{\mathbf{Y}}$ by carrying out the process of forward propagation through all the different layers of the neural network. Just as we described earlier in section 2.2.4, $\mathbf{X}$ will be an $n \times m$ matrix, where n is the number of features and m is the number of instances.

Let us denote $\mathbf{W}^{[1]}$ as the matrix of weights for the connections between layers $l-1$ and l. Because layer $l-1$ has $n^{[l-1]}$ neurons and layer l has $n^{[l]}$ neurons, the matrix $\mathbf{W}^{[1]}$ will be an

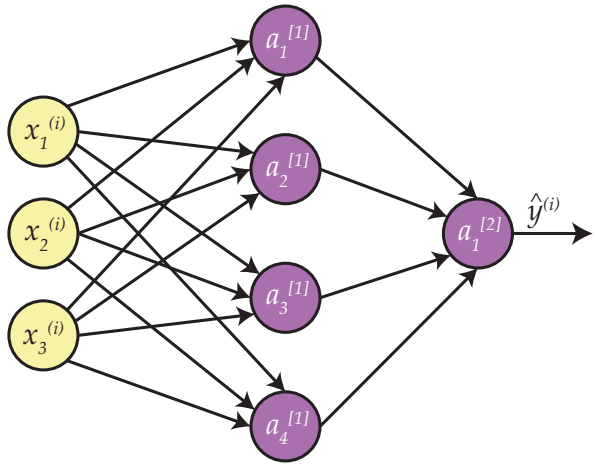


Figure 2.4: *Multilayer perceptron.*

$n^{[l]} \times n^{[l-1]}$ matrix. Therefore, if we denote $w_{j,i}^{[l]}$ to the weight of the connection from neuron i in layer $l-1$ to neuron j in layer l, then $\mathbf{W^{[l]}}$ is a matrix as shown in equation 2.25:

$$\mathbf{W^{[l]}} = \begin{bmatrix} w_{1,1}^{[l]} & \cdots & w_{1,n^{[l-1]}}^{[l]} \\ \vdots & \ddots & \vdots \\ w_{n^{[l]},1}^{[l]} & \cdots & w_{n^{[l]},n^{[l-1]}}^{[l]} \end{bmatrix} \tag{2.25}$$

Additionally, we will consider $\mathbf{b^{[l]}}$ to be vector of biases for the neurons in layer l, which is an $n^{[l]} \times 1$ vector as shown in equation 2.26:

$$\mathbf{b^{[l]}} = \begin{bmatrix} b_1^{[l]} \\ \cdots \\ b_{n^{[l]}}^{[l]} \end{bmatrix} \tag{2.26}$$

With this notation in mind, we can express the computation of the activation values in layer l for instance i following the expression shown in equation 2.27:

$$\begin{aligned} \mathbf{a^{[l](i)}} &= f\left(\mathbf{W^{[l]}}\mathbf{a^{[l-1](i)}} + \mathbf{b^{[l]}}\right) = f\left(\begin{bmatrix} w_{1,1}^{[l]} & \cdots & w_{1,n^{[l-1]}}^{[l]} \\ \vdots & \ddots & \vdots \\ w_{n^{[l]},1}^{[l]} & \cdots & w_{n^{[l]},n^{[l-1]}}^{[l]} \end{bmatrix} \begin{bmatrix} a_1^{[l-1](i)} \\ \vdots \\ a_{n^{[l-1]}}^{[l-1](i)} \end{bmatrix} + \begin{bmatrix} b_1^{[l]} \\ \vdots \\ b_{n^{[l]}}^{[l]} \end{bmatrix}\right) \\ &= \begin{bmatrix} f\left(w_{1,1}^{[l]} a_1^{[l-1](i)} + \cdots + w_{1,n^{[l-1]}}^{[l]} a_{n^{[l-1]}}^{[l-1](i)} + b_1^{[l]}\right) \\ \vdots \\ f\left(w_{n^{[l]},1}^{[l]} a_1^{[l-1](i)} + \cdots + w_{n^{[l]},n^{[l-1]}}^{[l]} a_{n^{[l-1]}}^{[l-1](i)} + b_{n^{[l]}}^{[l]}\right) \end{bmatrix} = \begin{bmatrix} a_1^{[l](i)} \\ \vdots \\ a_{n^{[l]}}^{[l](i)} \end{bmatrix} \end{aligned} \tag{2.27}$$

As a result of this computation, $\mathbf{a^{[l]}}$ would be a vector of dimension $n^{[l]} \times 1$. Also, the activation function of layer 0 (the input layer) corresponds to the value of the input, so that $\mathbf{a^{[0](i)}} = \mathbf{x^{(i)}}$.

Now, if we wanted to compute the activation values for layer l across all instances, we could rewrite the previous expression as shown in equation 2.28, where $\mathbf{A^{[0]}} = \mathbf{X}$ and $\mathbf{B^{[l]}}$ is the broadcast version of vector $\mathbf{b^{[l]}}$, to match the dimensions $n^{[l]} \times m$:

$$\mathbf{A^{[l]}} = f\left(\mathbf{W^{[l]}}\mathbf{A^{[l-1]}} + \mathbf{B^{[l]}}\right) \tag{2.28}$$

To conclude forward propagation, we must realize that the matrix $\hat{\mathbf{Y}}$, with dimensions $n^{[L]} \times m$, is the result of computing the activation values for the last layer, as shown in equation 2.29:

$$\hat{\mathbf{Y}} = \mathbf{A^{[L]}} \tag{2.29}$$

To learn the parameters of the network, we will carry out backpropagation. Let us define $\mathbf{Z}^{[\mathbf{l}]}$ so that $\mathbf{A}^{[\mathbf{l}]} = f\left(\mathbf{Z}^{[\mathbf{l}]}\right)$. We can then compute the derivatives with respect to the weights and biases of each layer following equation 2.30:

$$\begin{aligned} \frac{\partial \mathcal{J}}{\partial \mathbf{W}^{[\mathbf{l}]}} &= \frac{1}{m} \frac{\partial \mathcal{J}}{\partial \mathbf{Z}^{[\mathbf{l}]}} \mathbf{A}^{[\mathbf{l-1}]^\intercal} \\ \frac{\partial \mathcal{J}}{\partial \mathbf{b}^{[\mathbf{l}]}} &= \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{J}}{\partial \mathbf{Z}^{[\mathbf{l}](\mathbf{i})}} \end{aligned} \tag{2.30}$$

The derivative of the cost with respect to $\mathbf{Z}^{[\mathbf{l}]}$ is described in equation 2.31, where $\odot$ refers to the matrix element-wise multiplication operator and f' is the derivative of the activation function[2]:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{Z}^{[\mathbf{l}]}} = \frac{\partial \mathcal{J}}{\partial \mathbf{A}^{[\mathbf{l}]}} \odot f'\left(\mathbf{Z}^{[\mathbf{l}]}\right) \tag{2.31}$$

To compute the derivative of the cost with respect to the activation value of layer $l-1$ given the parameters of backpropagation computed for the following layer, we can apply equation 2.32:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{A}^{[\mathbf{l-1}]}} = \mathbf{W}^{[\mathbf{l}]^\intercal} \frac{\partial \mathcal{J}}{\partial \mathbf{Z}^{[\mathbf{l}]}} \tag{2.32}$$

Finally, to initialize backpropagation we need to compute the derivative of the cost function with respect to the activation values of the last layer (i.e., the output). To do so, first we will compute the values of the cost function $\mathcal{J}$ for all instances given $\mathbf{Y}$ and $\hat{\mathbf{Y}} = \mathbf{A}^{[\mathbf{L}]}$. If the chosen loss function $\mathcal{L}$ is the cross entropy, then the expression for the cost function $\mathcal{J}$ was provided in equation 2.14. Then, the derivative is computed following equation 2.33:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{A}^{[\mathbf{L}]}} = -\frac{\mathbf{Y}}{\mathbf{A}^{[\mathbf{L}]}} + \frac{1-\mathbf{Y}}{1-\mathbf{A}^{[\mathbf{L}]}} \tag{2.33}$$

To conclude backpropagation, we can adjust the parameters of the network (weights and biases for each layer) following the update rules in equation 2.34:

$$\begin{aligned} \mathbf{W}^{[\mathbf{l}]} &\leftarrow \mathbf{W}^{[\mathbf{l}]} - \eta \frac{\partial \mathcal{J}}{\partial \mathbf{W}^{[\mathbf{l}]}} \\ \mathbf{b}^{[\mathbf{l}]} &\leftarrow \mathbf{b}^{[\mathbf{l}]} - \eta \frac{\partial \mathcal{J}}{\partial \mathbf{b}^{[\mathbf{l}]}} \end{aligned} \tag{2.34}$$

[2]Although is not a common practice, the MLP can have a different activation function in each of its layers. As a result, we could use the notation $f^{[l]\prime}$ to refer to the derivative of the activation function in layer l.

2.3 Deep Learning

The term "deep learning" is relatively new (at least, much newer than the concept of neural networks itself) and was probably popularized after the publication of a paper by Yoshua Bengio, considered one of the fathers of this discipline, entitled "Learning Deep Architectures for AI" [25]. Bengio is also coauthor, along with Goodfellow and Courville of the book "Deep Learning" [118] released in 2016 and which constitutes one of the main works of reference for getting in touch with this field. According to Bengio:

> *"Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Automatically learning features at multiple levels of abstraction allow a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features."*

In other words, the purpose of deep learning is to automatically learn features with several degrees of abstraction. The field studying the development of techniques for such objective is known as "representation learning" or "feature learning" [26], and the ultimate purpose is to be able to replace the human-engineered process of feature design.

So far, the neural network models described in the previous section had little or none to do with representation learning, since they were able to learn a mapping functions from the input to the expected output, but unable to learn higher level features from the input. In fact, for years the common input to neural network such as a multilayer perceptron was often a set of manually-engineered features. However, this trend has changed since the invention of convolutional neural networks, that will be described later in section 2.5.

Additionally, deep learning has popularized in the era of big data. In the words of Bengio [25]:

> *"The ability to automatically learn powerful features will become increasingly important as the amount of data and range of applications to machine learning methods continues to grow."*

It is generally acknowledged that the ability of deep learning to learn reliable and successful models increases as more data are available. For this reason, the availability of larger amounts of data in recent years, as well as the advances in artificial intelligence and the improvement of hardware devices, have made it possible to apply deep learning to leverage fields such as image recognition, speech recognition or image translation, attaining performances far above those that were reported before deep learning was applied.

2.3.1 The Depth in Deep Learning

Also, Bengio [25] further described the concept of "depth" in deep learning, establishing an association between this concept and the structure of the mammal brain and visual system:

> *"Depth of architecture refers to the number of levels of composition of non-linear operations in the function learned. Whereas most current learning algorithms correspond to shallow architectures (1, 2 or 3 levels), the mammal brain is organized in a deep architecture [326] with a given input percept represented at multiple levels of abstraction, each level corresponding to a different area of cortex. Humans often describe such concepts in hierarchical ways, with multiple levels of abstraction. The brain also appears to process information through multiple*

> *stages of transformation and representation. This is particularly clear in the primate visual system [326], with its sequence of processing stages: detection of edges, primitive shapes, and moving up to gradually more complex visual shapes."*

This definition establishes several important concepts related to deep learning. First, when we refer to deep learning we are often referring to deep neural networks, although other techniques are also considered within the field. As stated by Bengio, in the past most artificial neural networks involved what he calls a "shallow" architecture; i.e., a small number of layers. As opposed to a shallow neural network, a deep neural network would comprise a larger number of layers, each of them performing a non-linear operation over its input.

The fact that operations are required to be non-linear can be easily explained mathematically. Remember that the output of one layer is defined as expressed by equation 2.35:

$$\mathbf{A}^{[\mathbf{l}]} = f\left(\mathbf{W}^{[\mathbf{l}]}\mathbf{A}^{[\mathbf{l-1}]} + \mathbf{b}^{[\mathbf{l}]}\right) \tag{2.35}$$

In the previous equation, $\mathbf{W}^{[\mathbf{l}]}$ and $\mathbf{b}^{[\mathbf{l}]}$ are the matrix of weights and the vector of biases in layer l respectively, $\mathbf{A}^{[\mathbf{l}]}$ is the matrix of activation values from layer l, and f is the activation function.

Now, let us consider f a linear function. To simplify the mathematical development, we will assume f is the identity function $f(x) = x$ and consider all the bias parameters to be zero. Also, it is worth recalling that $A^{[0]} = X$. Then, the output of a neural network with an identity activation function can be computed as described in equation 2.36:

$$\hat{\mathbf{Y}} = \mathbf{A}^{[\mathbf{L}]} = \mathbf{W}^{[\mathbf{L}]}\mathbf{A}^{[\mathbf{L-1}]} = \mathbf{W}^{[\mathbf{L}]}\mathbf{W}^{[\mathbf{L-1}]}\mathbf{A}^{[\mathbf{L-2}]} = \mathbf{W}^{[\mathbf{L}]}\mathbf{W}^{[\mathbf{L-1}]}\dots\mathbf{W}^{[\mathbf{1}]}\mathbf{X} \tag{2.36}$$

Because of the associative property of matrix multiplication, if we denote $\mathbf{W} = \mathbf{W}^{[\mathbf{L}]}\mathbf{W}^{[\mathbf{L-1}]}\dots\mathbf{W}^{[\mathbf{1}]}$, then we can express the output of the L layers neural network as $\hat{\mathbf{Y}} = \mathbf{W}\mathbf{X}$; in other words: for any neural network with an arbitrary number of layers L and a linear activation function, an equivalent network can be found comprising only one hidden layer. Therefore, a linear function is unable to accurately model deep representations.

Finally, it is remarkable how Bengio relates the problem of deep learning with the physiological features of the mammal brain and visual system. The relationship between artificial neural networks and biological neural networks was addressed earlier in this chapter, in section 2.2.2; and despite both posing significant differences in their structure and their learning mechanism, Bengio seems to describe a trend to design more complex structures that would approach those present in the brain. Additionally, convolutional layers, that will be described in section 2.5, hold some similarities to the structure of the mammal primary visual cortex [158].

2.3.2 *The Vanishing or Exploding Gradient Problem*

According to Bengio [25], no successful attempts at learning a deep neural network were reported before 2006, despite the fact that the backpropagation process had been presented two decades earlier, in 1986. This can be explained in part due to the lack of computational resources to train complex architectures; however, that cannot be the sole reason since deeper models were indeed trained and yielded poorer results.

An important problem when training deep networks is the vanishing or exploding gradient problem. To illustrate this problem, let us define a very simple model with L layers, compris-

ing n neurons each, an identity activation function, no biases and whose weights matrices are all equivalent and described by a diagonal matrix, shown in equation 2.37:

$$\mathbf{W}^{[1]} = \begin{bmatrix} v & 0 & \cdots & 0 \\ 0 & v & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & v \end{bmatrix} \tag{2.37}$$

Now, we know from the previous section that with this architecture, the output of the network can be computed as $\hat{\mathbf{Y}} = \mathbf{W}^{[\mathbf{L}]}\mathbf{W}^{[\mathbf{L-1}]} \dots \mathbf{W}^{[\mathbf{1}]}\mathbf{X}$. Since all weights matrices are equivalent, we could compute the output as $\hat{\mathbf{Y}} = \mathbf{W}^{\mathbf{L}}\mathbf{X}$.

Because of the exponentiation of the weight matrix to the number of layers, the following effect will occur: If $v > 1$, then as the number of layers increases, the final value for the weight will grow exponentially resulting in a very large value. Conversely, if $v < 0$, with a sufficiently large number of layers the final value will be very close to zero. Therefore, the output will approach extreme values, such as zero or infinite, as the number of layers increases. In fact, this value increases or decreases exponentially with respect to the number of layers.

This effect not only occurs in forward propagation, but also during backpropagation. In that case, very large values of the gradient lead to "the exploding gradient problem" and could modify the parameters in the first layers of the network so abruptly as to prevent convergence. On the other hand, a very small gradient would be known as "the vanishing gradient problem", and would let the parameters in the early layers mostly unaffected. While this problem is common across different activation functions, in recent years specific techniques have been developed (such as weight initialization algorithms, activation functions or optimizers), reducing the consequences of these problems significantly.

2.4 Tensor-based Representation of Data

When we described classical neural networks architectures and the multilayer perceptron, we showed that the input to the networks comprised several instances, each one defined by a vector of n features. However, most of the data we can find nowadays are structured in more than one dimension. For example, if we wanted to accomplish image recognition (classifying an image based on its content), then our data would comprise two dimensions (width and height) and probably three channels, if it were an RGB-encoded image. However, we should be able to convert this rich structure into a vector in order to be able to pass the data through the neural network.

Of course, one option would be to unroll the data into a vector. Thus, if we had a 640×480 RGB image, we could convert it to a vector of 921,600 bits, which is the product of the width, the height and the number of channels. However, this approach is impractical due to a couple reasons.

First, this approach leads to a very high dimensionality, which will increase significantly the number of parameters of the network. For example, in the previous case, if we had one hidden layer with 100 neurons, the number of weights between the input layer and this hidden layer would be over 92 million. Therefore, learning these parameters will be computationally expensive and, for bigger networks, could be unfeasible at all.

Second, when data are inputted in this format, the model is often not invariant to translations, rotations, or other small changes in the input. For example, if we were about to classify an image

of a car, under this approach the model would be trying to learn a function that describes a car in a pixel-by-pixel basis. However, this is a very low-level description, and therefore if the color of the car changes, or if is is slightly translated or rotated, then the model will not be able to properly recognize it because the input is significantly affected.

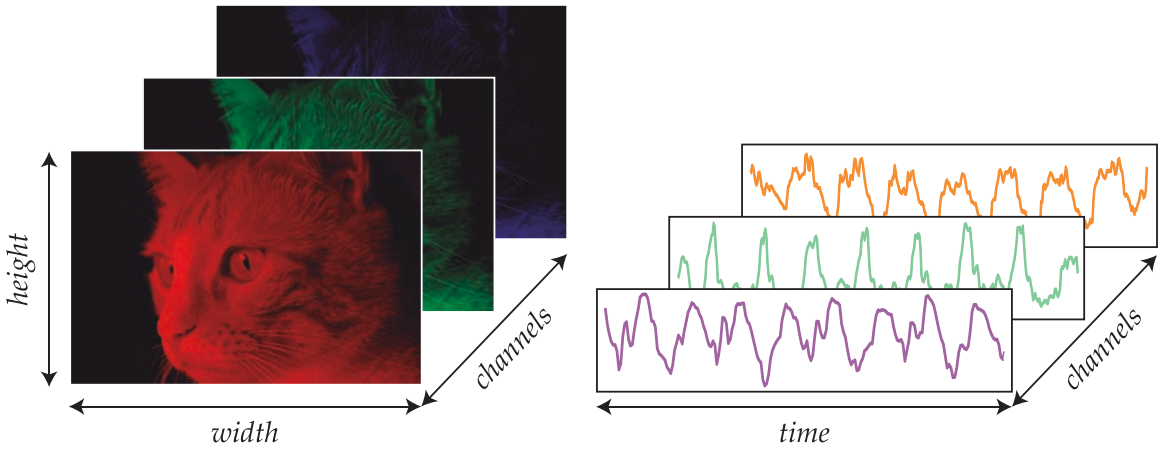
Another approach would be to perform feature engineering of the input data. In this case, an expert or group of experts would decide which features are useful for the model to accurately learn the mapping between the input and the output. Such experts could decide, for example, that relevant features are the number of edges in the image, the number of circles, the percentage of predominantly red bits, etc. Then, those features would be encoded in the input vector.

The main drawback of this approach is that human expertise is required, and this knowledge may be difficult to locate. The relevant features change across different domains, and significant amounts of trial-and-error could be required before eventually finding a good set of features.

To solve this problem, the field of "representation learning" encompasses a set of techniques to automatically discover relevant features from raw data. When talking of deep learning, representation learning is often performed by means of convolutional neural networks. These are deep neural networks that involve some convolutional layers (described in the following section) which are responsible for the representation learning task.

In this kind of neural networks, instances are introduced not as vectors, but as tensors. A tensor is no more than a multidimensional array, that can contain any arbitrary number of dimensions. A vector and a matrix are two particular instances of 1- and 2-dimensional tensors respectively. To illustrate how tensors can be used to represent data, let us describe some scenarios:

- One scenario could involve RGB images. In this case each image could be represented using a 3-dimensional tensor: the first two dimensions would refer to the width and height of the image respectively, whereas the third would store the channels: red, green and blue. An example of such a 3-dimensional tensor can be found in figure 2.5a.
- Another scenario could involve motion data acquired from an accelerometer. In this case, each instance could be represented with a 2-dimensional tensor, where the first dimension refers to the time and the second dimension would store the three channels: x, y and z. An example of this data structure can be found in figure 2.5b.



(a) *RGB image (3D tensor).* **(b)** *Accelerometer data (2D tensor).*

Figure 2.5: *Examples of how different data can be represented using tensors.*

- Our last scenario involves color video, which is a temporal sequence of images. Therefore, we could represent it using a 4-dimensional tensor: the first dimension would be time and the following three would store the information for the images (width, height and color).

Finally, inputting data as tensors into a convolutional neural network will enable to automatically obtain features that are based on the topology of the data. Whereas unrolling the data into a vector was a poor decision because structure was lost, representation learning can now exploit the fact that data are structured to find features that are present in space, time, or other dimensions.

2.5 Convolutional Neural Networks

A convolutional neural network (CNN) is a kind of deep neural network that includes convolutional layers in order to carry out representation learning. CNNs were first introduced by LeCun et al. in 1998 [202, 203]. The idea behind CNN is to combine a feature learning module along with a trainable classifier, which often consists of a fully connected network. The feature learning module would replace a prior feature engineering stage, often done by hand, to reduce data processing to a minimum. In fact, CNNs are intended to work with raw data (or data with very few preprocessing). After features have been learned from raw data, they are introduced to a trainable classifier.

CNNs are interesting for solving many different problems since they provide invariance to translations or local distortions of the input. Also, CNNs rely on the topology and structure of input data for extracting features. For example, images are 2-dimensional, and CNNs will take advantage of this structure to compute local features, providing a major advantage over traditional feed-forward networks with 1-dimensional inputs.

2.5.1 Forward Propagation in Convolutional Layers

The typical anatomy of a CNN is shown in figure 2.6 (elements are not in scale), showing some of the key hyperparameters that can be determined. In this thesis, we have considered the case of a network where layers are stacked sequentially one after the other. We have not considered non-sequential networks, although some works have explored their use, which is frequent in Inception networks [354] or residual networks (ResNets) [145].

CNNs comprise first a sequence of one or more convolutional layers, responsible for learning relevant features from raw data. First, raw data will be directly introduced as an input to the first convolutional layer. This layer will output "*feature maps*" from the input, which will then be introduced as the input for the subsequent layer. This process is repeated until there are no more

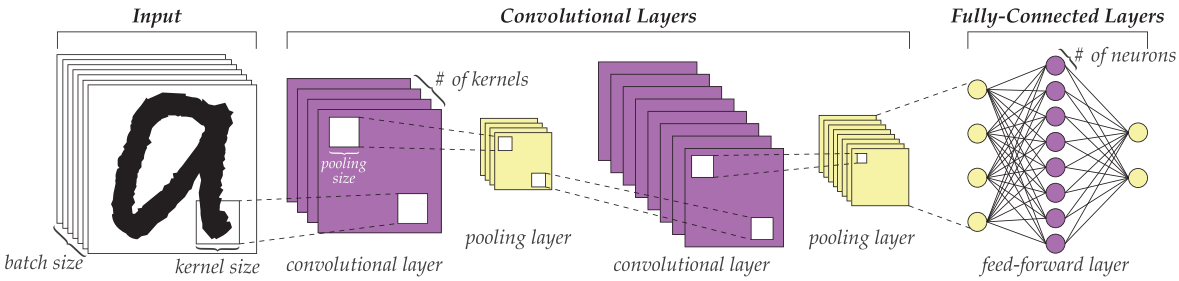


Figure 2.6: *Typical structure of a sequential convolutional neural network.*

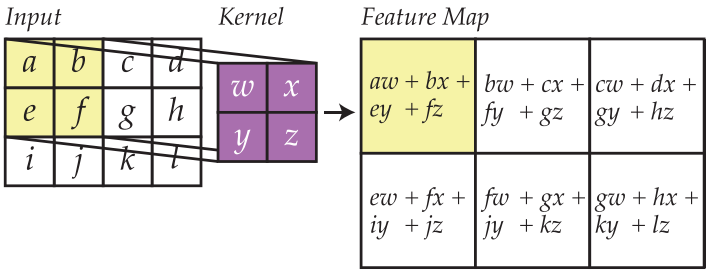
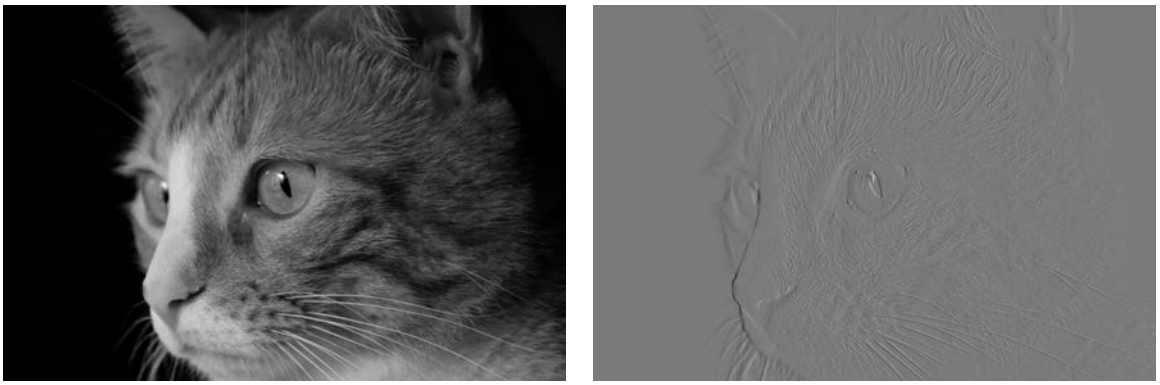


Figure 2.7: *Example of how a kernel is used to convolve the input.*



(a) *Original input.* **(b)** *Input after convolution.*

Figure 2.8: *Features extracted from an image after convolution.*

convolutional layers. Because convolutional layers compute feature maps over their input, the more number of layers the network has, the more abstract (or high level) features it will be able to extract.

Each convolutional layer will comprise several kernels, also known as filters or patches, which *convolve* the input to generate a feature map as an output. Input data and kernels will be structured as tensors. As an example, if we were dealing with images then we would have 2-dimensional inputs and kernels. Whereas the input size is given by the domain or by the output size of the previous layer, kernel size is defined as a hyperparameter of the network topology.

An example of how a kernel convolves the input to generate a feature map is shown in figure 2.7, assuming a unitary stride (the number of positions that the kernel will move when sliding over the input) and no padding[3] (which could be used to pad the feature map with zeros in order to retain the original dimensions of the input). Under these conditions, the feature map for an input of size (i_w, i_h) and a kernel of size (k_w, k_h) will have dimension $(i_w - k_w + 1, i_h - k_h + 1)$. Convolution is computed following equation 2.38, assuming a 2-dimensional input and kernel, where $\mathbf{X}$, $\mathbf{K}$ and $\mathbf{F}$ correspond to the tensors for the input, the kernel and the feature map respectively:

$$\mathbf{F}(i,j) = (\mathbf{K} * \mathbf{X})(i,j) = \sum_m \sum_n \mathbf{X}(i-m, j-n)\mathbf{K}(m,n) \tag{2.38}$$

An example of a convolution over a 2D image is shown in figure 2.8. The left image shows the original input, whereas the right image shows the result of convolving the input with the kernel

[3]This padding configuration where no padding is performed at all is sometimes referred to in some contexts and programming frameworks as "valid padding".

$[1, -1]$, which corresponds to a vertical edges detector. Edges can be clearly seen in the right image. This explanation describes the behavior of a general CNN. However, while many different setups can be defined, these will often have little effect on the network performance.

In many cases, the input will involve several channels (for example, RGB images comprise three channels, one per color), and in those cases each kernel contains as many channels as the number of channels in the input. As a result, the convolution is applied across each channel and then all channels are aggregated. The number of feature maps resulting from the output of a convolutional layer will be equivalent to the number of kernels in that layer. It is worth noting that the concept of "channel" is equivalent to that of a "feature map", with the sole difference that the former is more often used to refer to the input whereas the latter is used for subsequent layers.

Regarding the number of parameters involved in a layer, a kernel of size (k_w, k_h) will comprise $k_w \times k_h \times k_c$ weights, where k_c is the number of channels or feature maps introduced to that layer. Often, an additional bias parameter is included, to be added to the result; however, this bias was not shown in figure 2.7 to keep it simple. Therefore, if a convolutional layer involves n_k kernels of size (k_w, k_h), then it will comprise $n_k \times (k_w \times k_h \times k_c + 1)$ parameters.

After feature maps are computed, they can be transformed by applying a function element-wise. If this function were non-linear, this would compute a non-linear transform of the feature map, potentially enabling the extraction of more complex features. Finally, the output of the layer will be passed as the input for the next layer in the sequence.

Optionally, a pooling operator can be found after a convolutional layer. The purpose of pooling is to reduce the dimensions of the input by performing down-sampling, replacing part of a feature map in a certain location by a statistical summary of the nearby locations.

The most common example is max-pooling, where the input is reduced by taking a subtensor of the feature map and replacing it by its maximum value. In the case of a 2D feature map (as in the image example), the tensor would be a matrix. The pooling operator is defined by its size, for instance, an $i \times j$ pooling operator will reduce the input width by a factor of i and its height by a factor of j. Besides max-pooling, additional functions can be used, like avg-pooling. However, the function used does not have a great impact on the result.

Additionally, pooling has been proved to introduce certain invariance to translation, meaning that if the input is slightly translated, most of the pooling output will remain unchanged.

2.5.2 *Backpropagation in Convolutional Layers*

In order to learn the parameters of the convolutional layers, backpropagation is used, similarly to the process described for the multilayer perceptron. In most cases, the implementation of backpropagation is not explicitly required by the engineer or scientist since most deep learning frameworks compute the derivatives and apply this process automatically. However, in the following lines we provide some intuition of how backpropagation is performed.

To understand how the parameters are updated, we first need to understand that, as shown in figure 2.6, the output of the last convolutional layer is introduced to a fully connected architecture, such as a multilayer perceptron, which will be used to perform classification (we will provide further detail about this process in the following section). When describing the multilayer perceptron, we used the superscript $[l]$ to refer to layer l. Since now we are dealing with two types of layers, we will use the following notation: superscript $[f_l]$ will refer to the l-th layer of the fully connected part of the network and superscript $[c_l]$ the l-th convolutional layer, having a total of f_L fully connected layers and c_L convolutional layers.

Now, we can apply the derivation rules described earlier to compute the gradient of the input of this feed-forward network. If we computed the gradient of the input of the fully connected part, we would denote this following equation 2.39:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{A}^{[\mathbf{f_0}]}} = \frac{\partial \mathcal{J}}{\partial \mathbf{Z}^{[\mathbf{f_0}]}} \tag{2.39}$$

It should be noted that $\mathbf{A}^{[\mathbf{f_0}]}$ and $\mathbf{Z}^{[\mathbf{f_0}]}$ are equivalent by definition, because the input layer of the fully connected network implements an identity function, and therefore the values do not change after computing the activation function.

Additionally, there is some value equivalence (yet not structural equivalence) between $\mathbf{Z}^{[\mathbf{f_0}]}$ and $\mathbf{A}^{[\mathbf{c_L}]}$, because the input of the fully connected network corresponds to the output of the last convolutional layer. The difference between these two values regarding their structure is due to the fact that the input to the fully connected network is a vector (or a matrix if we are computing it across m instances) and the output of the last convolutional layer is a tensor with potentially several feature maps. However, the values of the gradient are preserved, and in backpropagation the "unroll" process carried out during forward propagation can be reversed.

Given the value of the derivative of the cost with respect to $\mathbf{A}^{[\mathbf{c_L}]}$, we can obtain the value of the gradient before the activation function is applied following equation 2.40, with f' being the derivative of the activation function of the convolutional layer:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{Z}^{[\mathbf{c_l}]}} = \frac{\partial \mathcal{J}}{\partial \mathbf{A}^{[\mathbf{c_l}]}} \odot f'\left(\mathbf{Z}^{[\mathbf{c_l}]}\right) \tag{2.40}$$

The weights and biases can be updated following equation 2.34. In this case, the derivative of the cost with respect to the parameters of a certain convolutional kernel $K_c^{[c_l]}$ defined by weights $\mathbf{W_c}^{[\mathbf{c_l}]}$ and a bias $b_c^{[c_l]}$ can be computed as described in equation 2.41:

$$\begin{aligned}
\frac{\partial \mathcal{J}}{\partial \mathbf{W_c}^{[\mathbf{c_l}]}} &= \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} \mathbf{A}^{[\mathbf{l-1}]}(h:h+i, w:w+j, c) \odot \frac{\partial \mathcal{J}}{\partial \mathbf{Z}^{[\mathbf{c_l}]}(h,w,c)} \\
\frac{\partial \mathcal{J}}{\partial \mathbf{b_c}^{[\mathbf{c_l}]}} &= \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} \frac{\partial \mathcal{J}}{\partial \mathbf{Z}^{[\mathbf{c_l}]}(h,w,c)}
\end{aligned} \tag{2.41}$$

In the previous equation, i and j corresponds to the height and width of kernel $K_c^{[l]}$ respectively, and we use the notation $\mathbf{X}(a_1 : a_2, b_1 : b_2, \dots)$ to refer to the subtensor of $\mathbf{X}$ delimited by indices a_1 and a_2 in the first dimension, b_1 and b_2 in the second dimension, etc.

Finally, we need to compute the derivative of the cost with respect to the activation values of the previous layer given the values known for the current layer. In this case, we can face two scenarios: the current layer performs pooling after the convolution or not. In case it performs pooling, we will first need to compute the gradient before the pooling takes place, following equation 2.42:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{A}^{[\mathbf{c_l}]}(h:h+i, w:w+j, c)} = \max\left(\mathbf{A}^{[\mathbf{c_l}]}(h:h+i, w:w+j, c)\right) \odot \frac{\partial \mathcal{J}}{\partial \mathbf{A}_{\mathbf{pool}}^{[\mathbf{c_l}]}(h,w,c)} \tag{2.42}$$

In this case, $\max(\cdot)$ is a function that receives as input a matrix and returns a binary mask with the same dimensions of the input matrix where the maximum value will be set to 1 and all the remaining values will be set to zero. $\mathbf{A}_{\mathbf{pool}}^{[\mathbf{c_l}]}$ is the output of the pooling layer, which corresponds to the input of the following layer. It should be noted that to obtain the gradient, we need to slide through the feature maps, just as we did during forward propagation.

To compute the gradient given a convolutional layer with no stride and no padding, just as the one we described before, we could apply equation 2.43:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{A}^{[\mathbf{c_l}-\mathbf{1}]}(h:h+i, w:w+j)} = \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} \sum_{c=1}^{n_C} \mathbf{W}_{\mathbf{c}}^{[\mathbf{c_l}]} \odot \frac{\partial \mathcal{J}}{\partial \mathbf{Z}^{[\mathbf{c_l}]}(h,w,c)} \tag{2.43}$$

Again, to obtain the gradient we need to iterate through the different feature maps, sliding through them. The process described above can be repeated for every convolutional layer until all parameters in the network have been updated.

2.6 Dense and Recurrent Layers in CNNs

As we have described before, convolutional layers aim at extracting relevant features from raw data. Once features are extracted, these can be introduced to a classifier. In most cases, a fully connected network is used for classification, though some works have explored different approaches, such as GoogLeNet using average pooling [354]. In order to introduce the output tensor of feature maps to the fully connected network, this tensor must be flattened or unrolled; i.e., reshaped into a vector. The fully connected network can comprise several layers of different types; e.g. feed-forward or recurrent. Sometimes, fully connected layers are also called dense layers.

As we have already seen, neurons in each layer will process the input and generate an output. In the case of feed-forward layers, they will receive an input from neurons in the previous layer by means of connections with assigned weights. Given $\mathbf{a}^{[\mathbf{f_l}-\mathbf{1}]}$ as the input vector to layer f_l, $\mathbf{W}^{[\mathbf{l}]}$ as the matrix of weights for that layer, $\mathbf{b}^{[\mathbf{l}]}$ as the vector of biases, and f as an activation function, the output of the layer is computed following equation 2.44:

$$\mathbf{a}^{[\mathbf{f_l}]} = f\left(\mathbf{a}^{[\mathbf{f_l}-\mathbf{1}]}\mathbf{W}^{[\mathbf{l}]} + \mathbf{b}^{[\mathbf{l}]}\right) \tag{2.44}$$

Regarding recurrent layers, they can be of different types. A basic version of a recurrent layer would match a feed-forward layer but with the input not only coming from the previous layer, but also by itself, and a matrix of weights $\mathbf{W}_{\mathbf{r}}^{[\mathbf{l}]}$ must be considered for these new connections. Then, the output of the layer is computed following equation 2.45, where t refers to a certain point in time:

$$\mathbf{a}^{[\mathbf{f_l}]}{}_t = f\left(\mathbf{a}^{[\mathbf{f_l}-\mathbf{1}]}{}_t\mathbf{W}^{[\mathbf{l}]} + \mathbf{a}^{[\mathbf{f_l}]}{}_{t-1}\mathbf{W}_{\mathbf{r}}^{[\mathbf{l}]} + \mathbf{b}^{[\mathbf{l}]}\right) \tag{2.45}$$

Because recurrent layers have connections to themselves, they receive the output from previous time steps, and are able to learn patterns or functions which depend on temporal context. However, if this context goes back long into the past, then they do not seem able to learn them properly.

To solve this issue, LSTMs (standing for long short-term memory) were introduced by Hochreiter and Schmidhuber in 1997 [153]. The architecture of a LSTM cell with peephole connections is

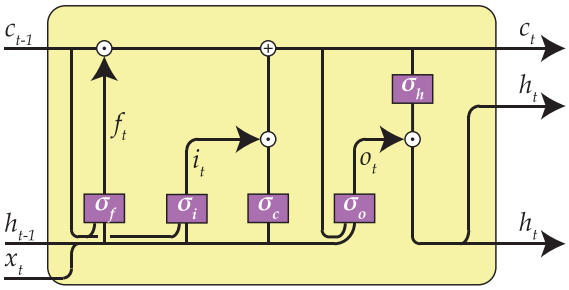


Figure 2.9: *LSTM cell.*

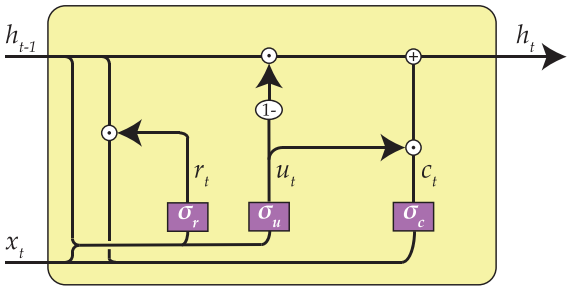


Figure 2.10: *GRU cell.*

shown in Figure 2.9. In LSTM cells, c_t is called the "cell state". A cell contains three gates that control the cell state by adding or removing information. The first gate is the "forget gate" and decides which information will be forgotten from c_{t-1}, the second gate is the "input gate" and decides which new information will be included into c_t, and the last gate is the "output gate" and decides which information will constitute the output. The math behind a LSTM cell is shown in equation 2.46, where $\odot$ stands for an element-wise product operator:

$$\begin{aligned} f_t &= \sigma_f(x_t W_{xf} + h_{t-1} W_{hf} + w_{cf} \odot c_{t-1} + b_f) \\ i_t &= \sigma_i(x_t W_{xi} + h_{t-1} W_{hi} + w_{ci} \odot c_{t-1} + b_i) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \sigma_c(x_t W_{xc} + h_{t-1} W_{hc} + b_c) \\ o_t &= \sigma_o(x_t W_{xo} + h_{t-1} W_{ho} + w_{co} \odot c_t + b_o) \\ h_t &= o_t \odot \sigma_h(c_t) \end{aligned} \tag{2.46}$$

The first of these gates, σ_f is the "forget gate" which will look at the input and decide which information will be forgotten from c_{t-1}. In particular, f_t will be a vector of real values in the interval $[0, 1]$, that will be multiplied element-wise by c_{t-1}. A 0 means that the corresponding information in c_{t-1} will be ignored, whereas a 1 would leave that information unaltered. The second gate, σ_i is called the "input gate", and decides which new information will be included into the cell state. In particular, i_t will be a real vector with the same format than f_t, and a function σ_c will transform the input data. Later, the transformed data and f_t will be multiplied element-wise, and the result will be summed up with c_{t-1}, resulting in c_t. The final gate, σ_o, is called the "output gate" and will decide which information will constitute the output. o_t will be a vector similar to f_t. However, it will not be used to modify the cell state; instead, it will be applied over a transformed version of the cell state to be returned as output.

A variation of LSTM is called GRU (gated recurrent unit), introduced by Cho et al. in 2014 [64]. It combines forget and input gates into an "update gate", and combines the cell state with the hidden state, resulting in a simpler unit, sometimes preferred to LSTM. The architecture of a GRU cell is shown in Figure 2.10, and the math behind this kind of units is shown in equation 2.47:

$$\begin{aligned} r_t &= \sigma_r(x_t W_{xr} + h_{t-1} W_{hr} + b_r) \\ u_t &= \sigma_u(x_t W_{xu} + h_{t-1} W_{hu} + b_u) \\ c_t &= \sigma_c(x_t W_{xc} + r_t \odot (h_{t-1} W_{hc} + b_c) \\ h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t \end{aligned} \tag{2.47}$$

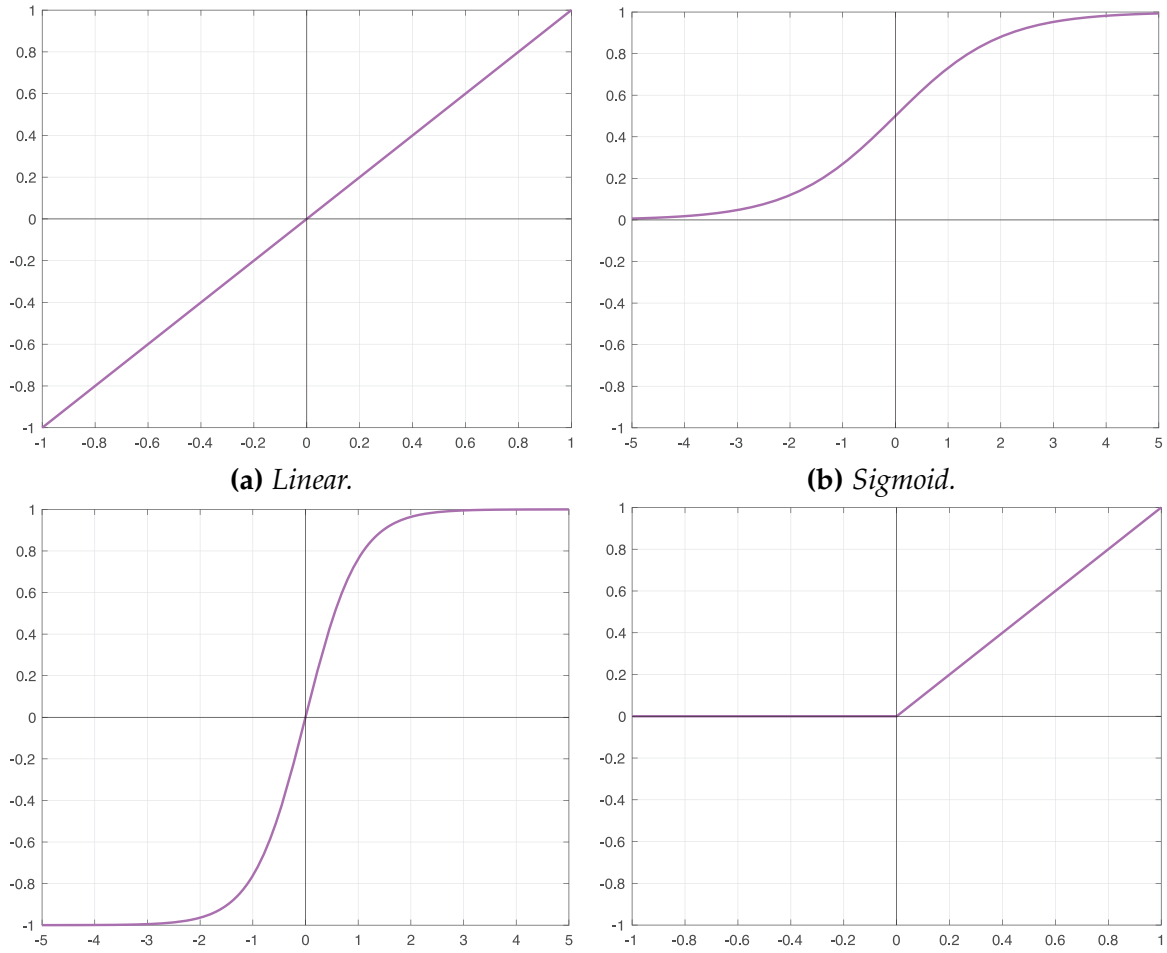
While there are many more implementations for recurrent cells, the selection of one over the other often has few impact on the outcome [128].

2.7 Activation Functions in Deep Neural Networks

We have seen earlier how an activation function is computed both after each convolutional layer and in each of the neurons of dense fully connected layers. Often, a distinction is made between linear and non-linear activation functions. When non-linear activation functions are used, a neural network such as a multilayer perceptron can act as a universal function approximator [78].

The most basic linear function is the identity function. The plot for the identity function is depicted in figure 2.11a and its formula is shown in equation 2.48:

$$f(z) = z \tag{2.48}$$



(a) *Linear.* **(b)** *Sigmoid.*

(c) *Hyperbolic tangent.* **(d)** *Rectified linear unit (ReLU).*

Figure 2.11: *Typical activation functions in neural networks.*

The identity function is easy to compute and can serve to approximate linear functions (we saw an example of a basic neural model implementing a linear activation function when we described the Adaline in section 2.2.3. However, one important limitation of the linear function, as we described earlier, is that an arbitrary number of stacked layers with linear activation functions can be replaced with a single layer. Therefore, complex functions cannot be approximated. The derivative of the identity function is trivial, as shown in equation 2.49:

$$f'(z) = 1 \tag{2.49}$$

The early models of neural networks with non-linear activation functions often implemented the sigmoid function (see figure 2.11b), also known as logistic function, which is the activation present in logistic regression. The formula for this function is shown in equation 2.50:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.50}$$

The range of the sigmoid function is the interval $(0, 1)$, and therefore the output can be easily interpreted as the probability that an instance belongs to a certain class. In 1989, only three years after backpropagation was introduced, Cybenko et al. [78] proved that a neural network implementing a sigmoid activation function could work as a universal function approximator. The derivative of the sigmoid function is shown in equation 2.51:

$$f'(z) = f(z)(1 - f(z)) \tag{2.51}$$

One problem of the sigmoid function is that it is specially prone to the vanishing gradient problem, described earlier in the chapter, which is specially harmful when training a deep neural network. One reason for this is that a strongly negative input outputs a value very close to zero, and since backpropagation uses the value of forward-propagation to compute the gradient, the parameters could be prevented from update in these scenarios. Additionally, the derivative of this function is never larger than 0.25, and when the gradient gets multiplied across the different layers during backpropagation, this gradient gets smaller and smaller in the early layers.

To partially solve this issue, the hyperbolic tangent activation function was introduced (see figure 2.11c). This function is shaped like the sigmoid function, but is defined in the range $(-1, 1)$. Its formula is shown in equation 2.52:

$$f(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.52}$$

The derivative of the hyperbolic tangent function is shown in equation 2.53, and it can be seen that the value $f'(z)$ can be easily calculated given the value of $f(z)$:

$$f'(z) = 1 - f^2(z) \tag{2.53}$$

Since the rise of deep and convolutional neural networks, sigmoid and hyperbolic tangent functions have been replaced in many works in favor of the rectified linear unit, known as ReLU. This non-linear function is shown in figure 2.11d and its formula is shown in equation 2.54:

$$f(z) = \max(0, z) \tag{2.54}$$

ReLU activations have been proven to leverage faster training than other non-linear activation functions. In the words of Krizhevsky et al. [190], "*networks with ReLUs consistently learn several times faster than equivalents with saturating neurons*".

Finally, the derivative of the ReLU is shown in equation 2.55. Technically, the derivative when $x = 0$ is undefined, but by convention the value 0 is considered for the derivative at that point:

$$f'(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.55}$$

2.7.1 A Note on C Classes: Softmax Regression and One-Hot Encoding

So far, we have described the case of supervised learning in a binary classification problem. Let us denote these classes as 0 and 1 (or negative and positive respectively). In that case, the last layer of the neural network will often comprise a single unit with a sigmoid activation function. Therefore, $\hat{y} = a_1^{[L]} = \sigma\left(z_1^{[L]}\right)$ will be a value between 0 and 1, that can be interpreted as the probability of the instance belonging to class 1. For example, if $\hat{y} = 0.3$, then we can interpret that output as the instance having a 30 % of probability of being positive and, conversely, a 70 % of being negative. Since we require the output to be a class, we will often choose the most likely one.

If we wanted to perform multi-class classification, with C different classes (which we will denote from c_1 to c_C), we could use a softmax activation function in the output layer. To do so, we would need to have C output neurons, one per each class. Therefore, before computing the activation function, the output vector $\mathbf{z}^{[\mathbf{L}]}$ will have C values, as shown in equation 2.56:

$$\mathbf{z}^{[\mathbf{L}]} = \begin{bmatrix} z_1^{[L]} \\ z_2^{[L]} \\ \vdots \\ z_C^{[L]} \end{bmatrix} = \begin{bmatrix} \mathbf{w_1}^{[\mathbf{L}]\intercal} \mathbf{a}^{[\mathbf{L-1}]} + b_1^{[L]} \\ \mathbf{w_2}^{[\mathbf{L}]\intercal} \mathbf{a}^{[\mathbf{L-1}]} + b_2^{[L]} \\ \vdots \\ \mathbf{w_C}^{[\mathbf{L}]\intercal} \mathbf{a}^{[\mathbf{L-1}]} + b_C^{[L]} \end{bmatrix} \tag{2.56}$$

Then, softmax activation will compute the output vector $\hat{\mathbf{y}}$ following equation 2.57:

$$\hat{\mathbf{y}} = \mathbf{a}^{[\mathbf{L}]} = \frac{\exp\left(\mathbf{z}^{[\mathbf{L}]}\right)}{\sum_{j=1}^{C} \exp\left(\mathbf{z_j}^{[\mathbf{L}]}\right)} \tag{2.57}$$

As it can be seen, the output vector is normalized, therefore all values will sum up to 1. As a result, the output $\hat{\mathbf{y}}$ can also be interpreted as a vector of probabilities, with value $a_c^{[L]}$ representing the probability of the instance belonging to class c.

In order to start the training of a neural network using a softmax activation function in the last layer, we need to do the following. First, the class of the instance must be represented with a vector $\mathbf{y}$ of length C, following an approach known as "one-hot encoding": if an instance has class c, then the target vector $\mathbf{y}$ will be full of zeros except for position c, which will have a value of 1.

The general loss function given a softmax output with C classes is described in equation 2.58, which is known as "cross entropy loss":

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{j=1}^{C} y_j \log \hat{y}_j \tag{2.58}$$

However, it should be noted that y_j is 0 for every $j \neq c$, with c being the class of the instance. Also, we know that y_c is 1, and thus the previous loss function can be simplified to the expression shown in equation 2.59:

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = -\log \hat{y}_c \tag{2.59}$$

Finally, in order to start backpropagation given a softmax activation in the output layer, we need to compute the gradient of the values in the last layer, for which we can apply equation 2.60:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\mathbf{L}]}} = \hat{\mathbf{y}} - \mathbf{y} \tag{2.60}$$

2.8 Regularization of Network Parameters

A problem that can arise when training artificial neural networks is overfitting. By overfitting we understand learning a model that can accurately approximate the training set, but that generalizes poorly; i.e., it would not perform that well given a test set. In other words, if we validate a neural network model against a test set which follows an identical distribution that the training set, and the performance over that test set is worse than the performance over the training set, then we can state that the model suffers from overfitting.

Two causes that can explain overfitting are very small training sets and very complex architectures. Traditionally, it was found that neural networks with many parameters could easily result in overfitting. However, in the era of deep learning, it has been found that complex models with several convolutional and fully connected layers, which can have millions or tens of millions of parameters, often perform better as long as there are enough training data available.

Nevertheless, this is an important lesson: deep neural networks often require large datasets in order to attain models that can generalize properly. In some fields, such as computer vision or natural language processing, data augmentation techniques are often used to artificially increase the training dataset with synthetic data resulting from distortions of the original data, for instance, including rotations or translations of images or adding background noise to a piece of audio.

However, obtaining more training data can be a difficult task additional, and other measures can be applied to avoid overfitting. One of the most widely used is regularization.

A common way to implement this technique is by means of L2 or L1 regularization. In L2 regularization, the L2 norm (or Frobenius norm[4]) of the weights matrices are added to the cost function as shown in equation 2.61:

$$\mathcal{J}\left(\mathbf{W}^{[\mathbf{1}]}, \mathbf{b}^{[\mathbf{1}]}, \ldots, \mathbf{W}^{[\mathbf{L}]}, \mathbf{b}^{[\mathbf{L}]}\right) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{\mathbf{y}}^{(\mathbf{i})}, \mathbf{y}^{(\mathbf{i})}\right) + \frac{\lambda}{2m}\sum_{l=1}^{L} ||\mathbf{W}^{[\mathbf{l}]}||_2^2 \tag{2.61}$$

[4]The term "Frobenius norm" is preferred for matrices.

In the previous equation, the last term (also called "penalty") is the one providing regularization to the model, where m is the number of instances, L is the number of layers and λ is a hyperparameters known as the "regularization hyperparameters", which determines how much the model is affected by regularization. This technique is known as L2 regularization because $||\mathbf{w}||_2^2$ is the square of the L2 norm of the weight matrix, whose computation is described by equation 2.62:

$$||\mathbf{W^{[l]}}||_2^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} \left(W_{ij}^{[l]}\right)^2 \tag{2.62}$$

Despite in equation 2.61 we added the regularization term for all the weights of all layers in the network, in practice only some layers could be considered for regularization.

Finally, a change in the computation in the gradient is required to consider the regularization term during backpropagation. In particular, the derivative of the cost with respect of the weights in a certain layer must be adapted as shown in equation 2.63:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{W^{[l]}}} \leftarrow \frac{\partial \mathcal{J}}{\partial \mathbf{W^{[l]}}} + \frac{\lambda}{m}\mathbf{W^{[l]}} \tag{2.63}$$

Now, how does regularization works? An intuition can be easily found if we consider a extreme case where the hyperparameter λ is very big. In that case, gradient descent will try to make weights very small, since large weights will lead to a higher cost value. It is worth noting that if some weights were zero, then we could consider that associated neurons were disabled, thus resulting in a simpler network. Additionally, this "weight decay" has an interesting property when using sigmoid or hyperbolic tangent activation functions: these functions behave as linear when values are close to 0; therefore some units could take a linear behavior, again simplifying the network.

Of course, the previous consequences are specially noticeable in the case where regularization is applied too strongly by setting a very large regularization hyperparameters. The purpose, however, is to set a value that can find a tradeoff between a simpler model (less prone to overfitting) and a model complex enough to accurately fit the data.

While L2 regularization is very common, L1 regularization can also be applied. In this case, the cost function is modified as shown in equation 2.64:

$$\mathcal{J}\left(\mathbf{W^{[1]}}, \mathbf{b^{[1]}}, \ldots, \mathbf{W^{[L]}}, \mathbf{b^{[L]}}\right) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\mathbf{\hat{y}^{(i)}}, \mathbf{y^{(i)}}\right) + \frac{\lambda}{m}\sum_{l=1}^{L} ||\mathbf{W^{[l]}}||_1 \tag{2.64}$$

The L1 norm is computed as described in equation 2.65:

$$||\mathbf{W^{[l]}}||_1 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} |W_{ij}^{[l]}| \tag{2.65}$$

In L1 regularization, the gradient of the cost with respect to the weights will need to be modified according to equation 2.66:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{W^{[l]}}} \leftarrow \frac{\partial \mathcal{J}}{\partial \mathbf{W^{[l]}}} + \frac{\lambda}{m}\frac{\mathbf{W^{[l]}}}{|\mathbf{W^{[l]}}|} \tag{2.66}$$

The idea behind L1 regularization is similar to L2 regularization, although it is more likely to set some weights to 0, thus resulting in more sparse models. However, L1 regularization is used less frequently than L2. Also, both regularization terms could be used simultaneously.

Despite L2 regularization being widely used, in recent years a novel and extremely simple regularization technique has received significant attention and become the *de facto* standard in deep learning: dropout. Dropout was introduced by Srivastava et al. [340] and basically consists on disabling a certain number of units in each layer during training. The percentage of disabled neurons is a hyperparameter, although 50 % has been shown to be a good decision [12] and is the default value chosen by Srivastava et al.

For dropout to work, two considerations must be taken into account: the disabled neurons must change between different epochs and the activation of the layer must be adjusted to compensate the effect of the disabled units. This adjustment can be performed by dividing the activation function by $1 - p$, with p being the dropout probability. Because $p < 1$, then this division will increase the activation values and by doing so, they will not be reduced by effect of dropout.

One intuition why dropout works well is because it forces the network to be robust in order to properly classify instances even when some part of it (often half of it) is randomly disabled. In practice, a way to do so is to reduce the weights, since it makes no sense to have a large weight that gives a big importance to a feature that can disappear at any time in the future. This mechanism of reducing weights hold some similarities with the L1 and L2 regularizations we described earlier. It is worth noting that dropout is only used during training, because it adds randomness to the process which must be avoided at test time, and it can be used with different probability (or not used at all) in each layer.

2.9 Optimization Algorithms

Once the topology is determined, the network parameters or weights must be learned. This process is called "training" of the neural network. Previously, we have described the backpropagation process that adjust the weights and biases of the neural networks (both the convolutional and the fully connected layers), with the purpose of minimizing a cost function which measures how wrongly the input instances are classified.

To train the network, data from a training set will be introduced. In convolutional neural networks, raw data will be introduced as input to the first convolutional layer. So far we have described the case where the whole training set is introduced to the network, getting the benefit of vectorization, which enables faster computation of the activations and the gradient.

The problem of this approach is that the entire training set must be processed before the weights are updated in each step. This can result in a slow progress if the training set is very large, which is often the case in deep learning. For this reason, it is common to introduce a smaller set of samples, called a "mini-batch". We call this approach "mini-batch gradient descent", as opposed of "full batch gradient descent" which is the name often given to the earlier approach where the whole dataset is introduced. When using mini-batch training descent, we often call the process of completing a whole iteration over the training set an "epoch".

Mini-batch gradient descent has been shown to provide faster convergence with large training sets, when compared to full-batch gradient descent. Additionally, using mini-batches might be the only option when the full training set does not fit in CPU or GPU memory.

The specific implementation used to learn the parameters of the neural network is called "learning rule" or "optimization algorithm" (although in some contexts it is also known as "optimizer").

This implementation uses the value of a cost function computed over the output of the network and the expected output to modify the weights in a certain direction (gradient) in order to reduce the value of the loss function.

So far, we have discussed gradient descent, which has been for decades the most common way of training neural networks. When mini-batches are used, we will talk of "stochastic gradient descent" (SGD), because the parameters will be updated with certain stochasticity, depending on the mini-batch. The expression for updating parameters was shown in equation 2.34.

However, in the latest years new optimization algorithms have appeared that are often faster than gradient descent. One modification to the original gradient descent algorithm introduces the concept of "momentum" to the parameter update stage, following equation 2.67:

$$\begin{aligned} v_{\partial \mathbf{w}} &\leftarrow \gamma v_{\partial \mathbf{w}} - \eta \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \\ \mathbf{w} &\leftarrow \mathbf{w} + v_{\partial \mathbf{w}} \end{aligned} \tag{2.67}$$

In the previous equation, γ is the momentum, and $v_{\partial \mathbf{w}}$ is the velocity at a certain iteration of gradient, which is initialized to zero upon start. This velocity is computed following an exponentially weighted moving average of the gradient, and serves for the purpose of accelerating or smoothing the updates depending on whether the position in each dimension is far or close to the optimum. The higher the momentum, the more update steps considered in the exponentially weighted moving average, although a typical value in most applications is $\gamma = 0.9$. For simplicity, we have only mentioned a generic parameter $\mathbf{w}$, though the previous equation is applicable to any weight or bias in the neural network.

A different implementation of gradient descent also introducing a momentum is called "Nesterov momentum". In this case, the updates are performed following equation 2.68:

$$\begin{aligned} v_{\partial \mathbf{w}} &\leftarrow \gamma v_{\partial \mathbf{w}} - \eta \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \\ \mathbf{w} &\leftarrow \mathbf{w} + \gamma v_{\partial \mathbf{w}} - \eta \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \end{aligned} \tag{2.68}$$

Another optimization algorithm that can be used to determine how much each weight must be updated in each direction is RMSprop [362]. In this algorithm, the learning rate is scaled by dividing it by the moving average of the root mean squared gradients, as shown in equation 2.69:

$$\begin{aligned} s_{\partial \mathbf{w}} &\leftarrow \rho s_{\partial \mathbf{w}} + (1-\rho) \left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}} \right)^2 \\ \mathbf{w} &\leftarrow \mathbf{w} - \frac{\eta}{\sqrt{s_{\partial \mathbf{w}} + \epsilon}} \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \end{aligned} \tag{2.69}$$

In the previous equation, ρ is the gradient moving average decay factor, and ϵ is a very small value used only to prevent numerical instability. The intuition behind RMSprop is that when the root mean square of the gradients in one dimension is large, then we must reduce the learning rate by a large factor to avoid divergence, whereas when the root mean square of the gradients is small, then the learning rate will not be reduced in order to accelerate convergence.

Two more optimization algorithms share a similar principle than RMSprop. One of them is AdaGrad [92], where the learning rate is scaled by dividing it by the square root of accumulated squared gradients, resulting in the update process shown in equation 2.70:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} \left(\frac{\partial \mathcal{J}_\tau}{\partial \mathbf{w}_\tau}\right)^2 + \epsilon}} \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \tag{2.70}$$

In the previous equation, the learning rate is divided by the element-wise sum of the gradient vectors in all previous times. All other operations are also performed element-wise. The other optimization algorithm sharing intuition with RMSprop is AdaDelta [407], where the learning rate is scaled by the ratio of accumulated gradients to accumulated updates, as shown in equation 2.71:

$$\begin{aligned} r_{\partial \mathbf{w}} &\leftarrow \rho s_{\partial \mathbf{w}} + (1-\rho)\left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}}\right)^2 \\ \eta &\leftarrow \eta \frac{\sqrt{s_{\partial \mathbf{w}} + \epsilon}}{\sqrt{r_{\partial \mathbf{w}} + \epsilon}} \\ s_{\partial \mathbf{w}} &\leftarrow \rho s_{\partial \mathbf{w}} + (1-\rho)\left(\eta \frac{\partial \mathcal{J}}{\partial \mathbf{w}}\right)^2 \\ \mathbf{w} &\leftarrow \mathbf{w} - \eta \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \end{aligned} \tag{2.71}$$

In the previous equation, the operations are performed in the displayed order. $r_{\partial \mathbf{w}}$ and $s_{\partial \mathbf{w}}$ are initialized to zero, and in AdaDelta the non-scaled learning rate is often 1. The hyperparameter ρ must be a number in the range $[0, 1]$, where larger values will decay the moving average slowly and smaller values will decay it faster, yet a value of $\rho = 0.95$ is suggested in the paper.

Finally, in 2014 the Adam optimization algorithm (standing for Adaptive moment estimation) was presented [183], which combines the ideas of gradient descent with momentum and RMSprop. The formulation behind Adam is shown in equation 2.72:

$$\begin{aligned} v_{\partial \mathbf{w}} &\leftarrow \beta_1 v_{\partial \mathbf{w}} + (1-\beta_1)\frac{\partial \mathcal{J}}{\partial \mathbf{w}} \\ s_{\partial \mathbf{w}} &\leftarrow \beta_2 s_{\partial \mathbf{w}} + (1-\beta_2)\left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}}\right)^2 \\ \hat{v}_{\partial \mathbf{w}} &\leftarrow \frac{v_{\partial \mathbf{w}}}{1-\beta_1^t} \\ \hat{s}_{\partial \mathbf{w}} &\leftarrow \frac{s_{\partial \mathbf{w}}}{1-\beta_2^t} \\ \mathbf{w} &\leftarrow \mathbf{w} - \eta \frac{\hat{v}_{\partial \mathbf{w}}}{\sqrt{\hat{s}_{\partial \mathbf{w}}} + \epsilon} \end{aligned} \tag{2.72}$$

The only particularity of the previous equation is that bias correction is applied to $v_{\partial \mathbf{w}}$ and $s_{\partial \mathbf{w}}$, resulting in $\hat{v}_{\partial \mathbf{w}}$ and $\hat{s}_{\partial \mathbf{w}}$. The exponent t in the denominator refers to the number of the current iteration, and since β_1 and β_2 are numbers smaller than 1, we can see that as t increases this correction will tend to have no effect (because of the denominator being 1). What bias correction

does is to increase the value of the exponentially weighted average during the first iterations, when there are still few values for calculating this average. In the original Adam paper [183], authors suggested default hyperparameters of $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

In the original Adam paper, the authors also discuss AdaMax, a variation of Adam based on the infinity norm. The formulation of AdaMax is shown in equation 2.73:

$$\begin{aligned} v_{\partial \mathbf{w}} &\leftarrow \beta_1 v_{\partial \mathbf{w}} + (1 - \beta_1) \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \\ s_{\partial \mathbf{w}} &\leftarrow \max \left(\beta_2 s_{\partial \mathbf{w}}, \left| \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \right| \right) \\ \mathbf{w} &\leftarrow \mathbf{w} - \frac{\eta}{1 - \beta_1^t} \frac{v_{\partial \mathbf{w}}}{s_{\partial \mathbf{w}}} \end{aligned} \tag{2.73}$$

The choice of a suitable learning rule and learning rate is important in order to achieve convergence when optimizing the network weights.

2.10 Deep Learning Frameworks

When implementing deep learning solutions, developers often do not have to manually implement forward propagation or backward propagation. In recent years, many specific programming frameworks and libraries have arisen that perform numerical computation applicable to deep learning. At the very least, these programming frameworks compute the convolution, carry out forward propagation, and perform automatic differentiation compute the gradient to perform backpropagation. In some cases, they also enable the design of neural networks by easily stacking different layers.

Some of these frameworks are being developed by or have significant support of key players in the industry, such as Google, Facebook, Amazon, Microsoft or NVIDIA. In this section, we enumerate the most relevant frameworks for deep learning as of 2018.

Theano Theano is one of the most well-established frameworks for numerical computation with tensors and deep learning, existing since 2010 [28] and distributed under BSD license. It is developed for Python, but since it relies in NumPy, the computation graph is transcompiled to C, therefore enabling faster computing. It can run either on CPU or GPU using the NVIDIA CUDA libraries. Unfortunately, in September 2017, Y. Bengio (one of the main developers of the project and person of reference in the deep learning field) stated their intention to stop contributing to the project after version 1.0 was released [193], since other alternatives developed by the industry were gaining significant attention. Theano 1.0 was finally released on November 15th, 2017.

TensorFlow TensorFlow is a deep learning framework released in late 2015 [1]. Despite it being very recent, it was developed by the Google Brain team and rapidly received significant attention, and as of today, this framework is used by key players such as Google, DeepMind, NVIDIA, Intel, Twitter, etc. The framework is released under Apache 2.0 license and is available both for Python and C++, but even when the development takes place in Python, the computation graph is transcompiled to C in order to run faster. Computations can run both in CPU or GPU, and in the latter case NVIDIA CUDA libraries are detected and used by default.

Torch Torch is, as Theano, a very well-established framework for scientific computing with tensors. The first version was released in 2002 [74,75], and it remains an active project as of 2018 under an open-source BSD license. It is developed for the Lua programming language, and has a package dedicated to neural networks, including convolutional and recurrent networks. Torch has support for GPU computing using NVIDIA CUDA libraries. Torch is currently used by the Facebook AI Research Group [63]. A port of Torch to Python, named PyTorch [290], has just abandoned the beta stage and turned into production in May 2018. Despite its early development status, PyTorch already implements most of features of Torch, including GPU support using CUDA.

Caffe Caffe (standing for Convolutional architecture for fast feature embedding) is a deep learning framework originally developed at University of California Berkeley and released in 2014 [167]. It provides an interface for C++, Python and Matlab, and includes most of the features of top-tier deep learning frameworks, such as convolutional and recurrent networks and GPU support using CUDA libraries. A new project, Caffe2, has been developed by Facebook and released under Apache 2.0 license in 2017. According to the developers, Caffe2 is based on Caffe and is built with modularity in mind, in order to provide more flexible ways to organize computation.

MXNet MXNet is a deep learning framework introduced in 2015 [61], yet it gained significant attention in 2016 when it was chosen by Amazon as its favorite [79], releasing tools to easily deploy it in AWS. It has recently been incorporated to Apache Incubator and is published under Apache 2.0 license. It is developed in C++ but provides interfaces compatible with many different programming languages, including Python, Java, Javascript, Julia, Matlab, R, Scala, Perl or Go. It allows the development of convolutional as well as recurrent networks, and is also compatible with GPU support using CUDA libraries.

Deeplearning4j Deeplearning4j is a deep learning framework launched in 2014 [270]. It is developed written for the Java Virtual Machine, thus supporting programming languages such as Java, Clojure or Scala. The software is open-source and released under Apache 2.0 license, and was contributed to the Eclipse Foundation in October 2017. Deeplearning4j enables the implementation of convolutional and recurrent, as well as being compatible with CUDA libraries. Moreover, training in Deeplearning4j has been designed to be performed in a distributed manner, using Apache Hadoop or Apache Spark, and distributed GPU computing is supported by this framework.

Microsoft Cognitive Toolkit Formerly known as CNTK, this deep learning framework was released in early 2016 [219] and later that year the project was redesigned and renamed as Microsoft Cognitive Toolkit [218]. The framework provides libraries for C++ and Python programming languages, and includes a description language for deep neural networks, namely Brainscript. This framework is published under MIT license, supports convolutional and recurrent networks and can train and run models in NVIDIA GPUs.

Keras Unlike the previous projects, Keras [179] is not really a deep learning framework but rather a library providing an abstraction to easily prototype deep neural networks, simplifying their design and training process. Keras was initially launched on early 2015 and was compatible with Theano. More recently, it has incorporated additional backends for TensorFlow, Deeplearning4j, MXNet and Microsoft Cognitive Toolkit. The fact that Keras provide a single interface to operate all these frameworks enables easily porting models between them. Also, Keras interface allows to design convolutional and recurrent networks, and to train them using GPUs.

Lasagne Similar to Keras, Lasagne [196] is not a complete deep learning framework, but rather a library providing a simplified interface to design and train neural networks. Unlike Keras, Lasagne is only compatible with Theano. The project name comes from the fact that neural networks can be built by stacking layers of different types (convolutional, feed-forward, recurrent, etc). Lasagne provides GPU support using CUDA libraries.

Gluon Gluon [112] is a library that, similarly to Keras and Lasagne, provides a simple interface for prototyping deep neural networks. Gluon is only available for MXNet, and just like the two previous alternatives, enables easily building of convolutional and recurrent networks.

Finally, some of these frameworks are delivered with so-called "model zoos", which are repositories of deep learning models which are ready to be loaded and used. These models are already pre-trained (sometimes following the steps described in research papers) and sometimes transfer learning can be used to apply these models to domains different from those used for training them.

2.11 Evolutionary Computation

For years, an important area of application of computer science has been optimization: the selection of an optimal or set of optimal elements from a larger set of candidates according to some specific criteria. Optimization constitutes one of the strongest pillars within the field of operations research, and this remarkable interest in optimization can be explained because of the numerous applications in the industry, including processes aimed at making decisions for minimizing manufacturing costs or maximizing production rate for a given cost. The problem of training an artificial neural network using backpropagation is also an optimization problem, since the process is aiming at finding the optimal set of weights for minimizing some error or maximizing some quality metric.

Optimization problems are indeed pervasive, and can be found in nature. An example of this is the principle of least effort, first documented in the 19th century [101], that has been acknowledged as the tendency of living beings to naturally choose the path of least resistance [411]. In extension, many problems can be modelled as an optimization problem, including some from economics, politics, engineering, game theory and evolutionary biology. In fact, many of these problems can be restated to become very similar problems (if not equivalent) among them[5].

In the past, some authors have defined evolutionary biology as an optimization process, with John Maynard Smith being one of the greatest exponents of this school of thought [232,278]. While this theory has also been criticized, it still seems that some relationship between optimization processes and evolution can be established. And it is in this intersection that computer science has led to the rise of the field of evolutionary computation, that approaches the solution of computational optimization problems using mechanisms dictated by Darwinian theory of evolution and natural selection, mainly described in his well-known work "On the Origin of Species" [80].

Evolutionary computation is a family of biologically inspired optimization algorithms (called evolutionary algorithms), which lie within the following taxonomy [30,356]:

[5]I personally find the book "Networks, Crowds and Markets" by Easley and Kleinberg [93] a good example of this. In the end, many different problems can be stated in terms of networks, or graphs. Graph theory has been extensively studied and its definitions and properties are widely settled in the literature. Also, several graph algorithms belong to the NP-complete class of computational complexity, hence every other NP-complete problem can, by definition, be quickly reduced to any of these graph-based problems.

- **Metaheuristics**: Evolutionary algorithms often perform optimization within a search space that is too large as to be computationally feasible to be traversed. These techniques do not implement knowledge specific to the problem (as in heuristic search), but instead rely on a measure of quality of each potential solution (called fitness), which may encode partial or even very little knowledge about the problem, and which serves for the purpose of guiding the search process. The algorithms and procedures used for searching are general, meaning that can be reused in different problems with few to no changes, although in some cases some problem-specific knowledge can be introduced to enhance the search process. On the other hand, the main disadvantage of evolutionary algorithms is that they are stochastic in nature and are not guaranteed to find an optimal solution.
- **Population-based**: In every state of the search process, a set of candidate solutions (known as "a population") is maintained. The evolutionary algorithm bases the search procedure in operating with the population by applying operators that involve several individuals at the same time. However, the quality metric (fitness) is intrinsic to each individual.
- **Biologically-inspired**: Evolutionary algorithms are inpired by natural principles, in particular by those established in the Darwinian theory of evolution. For such reason, these algorithms often implement operators such as selection, recombination or mutation, that are applied over one or more individuals of the population.

Algorithms applying principles from natural evolution for computational optimization arose in the 1960s, although the idea of evolving programs dates back to at least 1950, when Alan M. Turing mentioned this idea in his work "Computing Machinery and Intelligence" [367], a paper mostly known for introducing the Imitation Game. In this paper, Turing states:

> *"[...] We have thus divided our problem into two parts. The child programme and the education process. These two remain very closely connected. We cannot expect to find a good child machine at the first attempt. One must experiment with teaching one such machine and see how well it learns. One can then try another and see if it is better or worse. There is an obvious connection between this process and evolution, by the identifications*
>
> *Structure of the child machine = hereditary material*
>
> *Changes of the child machine = mutation*
>
> *Natural selection = judgment of the experimenter*
>
> *One may hope, however, that this process will be more expeditious than evolution. The survival of the fittest is a slow method for measuring advantages. The experimenter, by the exercise of intelligence, should he able to speed it up. Equally important is the fact that he is not restricted to random mutations. If he can trace a cause for some weakness he can probably think of the kind of mutation which will improve it."*

As for the actual implementation of these ideas, different paradigms materialized in the early days of evolutionary computation, in somehow overlapping timeframes:

- **Evolutionary programming**: This paradigm was presented by Fogel et al. [104] in 1966, and its working mechanism consists on optimizing the numerical parameters involved in a computer program whose structure remains fixed. This approach mostly relies on mutation for modifying individuals across generations.
- **Genetic algorithms**: This paradigm was introduced by Holland [154] in 1975, and proposes expressing a candidate solution in the form of a genetic encoding, which is often a binary string even though alternative representations have been suggested that do not conform to the building blocks established by Holland. The evolution then takes place by iteratively applying genetic operators, the most common being selection, recombination and mutation.

- **Evolution strategies**: This paradigm was introduced by Rechenberg [300] and Schwefel [324] in the 1970s. In this approach, a vector of real-valued numbers is evolved through selection and mutation, whose strength is often regulated by means of self-adaptation. In a naive version of evolution strategies, only one parent and one child exist at the same time, and the best from both is kept for the next generation: this particular implementation is known as $(1+1)$–ES. Additional versions have been described with several children, leading to $(1+\lambda)$–ES or $(1,\lambda)$–ES strategies, where in the former children can compete with the parent and in the latter the parent is always disregarded. Also, several parents can be considered, and sometimes recombination is also included as an operator, leading to $(\mu/\rho+,\lambda)$–ES.
- **Genetic programming**: This paradigm is probably the most consistent with the idea the Alan Turing had presented in the previously quoted text. Early implementations of this approach first appeared about 30 years later, with those from Forsyth [105] and Cramer [77], although John Koza is acknowledged to be one of the key researchers that helped to establish the field [187, 188]. In genetic programming, programs have been traditionally represented as trees, which can be modified using genetic operators such as recombination or mutation.

In this thesis we will use two different evolutionary algorithms: genetic algorithms and grammatical evolution. Some fundamental concepts about them will follow these lines, although for further information about basics of evolutionary computation in the context of this work and the specifics of how these techniques are applied the reader is redirected to sections 4.5.

2.11.1 Genetic Algorithms

As stated earlier, genetic algorithms (GAs) are one of the main four paradigms of evolutionary computation, introduced by John Holland in the mid-70s [154].

In a genetic algorithm, candidate solutions are encoded in the form of a genotype, which is a string of genes, otherwise known as chromosome. This terminology is strongly influenced by underlying biological concepts, although in practice a gene is often represented as a bit (or in some cases, a number, a character or other symbol).

Genetic operators are applied over individuals of a population in every generation. Most common operators are selection for determining the fittest individuals, recombination or crossover for producing offspring given a couple of individuals and mutation for randomly modifying a small aspect of the chromosome.

As it happens in nature, in GAs a genotype has an associated phenotype, which is a representation of a candidate solution whose fitness (degree of *goodness*) can be directly evaluated[6]. The genotype-phenotype relationship might not be 1:1, since in some cases two different genotypes can lead to the same phenotype, this property known as redundancy.

Finally, it is worth mentioning that in GAs, the mapping function between a genotype and a phenotype must be provided by the researcher, and is problem-specific. In other words, while the phenotype is given by the problem, the genotype and how it is transformed into a phenotype is a design decision. There are two desirable properties of the genotype: it should limit redundancy as much as possible, hence reducing the search space given a fixed finite set of potential phenotypes; and it should enforce that similar genotypes represent similar phenotypes[7], since otherwise small mutations in the genotype could be the source of very large modifications of the individuals.

[6] In many cases, the phenotype is the name given to the candidate solution itself, although the terminology is a bit blurry.

[7] This is the reason behind the decision, prevalent in the literature, of representing integer numbers using Gray encoding when designing the genotype of a genetic algorithm.

2.11.2 *Grammatical Evolution*

Grammatical evolution (GE) is an evolutionary algorithm introduced by Ryan, Collins and O'Neill in 1998 [310]. The principles of GE are based on those of genetic programming, since the purpose of both consists on evolving computer programs.

To evolve programs, GE relies in a grammar provided by the researcher, which is often written in Backus-Naur form (BNF), a notation used to specify context-free grammars that generates formal languages. Although a detailed description of the concepts involved in the theory of formal languages and grammars is beyond the scope of this document[8], some of the basics required to understand this dissertation are the following:

- A context-free grammar is defined by the 4-tuple $G = (V, \Sigma, R, S)$.
- V and Σ are sets so that their union define the whole set of symbols available in the grammar.
- V is the set of non-terminal symbols.
- Σ is the set of terminal symbols.
- R is the set of production rules in the grammar. Each rule relates a single non-terminal symbol to a sequence of zero or more symbols in $V \cup \Sigma$.
- $S \in V$ is the start symbol, which serves as the root for deriving new strings from the grammar.
- Production rules are of the form $\alpha \to \beta$, where $\alpha \in V$ is a non-terminal and $\beta \in (V \cup \Sigma)^*$ is a string composed of non-terminals and/or terminals.
- A production rule $\alpha \to \beta$ is expressed in BNF as <α> ::= β, with non-terminals enclosed between '<' and '>'.
- The β in a production rule can be the empty string. This is often expressed as $\alpha \to \lambda$, and these rules are called λ-productions.
- Given two production rules of the form $\alpha \to \beta_1$ and $\alpha \to \beta_2$, these can be listed together as $\alpha \to \beta_1 | \beta_2$.
- Strings can be derived from a grammar by starting from S and applying rules until the string contains only terminal symbols in Σ.
- The language L generated by the grammar G is defined as the set of all possible different strings that can be derived from G: $L(G) = \{w \in \Sigma^* : S \overset{*}{\Rightarrow} w\}$.

In GE, phenotypes (or programs) are strings that belong to the language generated by the grammar provided. More precisely, the genotype in GE consists of a sequence of integer numbers, which are called codons. This list of numbers will be used to generate a string as follows:

1. Starting in S, the first number in the list allows to choose a rule to start deriving the string. If there are N rules of the form $S \to \beta_i$, then the first codon c_0 will be used to choose among these rules, such that the selected rule is $S \to \beta_i, i = c_0 \mod N$, where "mod" is the modulo operator (the remainder after division).
2. As the string is being formed, the previous process is repeated to expand the first non-terminal in the string, using the following codon every time.

[8] The founding fathers of this discipline are Chomsky and Schützenberger [65, 66].

3. If the whole chromosome has been traversed (i.e., the last codon has been already used for selecting a production rule) and the string still contains non-terminals, then the first codon is used again, thus working in a circular fashion.
4. The process will stop when a valid string, only formed by terminal symbols, is obtained. It is worth noting that some codons may remain unused.

In GE, genetic operators are also used, with selection, recombination and mutation being the most frequent, similar to the case of GAs. It is worth noting that unlike in GAs, the mapping function between phenotype and genotype is determined by the technique instead of by the researcher. On the other hand, the researcher must provide the formal grammar that generates the language of all possible genotypes.

Compared with GAs, grammatical evolution can lead to a more natural and flexible representation of individuals. On the other hand, small changes in the chromosome can produce larger modifications in the phenotype, and therefore the effect of recombination and mutation can have a larger impact than in genetic algorithms.

2.12 Summary

In this chapter we have provided a broad, yet quite detailed visit to neural networks and deep learning. Artificial neural networks are not new, in fact, the ideas behind them and early developments can be traced back to the mid-20th century. However, it was not until 1986 that they became a well-established machine learning technique, due to the discovery of backpropagation, the process for learning the network parameters in order to minimize the classification error.

More recently, the availability of big data, the improvement of hardware resources and the settlement of neural networks have given birth to a new field of study: deep learning. In deep learning, more complex neural network models are built, allowing for better results than those achieved by earlier models. One type of neural network architecture that is popularized in the era of deep learning is the convolutional neural network.

CNNs are networks that implement the convolution operator in the early layers in order to automatically extract relevant features from input data. By doing so, manual feature engineering is no longer required. CNN models have been applied to a great variety of fields, including computer vision or signals classification, and have achieved performances that had never been attained before with any other technique. Additionally, research in deep learning has introduced new activation functions, additional forms of regularization and novel optimization algorithms which are accelerating the discovery of new models and enhancing their efficacy.

Open-source communities have engaged in the development of deep learning frameworks that simplify the tasks of designing and building training neural network models even with complicated topologies which include convolutional and recurrent layers. Also, they enable simple and fast training of the network parameters which can be accelerated using GPUs. In many cases, these frameworks are being developed, promoted or adopted by key industry players.

Finally, we have provided a broad definition of evolutionary algorithms, which are optimization techniques that lie within the field of biologically-inspired, population-based metaheuristics. Evolutionary computation can be useful for choosing optimal (or good) solutions from a set of candidate solutions when no problem-specific heuristics are available, and we can only rely on some measure of quality of these solutions, which is called fitness. In this thesis, we will see how some of these optimization techniques (namely, genetic algorithms and grammatical evolution) can be applied to the design of deep convolutional neural networks.

Chapter 3

State of the Art

In this chapter we will first study the need for automatic design of convolutional neural networks in section 3.1, a need that constitutes the main motivation for this research work.

We will then explore some of the works that have already tackled this problem. There are many approaches to determine the best topology for multilayer perceptrons, and a selection of the most remarkable techniques have been reviewed in this chapter. Many of them adhere to the idea of neuroevolution, where evolutionary algorithms are used, and we review this field in section 3.3, whereas non-evolutionary approaches are discussed in section 3.4.

Only in the latest years some works have appeared trying to determine optimal architectures for convolutional neural networks. As a result, this area remains mostly unexplored and there is a high potential for research advancements in it. The works found in this area, which started to appear as recent as 2015, are described in section 3.5

Finally, we will summarize the review of the state of the art, analyzing some flaws or margins for improvement of current solutions, in order to better contextualize our proposal, which will be described in the following chapter.

3.1 Need for Automatic Design of Neural Networks

The latest edition of the Encyclopedia of Machine Learning and Data Mining [313] defines the topology of a neural network as the "*way the neurons are connected*" [241], and admits that it is "*an important factor in network functioning and learning*". However, this encyclopedic entry states that "*the most common topology in supervised learning is the fully connected, three-layer, feed-forward network*", ignoring later advances in deep neural networks and convolutional neural networks, where a larger number of layers (and thus of configurable hyperparameters) is used.

The importance of the topology in neural networks has been addressed several times in the past. One of the earliest studies was performed in 1989 by Baum and Haussler [18], where they suggested theoretical upper and lower bounds on the sample size versus the network size for the sake of improving generalization in networks with one hidden layer.

Also, Lawrence et al. [198] tried to study in 1996 the correlation between network size and optimal generalization. In their work, they support the observation that larger networks can produce better training and generalization error. However, their results also show that some oversized networks suffer from overfitting: an MLP with two hidden units outperformed 10 and 20 hidden units

when approximating the function $y = \sin(x/3)$ in the range $[0, 5]$. However, when a larger range ($[0, 20]$) was used, the 50-units MLP was the best performer. They also concluded that committees or ensembles can be more beneficial when input data is noisy. Prior to that, Caruana [52] reported that large networks rarely do worse than small networks, though he only studied a limited set of problems and this statement cannot be generalized to further domains.

More recently, in 2011, Hermundstad et al. [148] tested different architectures, from 'fan' architectures (one hidden layer with many units), to 'stacked' architectures (many hidden layers with very few units each), along with intermediate architectures. However, they kept the total number of weights as steady as possible (between 37 and 41). Even if there were very few hyperparameters and topologies were similar to some extent, authors concluded that "different network architectures produce error landscapes with distinguishable characteristics, such as the height and width of local minima, which in turn determine performance features such as speed, accuracy, and adaptability".

Some works have explored the impact on performance of different ANN topologies in specific domains. For instance, Choudhary et al. [67] looked for the influence of one versus two hidden layers in the scope of character recognition and concluded that the addition of one hidden layer led to higher accuracy of the neural network. In particular, accuracy in one test set improved from 65.38 % to 88.46 %, and in other test set from 80 % to 84.61 %. It is worth noting that this work is very limited, as researchers only compared one and two hidden layers using 10 hidden units in each one, and higher differences in accuracy could be expected when adding more neurons or further layers.

The truth is that, even if neural network topologies are an important factor in their performance, there is not a rule-of-thumb for determining what constitutes a good topology. Researchers in an old Usenet thread from 1995 maintained by Prof. Lutz Prechelt [286] expressed this fact in a quite far-from-subtle language [emphasis added]:

> *"There is no way to determine a good network topology just from the number of inputs and outputs. It depends critically on the number of training examples and the complexity of the classification you are trying to learn. There are problems with one input and one output that require millions of hidden units, and problems with a million inputs and a million outputs that require only one hidden unit, or none at all. Some books and articles offer 'rules of thumb' for choosing a topology –Ninputs plus Noutputs divided by two, maybe with a square root in there somewhere– but such rules are* ***total garbage****. Other rules relate to the number of examples available: use at most so many hidden units that the number of weights in the network times 10 is smaller than the number of examples. Such rules are only concerned with overfitting and are unreliable as well."*

As for convolutional neural networks (CNNs), recent works have explored the performance of diverse well-known architectures. A good benchmark is annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [309], as many different topologies have been tested over the years. For example, a 2016 work from Mishkin et al. [252] studies the influence of different hyperparameters in CNN topologies on the ILSVRC problem. This work is interesting as it explores how the accuracy is affected by the network width, batch size or activation functions. By the end of the paper, authors provide some recommendations for a good topology, based on the knowledge they acquired from evaluating the performance of several CNNs, yet these are only valid for the ImageNet domain.

An even more recent work by Canziani et al. [50] reviews the accuracy values reported in the literature for very relevant models in the ImageNet domain. Whereas the authors study the performance of these models in several dimensions (including accuracy, power consumption, speed or memory utilization), the most relevant results regarding our current research are summarized in figure 3.1. In the left side, we can see the accuracy of different deep neural networks models. It is

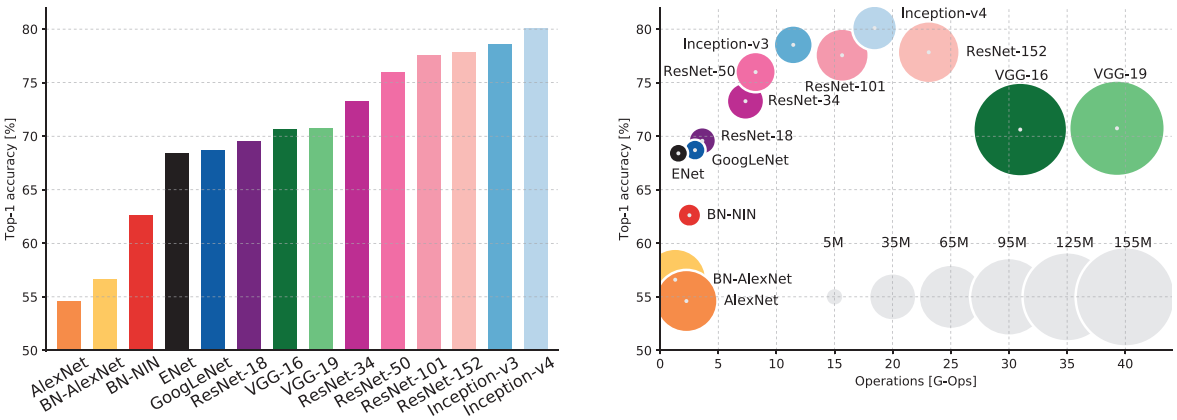


Figure 3.1: *Performance of various well-known CNN models in the ImageNet dataset. Source: Canziani et al. [50]*

remarkable that accuracy ranges from 54 % to 80 %, and it gets more impressive if we consider that these models have all been published in a four-years period: AlexNet was introduced in 2012 by Krizhevsky et al. [190], resulting in the first approach of convolutional neural networks to ILSVRC, whereas Inception-v4 was presented in 2016 [353] by Szegedy et al. from Google.

The right side chart shows the accuracy in the y-axis, the number of operations required for a single forward pass in the x-axis and the number of network parameters, which is proportional to the size of each blob. We can see how there is not a correlation between these three dimensions: larger networks, or networks with more operations do not imply a higher accuracy in all cases. The best performing network, Inception-v4, has an average size and number of operations.

Also in the domain of human activity recognition, Hammerla et al. [138] evaluate different deep learning models and conclude that convolutional neural networks show the most characteristic behavior with respect to the topology, as a fraction of model configurations do not work at all. They also state that functional setups show little variance in their performance, though we consider 7 percentage points in F1 score to be a remarkable variance.

In summary, the topology of a neural network is an important factor affecting its performance, and its impact is especially noticeable in convolutional neural networks. However, the influence of neural network design decisions can be even greater when recurrent neural networks (RNNs) come into play, as they enable many new applications when time series are available. The effectiveness of RNNs is well defined in Andrej Karpathy's blog [172]. An interesting conclusion is drawn in the recent work by Lipton and Berkowitz [221], who after reviewing three decades of research in recurrent neural networks, have concluded [emphasis added]:

> *"While LSTMs and BRNNs have set records in accuracy on many tasks in recent years, it is noteworthy that advances come from novel architectures rather than from fundamentally novel algorithms. Therefore,* ***automating exploration of the space of possible models****, for example via genetic algorithms or a Markov chain Monte Carlo approach,* ***could be promising****. Neural networks offer a wide range of transferable and combinable techniques. New activation functions, training procedures, initializations procedures, etc. are generally transferable across networks and tasks, often conferring similar benefits. As the number of such techniques grows, the practicality of testing all combinations diminishes. It seems reasonable to infer that as a community, neural network researchers are exploring the space of model architectures and configurations much as a genetic algorithm might, mixing and matching techniques, with a fitness function in the form of evaluation metrics on major datasets of interest."*

In this thesis, it is our aim to continue this community work, by letting evolutionary computation techniques search for outperforming topologies in different domains, comprising convolutional, feed-forward, and recurrent layers.

3.2 Early Approaches to Automatic Design of Neural Networks

In 1986, the discovery of the backpropagation process by David Rumelhart, Geoffrey Hinton and Ronald Williams [308] meant the awakening of artificial neural networks within the AI scene. Backpropagation was used for determining the best weights for a network in order to minimize a loss function between the computed output and the real output.

Few years later, due to the absence of an analytic procedure to compute the best topology of ANNs, there were an increasing interest on developing techniques for determining, or at least estimating, optimal neural network topologies. Some of the earliest approaches have tried to estimate the optimal topology for solving a certain problem using constructive or destructive algorithms. In the former, the process starts with a minimal network and new nodes and connections are added during the training phase, until performance stops improving (or starts degrading). As for destructive algorithms, the idea is similar yet starting with a oversized network from which nodes and connections are removed. Examples of these simple algorithms can be found in the works by Fahlman and Lebiere [98], Frean [106], Mozer and Smolensky [260], Sietsma and Dow [329] or Hirose et al. [151]. However, one the most widely cited works regarding destructive algorithms is that by LeCun et al. published in 1990 [205].

Another early contribution was published by Wang et al. [381] in 1994, though the article had been submitted to the journal two years before, in 1992. In this work, authors constrained their research to neural networks with only two hidden layers, and proposed an algorithm for determining the optimal number of hidden units by evolving the network topology during training using an algebraic approximation. Though they validated their approach by simulation over five noise-free and noise-corrupted datasets, they did not use more than 10 neurons per hidden layer, and thus their work explores a very small search space.

3.3 Neuroevolution

When talking of evolutionary design of neural networks, a very appropriate concept that must be introduced is "neuroevolution". The Encyclopedia of Machine Learning and Data Mining [313], defines neuroevolution as follows [240]:

> *"Neuroevolution is a method for modifying neural network weights, topologies, or ensembles in order to learn a specific task. Evolutionary computation is used to search for network parameters or hyperparameters that maximize a fitness function that measures performance in the task. Compared to other neural network learning methods, neuroevolution is highly general, allowing learning without explicit targets, with non-differentiable activation functions, and with recurrent networks."*

We can see how the current work can be placed within the field of neuroevolution. Notwithstanding, neuroevolution is a very general topic that embraces many different techniques. For example, neuroevolution could be used to evolve the weights of a neural network given a specific topology, when the activation functions of the neurons are non-differentiable. Also, it could be

used in problems other than supervised learning, when any arbitrary fitness function can be used to determine the performance of the network; e.g., in some problems of reinforcement learning. And of course, another application of neuroevolution is to search for the optimal topology of a neural network in order to maximize its performance. In the latter case, the network weights could be either evolved by neuroevolution as well, determined by a classical backpropagation algorithm, or learned in any other possible way.

3.3.1 Origins

The concept of what would later be known as "neuroevolution" arose in the late 1980s, with some authors motivating research in this area, such as Miihlenbein and Kindermann [239]. Most early works in neuroevolution are concerned with evolving the weights of a neural network. A straightforward approach to this problem is to encode in the genotype the sequence of weights of the network, either as a binary string or a list of floating point numbers. However, this approach is not convenient when the network has many hidden layers and hidden units, because of the large number of weights and, as a consequence, the large search space. Some of these early works are those from Montana and Davis [255] or Whitley and Hanson [388] in 1989.

By that time, the application of neuroevolution for learning the weights of a neural network was specially interesting for those cases where backpropagation was not a good choice, e.g., multi-layer and recurrent networks. Whitley et al. [387] already considered the evolution of multi-layer feed-forward networks in 1991. Regarding recurrent neural networks, some remarkable early works are those by Torreele in 1991 [365] or de Garis in 1992 [84]. In the works by de Garis, the term "GenNET" is used to refer to fully connected recurrent neural networks whose weights are evolved using genetic algorithms, and the author describes this paradigm as "*much more flexible and powerful than the traditional neural network paradigms*". Whereas most works involved genetic algorithms, in 1991 Scholz suggested an approach using a modified version of evolution strategies [322].

Besides evolving the weights of a neural network, evolutionary computation was used in the beginning for a diversity of tasks within the field of improving neural networks. For example, Harp et al. used genetic algorithms to find good values for the learning rate and decay in 1989 [141], finding that resulting values are higher than expected in some problems. Additionally, Belew et al. also used genetic algorithms to find a suitable initialization of the network for back-propagation in 1991 [21]. In 1990, Chalmers [54] used genetic algorithms in order to decide the best learning algorithm, with non-competitive results on feed-forward networks.

Finally, some early works were also concerned with designing the topology of the neural network. For example, already in 1989 Miller et al. [245] suggested the use of genetic algorithms for evolving the network structure. There is an explosion of this area of research in the early 1990s: Harp et al. [142] described NeuroGENESYS, a genetic algorithm for learning the network structure and some additional learning hyperparameters, yet using backpropagation for learning the weights; and a similar approach was also described by Schaffer et al. [318]. Also in that year, Kitano [184] suggested an alternative encoding based on graph generation grammars to evolve the network architecture using genetic algorithms. Schiffmann et al. [321] compared fixed and evolved network topologies with an application to handwritten digits recognition in 1991.

The first extensive survey in this area had been provided by Schaffer et al. [319] already in 1992, in an international workshop on combinations of genetic algorithms and neural networks. It is remarkable that, as soon as in 1992, there was so much research interest in this field. By that time, some authors were already working in the evolutionary design of the neural network structure; e.g.: Hancock [139] explored the performance of different recombination operators when evolving the network structure, Elias [95] described the use of a genetic algorithm to evolve the connection patterns of a neural network implemented over analog hardware, Dasgupta and McGregor [81] used

structured genetic algorithms for the evolution of both the weights and topology of application-specific neural networks, Karunanithi et al. [173] described the use of genetic cascade learning to improve the network topology by adding one hidden unit at a time, and Lindgren et al. [217] evolved the topology and weights of a neural network for regular language inference.

It is worth noting how only six years after backpropagation was introduced, there was a large community of researchers addressing the issues found with backpropagation and applying evolutionary computation techniques in order to determine the weights and topology of neural networks. A summary of these works was published in 1995 by Balakrishnan and Honavar [11], according to a taxonomy based on the genotype representation, the network topology, the variables of evolution and the application domain.

3.3.2 Concepts of Neuroevolution Applied to Topology Design

An additional review of the state of the art was provided one year later, in 1993, by Yao [400]. In this work, the author distinguishes three different types of works within the field of evolutionary artificial neural networks: evolution of weights, of architectures, and of learning rules; and studies the interactions between these problems.

Yao's work also settles some of the basic concepts of neuroevolution (though he refers to this concept as "*evolutionary artificial neural networks*", EANN). First, Yao described the need for automatic design of neural networks architectures, in similar terms than the ones we have used before:

> *"But how do we decide EANN architecture? It is well known that EANN architecture has significant impact on EANN information processing abilities. Unfortunately, EANN architecture still has to be designed by experienced experts through trial-and-error. There is no systematic way to design an optimal (near optimal) architecture for a particular task."*

Yao also agrees with Miller et al. [245] in that GA-based approaches are suitable for finding optimal solutions within the search space, i.e., the set of candidate solutions composed of all possible ANN architectures, because of the next characteristics of the search space:

- It is potentially infinite, since the number of possible nodes and connections is unbounded.
- It is non-differentiable, as changes in the number of nodes or connections are discrete but can have a continuous effect on the network performance.
- It is complex and noisy, because the mapping between a network and its performance is indirect and stochastic due to the randomness of initial weights.
- It is deceptive, since similar network architectures can lead to very different performances.
- It is multimodal, since very different architectures can have similar performance.

After addressing the advantages of using evolutionary computation for determining optimal or near-optimal topologies for artificial neural networks, Yao establishes a taxonomy of works on neuroevolution based on the encoding: direct or indirect. He also insists on the importance of evolving the learning rules.

Direct Encoding In direct encoding, a binary chromosome specifies whether a connection between two nodes exists or not. To use this schema, we need some previous knowledge about the problem in order to determine a maximal topology. The maximal topology can consist of as many layers and hidden nodes as desired. Once the maximal topology comprising N nodes is established, each neuron is numbered and a binary matrix $C = (c_{ij})_{N \times N}$ is created.

A '1' in the position c_{ij} indicates that there is a connection from neuron i to neuron j, where as a '0' indicates that such connection does not exist. Additional constraints can be imposed, for example to guarantee a feed-forward neuron (only enabling connections from nodes in one layer to nodes in the following one). These additional constraints can be observed in the genotype by removing genes c_{ij} so that j is in not in the following layer than i.

While this encoding is very natural, it entails two handicaps: first, certain assumptions must be made about the topology of the network ahead, in order to determine the maximal topology. Second, unless certain constraints are imposed (which requires even more knowledge and prior decisions on the topology), the size of the chromosome grows quadratically with the number of nodes in the network, posing a $\mathcal{O}(n^2)$ complexity.

Indirect Encoding In indirect encoding, some important features of the neural network topology are considered in the chromosome, instead of the full connectivity pattern. It leads to a more compact encoding when compared with the direct one. Yao describes three main approaches to this encoding that had been explored as of 1993: connectivity hyperparameters, developmental rules and, to a lesser extent, fractal representations of connectivity.

Yao also remarks that some works also encodes learning hyperparameters, such as the learning rate. Some have been mentioned in the previous section; e.g., the work by Harp et al. [141] in 1989.

Evolution of Learning Rules Yao recognizes that, as of 1993, there was very few work done on evolving learning rules, since most works tackled the evolution of connectivity. However, according to Mani [228], learning rules have an important impact on the performance of neural networks.

3.3.3 *Remarkable Milestones on Neuroevolution*

Since the mid-1990s there have been a lot of research works and different approaches to the evolution of neural networks. Since we want to keep this section as complete as possible, yet succinct enough to maintain its readability, we will focus on some of the most relevant milestones on this topic. When applicable, we will refer the reader to extensive reviews of the literature, in order to gain further insights of the historical evolution of the field.

In 1994, Frédéric Gruau [132] presented his doctoral dissertation, where he suggests representing neural networks via a cellular encoding, using grammar trees to describe the network's structure; thus using an indirect representation. These trees are optimized using a genetic algorithm, and Gruau concludes that his approach is indeed an application of genetic programming. In Gruau's thesis, both network structure and parameters are evolved over time.

Also that year, Angeline et al. [6] presented GNARL, which stands for GeNeralized Acquisition of Recurrent Links, an evolutionary programming-based approach with direct encoding for evolving the structure and weights of a recurrent neural network. In evolutionary programming, the recombination operator is not used, and only mutation is performed to obtain new individuals. In GNARL, the number of input and output neurons is defined by the problem, and the number of

hidden units varies from 0 to a user-defined maximum $h_{\max}$. Neurons are not previously assigned to layers, so there can be any recurrent connectivity pattern. In fact, individual nodes or groups of nodes can remain disconnected from the input and output neurons, thus being ignored when constructing the neural network.

In 1997, Vonk et al. [376] published a book describing the application of evolutionary computation in the automatic generation of neural network architectures. The main techniques they studied were genetic algorithms to optimize the weights of the network, and genetic programming and genetic algorithms with grammar encoding (thus, indirect encoding) for generating the network topology. As for genetic programming, two years before they had presented GPNN [377], though they had only tested their approach in two simple problems (XOR and one-bit adder) and admitted that the system did not scale out well for real world problems.

EPNet Also in 1997, Yao and Liu [402] proposed EPNet, a system based on evolutionary programming for evolving artificial neural networks using direct encoding. In the introduction, authors made an statement consistent with the motivation of this thesis:

> *"The problem of designing a near optimal ANN architecture for an application remains unsolved. However, this is an important issue because there are strong biological and engineering evidences to support that the function, i.e., the information processing capability of an ANN is determined by its architecture."*

In EPNet, both architecture and weights are evolved simultaneously, and authors noted that they put their focus on evolving the behavior of neural networks, keeping a behavioral link between parents and offspring during the evolution process. This is one of the motivations for which they chose evolutionary programming over genetic algorithms. Because they use direct encoding, the user needs to specify a maximum number (N) of hidden nodes allowable in the network. The specification of the neural network requires two matrices, each of them with dimension $(m + N + n) \times (m + N + n)$, being m the number of input nodes and n the number of output nodes; and a binary vector of length N. The first matrix is a binary connectivity matrix determining the existence or not of a link between two nodes, whether the second matrix specifies the connection weights. The vector specifies whether nodes exist or not in the network. Because Yao and Liu decided to constrain the search space to feed-forward networks, some constraints can be imposed on these matrices: only the upper triangle of the matrix will be used, and the connections between input nodes will be enforced to 0.

The fitness function consists of the inverse of an error metric computed over a validation set. Authors insisted on the importance of this decision in order to improve generalization ability.

One of the main innovations of EPNet is the mutation scheme: Yao and Liu came up with a sequence of mutations that were only executed if the previous action did not improve the network performance. This sequence comprises the next stages:

1. Hybrid training of the connection weights: first using backpropagation with a custom learning rate determined following a heuristic based on the network performance and, if it does not improve the network performance, using simulated annealing.
2. Hidden node deletion: one or more hidden nodes are deleted by setting the corresponding bits in the hidden nodes vector to 0.
3. Connection deletion: several connections are deleted by setting the corresponding bits in the connectivity matrix to 0. Connections are not chosen randomly, but are rather selected based on their importance.

4. Connection and node addition: new connections are added with a small random weight, and nodes can also be added using cell division [274].

It is worth noting that the mutation scheme in EPNet gives preference to changes in the connection weights rather than to architectural modifications. Also, when the topology is mutated, removal of nodes and connections is preferred over addition, in order to reduce the size of neural networks during evolution. Finally, after evolution further training is performed using both the training and validation sets with the backpropagation algorithm using custom learning rate.

After proposing EPNet, Yao and Liu validated their proposal using different real-world problems, including: the N parity problem, the two-spiral problem, medical diagnosis problems (including breast cancer, diabetes, heart disease and thyroid data sets), the Australian credit card assessment problem and MacKey–Glass chaotic time series prediction problem. Authors concluded that EPNet led to very competitive results because of the few constraints imposed on the network architectures, thus resulting in a large search space. However, they admitted that EPNet involved many user-defined hyperparameters.

Two years later, in 1999, Yao [401] reviewed different alternatives of the application of evolutionary algorithms to the evolution of neural networks and pointed out some future research directions for the new millennium. Also, Yao established a general framework for evolutionary artificial neural networks in different levels, as shown in figure 3.2, observing the cases where evolutionary

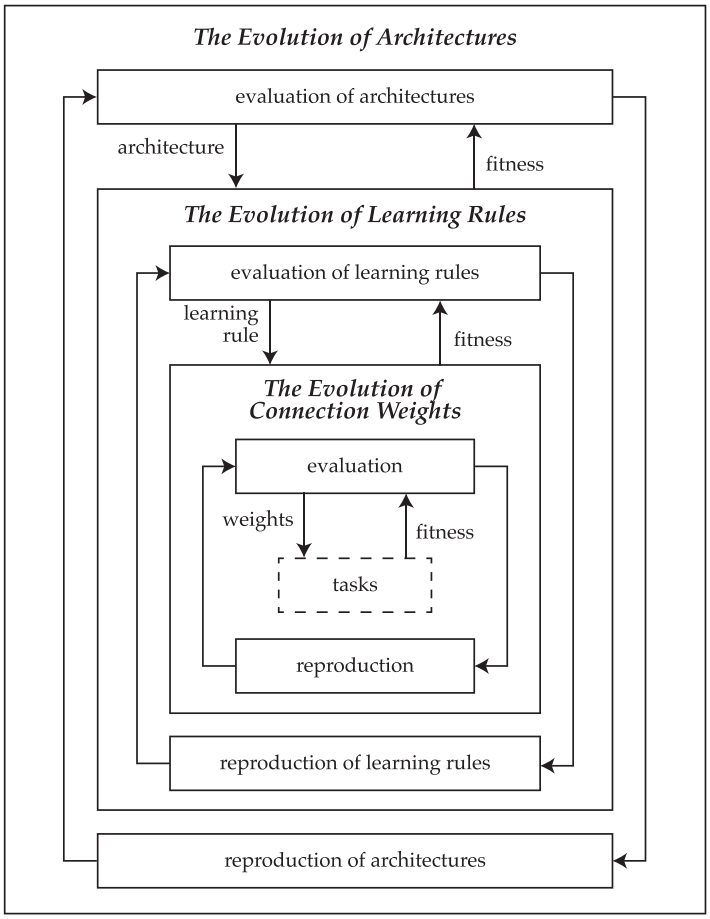


Figure 3.2: *General framework for evolutionary artificial neural networks. Source: Yao [401].*

computation can be used to optimize some of the dimensions (e.g. architecture and learning rules), using backpropagation for learning the network weights. Finally, Yao concluded that using evolutionary algorithms at the three levels can be highly computationally expensive, and suggested that it is a better idea to use these techniques only in some levels, especially in those where there is few knowledge about the optimal hyperparameters beforehand, as a trial-and-error or other heuristic methods are very ineffective for searching in these cases.

NEAT In 2002, Stanley and Miikkulainen [345] presented NEAT (neuroevolution of augmenting topologies), a solution which would become one of the most cited and used systems in neuroevolution. This is indeed one of the first works in which the term "neuroevolution" is used, as opposed to "evolutionary artificial neural networks", the term used by Yao in previous works. We think that the success of NEAT helped the term "neuroevolution" to settle as the *de facto* standard for referring to this area of knowledge[1].

NEAT evolves both the topology and the weights of the neural network using genetic algorithms with a direct encoding. Unlike in the working scheme of evolutionary programming, genetic algorithms include a recombination operator in order to perform the crossover between two parents to generate offspring. For this reason, the encoding must be thought in order to ease recombination, and authors stated that NEAT's encoding eases lining up corresponding genes when two genomes cross over during recombination. As shown in figure 3.3, networks are encoded using two vectors: one for nodes and other for connections. The nodes vector includes a list of input, hidden and output nodes, which can be connected. The connections vector include the input and output nodes of the connection, its weight, whether it is enabled or not, and a so-called *innovation number*.

Regarding the mutation operator, it can affect both the connection weights and the network structure. In the former case, weights evolve just following the standard mutation scheme from genetic algorithms. In the latter case, structural mutations can either add connections or add new nodes. When adding connections, the size of the connections vector is increased in order to add a new connection with a random weight between two existing unconnected nodes. Adding nodes is slightly more complicated: a new position is added to the nodes vector and then, a random

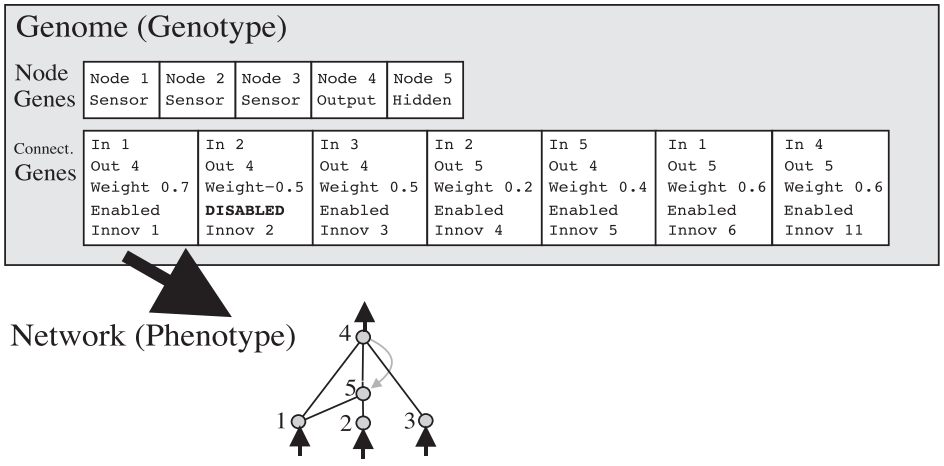


Figure 3.3: *Genotype and phenotype mapping in NEAT. Source: Stanley and Miikkulainen [345].*

[1]We have been unable to find the first appearance of this term with this meaning, though apparently Miikkulainen popularized the term, as it had been used in at least two works by Gomez and him before, the first in 1999 [114, 115].

connection c is chosen. To be more specific, let us consider n_i to be the input node of connection c, n_o to be the output node of that connection, and n_m the new node created during mutation. Then, connection c is disabled in the connections vector and two new connections are added: c_1 linking n_i to n_m, and c_2 linking n_m to n_o. According to the authors, these mutation rules minimize the initial effects of the mutation, as otherwise new extraneous structures would be added to the network.

As stated before, all connections' genomes include a so-called *innovation number*. This number is an incremental value that is added to each connection as it is created. The reason behind these numbers is that they represent the historical origin of each gene, indicating at which stage of the evolution process the gene appeared. This is because, during crossover, offspring inherit the same innovation numbers that their parents. In order to perform crossover, two individuals are lined up based on their innovation numbers, and offspring are composed by randomly choosing from either parent at matching genes. Disjoint and excess genes (those in one parent with innovation numbers not present in the other parent) will always be included from the fittest parent.

According to the authors, a problem with this approach is that recently augmented topologies often perform worst, and have little hope of surviving more than one generation even if their phenotype entails crucial innovations towards finding the optimal topology. In order to solve this problem, they propose a way of protecting innovation, namely "*speciation*". By speciating the population, individuals compete mostly within their own niche instead of with the whole population. Thus, the idea behind speciation is to divide the population into niches conformed by individuals representing similar topologies. Authors state that, while this problem can be seen as a topology-matching problem, innovation numbers enable an efficient way to estimate the similarity of two genotypes: the more excess and disjoint genes two genomes contain, the less compatible they are. After speciation, reproduction is performed using *explicit fitness sharing* [113], where individuals belonging to the same specie share the fitness of their niche, promoting small niches.

It is worth noting that, as opposed to EPNet [402], the mutation scheme in NEAT always increases the network size; thus the acronym NEAT standing for "*augmenting topologies*". According to authors, this enables starting the search in a search space of reduced dimensionality, increasing the search space only when required and leading to minimal topologies. After evaluating NEAT's performance, Stanley and Miikkulainen concluded that NEAT was a powerful method for artificially evolving neural networks, being more efficient than other neuroevolution techniques.

The approach used in NEAT has led to some novel techniques such as HyperNEAT by Stanley et al. in 2009 [344], where an indirect encoding is used in order to evolve *compositional pattern-producing networks* (CPPNs), enabling the efficient representation of large-scale neural networks (with over eight million connections); or ES-HyperNEAT by Risi and Stanley in 2012 [302], a work similar to the former but automatically deducing node geometry.

EANT In 2005, Kassahun and Sommer introduced EANT (evolutionary acquisition of neural topologies) [175], a work closely related to the works by Stanley and Miikkulainen [345] and by Igel [161]. In particular, they evolve the network starting from a minimal topology and the weights using an implementation of an evolution strategy called CMA-ES. It is worth noting that EANT was designed to apply the neural networks to reinforcement learning problems.

However, authors consider that the main contribution of their work is the encoding, which allows to be evaluated without decoding it. In particular, the genome in EANT is a linear sequence of genes which can represent different entities of the neural network: either a neuron, an input neuron, or a connection between two neurons, which can be either a feed-forward connection or a recurrent connection. All genes store the weight between the neuron they represent and the neuron to which it is connected. Also, connection genes (called "jumper genes" by the authors) also contain one number specifying the neuron to which it is connected. An example of the mapping between the genotype and the phenotype in EANT is depicted in figure 3.4.

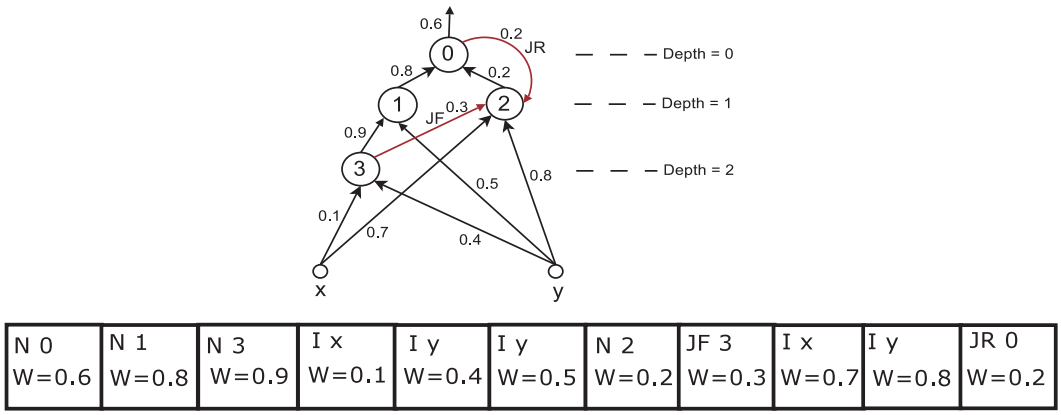


Figure 3.4: *Genotype and phenotype mapping in EANT. Source: Kassahun and Sommer [175].*

As stated above, EANT enables the evaluation of the network represented by a linear genome without decoding it. The process for evaluating a genotype is non-trivial and is exhaustively described in the original paper by Kassahun and Sommer [175]. The main advantage of their work is that they propose a single theoretical and mathematical framework for both direct and indirect encoding, called *common genetic encoding* (CGE). This framework is complete, as it is able to represent all possible phenotypes, and is closed as every valid genotype represents a valid phenotype. The previous properties of CGE have been formally proved by Kassahun et al. in 2007 [174].

For the evolution of the neural network topology, structural mutation is carried out, which is able to add or remove connections as well as to add sub-networks. As for the evolution of weights, the CMA-ES implementation of a evolution strategy is used.

In 2007, Siebel and Sommer proposed EANT2 [328], where the efficiency of EANT for searching the space of topologies and weights was improved. EANT2's performance was evaluated by evolving a network which must control a robotic arm equipped with a camera in order to steer the arm to a certain object. They compared the performance of their system with NEAT's, showing that EANT2 obtained a better maximum fitness.

Other works In this section we have described some of the most remarkable works in neuroevolution; however, this field is highly prolific and many works have been published throughout almost three decades of research activity. A remarkable work because of its relevance to this thesis was published by Tsoulos et al. in 2008 [366], where they used grammatical evolution in order to optimize both the topology of the neural network and its weights, developing a system known as NNC. However, this approach is constrained to 2-layer networks. Authors tested their proposal in 18 well-known supervised learning problems, half of them being classification problems and the other half being regression problems, reporting a very good performance of NNC.

Many other significant works and contributions on neuroevolution have been gathered in extensive surveys of the state of the art; e.g. Floreano et al. in 2008 [103] or Ding et al. in 2013 [90].

In the latest years, some novel open-source frameworks have appeared in order to perform neuroevolution. The availability of source code enables researchers to work on their own implementations and applications of neuroevolution. One of these frameworks is MABE (modular agent-based evolution framework), whose source code is currently maintained and receives frequent contributions as of June 2018 [150]. The foundations of the MABE framework were presented in 2011 by Edlund et al. in PLOS Computational Biology journal [94]. The framework is general enough as to support different encoding methods and optimization techniques.

Another relevant open-source framework is Physis-Shard, made available in 2016 [414]. Physis-Shard is based on an agent called SUNA (Spectrum-diverse Unified Neuron Evolution Architecture), whose details where published by Vargas and Murata in 2016 [372]. In particular, SUNA proposes a genome representation that incorporates most of the neural networks features described in the literature, optimized via a genetic algorithm implementing a diversity preserving mechanism. Authors reported a performance over five different problems that was at least as good as NEAT's.

3.3.4 Extensions and Innovations on Neuroevolution

In the previous section we have outlined some of the most remarkable works in the field of neuroevolution. Along three decades, some extensions and innovations have been incorporated to these works in order to improve their performance or efficiency.

A remarkable work due to its close relation to this thesis is the one published by Liu et al. in 2000 [224]. As far as we know, their work was the first to suggest the evolution of neural network ensembles, a very interesting approach since ensembles or committees of neural networks will often outperform the behavior of the individual networks they comprise. In their work, Liu et al. used evolutionary programming to evolve a population of neural networks and then used negative correlation learning to choose the most convenient set of individuals to build the ensemble.

Additionally, the Encyclopedia of Machine Learning and Data Mining [240] gathers some of the main extensions in neuroevolution. Most of the extensions that had been successfully applied to evolutionary computation techniques are suitable for neuroevolution; to mention a few, the work by Igel [161] employs an intelligent mutation technique, which is suitable because weights are often correlated; works by Stanley and Miikkulainen [346] and Chellapilla and Fogel [59] have focused on approaches to neuroevolution using coevolution, for example, evolving the behavior and environment where the network is applied as it gets more complex; and Lehman and Stanley [209] and Mouret and Doncieux [257] have defined behavioral diversity and novelty in terms of the behavior of the neural network.

Other extensions involve the evolution of networks with modular architectures, which can show multimodal behavior which translates into robust agents, arising from a combination of low-level behaviors. Some works in this area have been described by Clune et al. [72] or by Schrum [323] in his Ph.D. dissertation, whose supervisor was Miikkulainen.

Finally, some extensions are aimed towards guiding or driving the learning process using human knowledge, for example by incorporating human-coded rules in the network topologies during mutations, biasing the initial population or the evolution process, either in the evolutionary operators, the fitness function or the learning rules. An example work of this kind of innovations is provided by Bryant and Miikkulainen [47].

3.3.5 Cutting-Edge Applications of Neuroevolution

While so far we have explored a great diversity of literature in the design of neuroevolution techniques, it should be obvious that neuroevolution is a field that can be applied to many different areas. When we say that we want to "evolve the topology or the weights of a neural network", we refer that we want to obtain a network that behaves well in a given problem. Because of the nature of artificial neural networks, the fields of applications are very diverse and comprises problems of supervised, unsupervised, and reinforcement learning.

Of course, the number of works that can be found along three decades where neuroevolution is applied to different problems is immense. In this section we will mention a small subset of

them, remarkable due to their impact or recentness, for the reader to grasp an idea of the field's importante. An exhaustive listing does not lie within the scope of this dissertation.

Some applications of neuroevolution to the field of video games can be found in different works, in order to incorporate complex, intelligent, non-predefined behaviors. A sample work in this area was published by Miikkulainen et al. [242], where evolution is guided using human-knowledge to improve the behavior of non-playable characters, or by Samothrakis et al. [314], where general video games playing agents are developed through neuroevolution. Risi and Togelius [303] have recently published a survey of neuroevolution in games in five different axes: state/action selection, direct action selection, modeling player experience, content generation and strategy selection.

Also, the Encyclopedia of Machine Learning and Data Mining [240] does stress that neuroevolution is very powerful for real-world applications of reinforcement learning, as the control of adaptive physical devices. To this extent, there have been works focused on evolving multi-legged robots (Valsalam et al. [370]), specialized racing cars (Togelius and Lucas [364]), vehicles that navigate intersections (Nitschke and Parker [268]), or rockets (Gomez and Miikkulainen [117]). In 2000, Nolfi and Floreano [269] published an exhaustive book about the field of evolutionary robotics. In 2011, Bongard published a work in PNAS [40] examining the evolution of robots during their lifetime and comparing this process with biological evolution. An obvious limitation of these applications is that controllers must often be simulated in order to proceed with neuroevolution, due to the difficulty of evolving physical systems and the advantage of parallelism when evaluating solutions.

Another relevant field of application of neuroevolution is artificial life, since neural networks can implement behaviors. For example, in the early days of neuroevolution, Werner and Dyer [386] explored the evolution of communication. More recently, Lessin et al. [210] and Bongard [40] have explored the joint evolution of morphology and control to create agents with natural movement; and Keinan et al. [178] have described the analysis of neurocontrollers to better understand how evolved circuits map to functions.

Neuroevolution approaches for tackling supervised learning problems are less frequent than for reinforcement learning; however, some applications can be found. Some recent works have focused on the prediction of time series, such as those by Peralta et al. [280], by Chandra [56], by Wong et al. [392], or by Nand and Chandra [262].

Finally, many applications of neuroevolution to medical and biological domains have arisen recently; e.g., Limache and Portugal-Zambrano [215] evolved a feed-forward network to discriminate normal vs. abnormal digital brain images, Grisci and Dorn have applied neuroevolution to predict the 3-dimensional structure of protein sequences [129] and to predict the conformational flexibility of amino acids [130], and as of July 2017 Khan et al. [181] have evolved the parameters of wavelet neural networks using Cartesian genetic programming to do classification on a breast cancer dataset and a recent dataset on Parkinson's disease.

3.4 Non-Evolutionary Alternatives for Automatic Design of ANNs

So far, we have seen many different techniques, innovations and applications within the field of neuroevolution. Neuroevolution has proven to be a very convenient approach to estimate near-optimal topologies of ANNs, and in some cases also to evolve their parameters or learning rules.

However, in some cases, authors have decided to use other, non-evolutionary methods in order to automatically design neural networks architectures that fit a given problem. This section outlines some of these techniques; however, it is not intended to provide an extensive survey or in-depth detail about these non-evolutionary approaches.

As we described earlier in this chapter, some of the first approaches to the automatic design of neural networks topologies were either constructive or destructive methods. These methods were indeed simple ways to automate the trial-and-error procedure by adding or removing neurons. However, these works mostly focused on networks with only one hidden layer, which were most common networks by that time.

More sophisticated works, such as Khaw et al. [182] or Balestrassi et al. [16], have applied a statistical methodology known as "Design of Experiments" (DOE) to determine the best hyperparameters for a neural network. In particular, Balestrassi et al. [16] have applied the network to a nonlinear time series forecasting problem comprising six time series representing the electricity load of a production company in Brazil. They found factorial DOE using screening, Taguchi, fractional and full factorial designs to be a better approach than classical trial-and-error.

Whereas evolutionary algorithms have been widely used, Sossa et al. [338] tested non-evolutionary yet biologically-inspired techniques (such as particle swarm optimization or artificial bee colony) in order to automatically design neural networks. They reported promising results, yet using the very simple iris plant classification problem.

Finally, in the latest years, one work found in this area was published in 2016 by Mariyama et al. [229] who have used the *add-if-silent* function found in the neocognitron to automate the design of small neural topologies based on training data.

To conclude, it is worth noting that these approaches are a minority when compared to neuroevolution-related works: when looking for automatic design of neural network topologies, most works rely on evolutionary algorithms.

3.5 Automatic Design of Deep and Convolutional Neural Networks

So far, we have described almost three decades of research in the field of evolving the topology of neural networks. However, during most of this time neural networks were relatively small, containing only one hidden layer with few hidden units in the case of feed-forward networks or few recurrent connections, due to the limitations in computational power.

The emergence of deep and convolutional neural networks in recent years have brought back the need for coming up with topologies that are suitable to tackle specific problems. However, deep networks can have dozens of feed-forward or recurrent hidden layers with thousands of units each, where each neuron can implement one among different activation functions. Also, convolutional neural networks can also have several convolutional layers with thousands of kernels of various sizes, besides from many different pooling and padding setups. As a result, deep and convolutional neural networks can have millions of parameters and hyperparameters, and innovations must be introduced in neuroevolution for it to adapt to these new architectures.

To the best of our knowledge, the number of works in this area is very reduced, mainly because of the novelty of convolutional neural networks and the computational cost of training and testing the performance of these networks. In this section we will extensively cover the works in this field.

An early approach of neuroevolution to deep learning was proposed by Koutník et al. in 2014's edition of GECCO [186]. In the abstract, authors stated that their work "is the first use of deep learning in the context of evolutionary reinforcement learning", and to the best of our knowledge, it is also the first attempt (at least published in the proceedings of a flagship conference) to evolve convolutional neural networks; although earlier works had used grid search or Bayesian optimization to find a small set of optimal hyperparameters (Snoek et al. [335] or Bergstra et al. [29]). However, in this work the topology of the neural network is not encoded, but rather a fixed

architecture is used comprising four convolutional layers with max-pooling and finally a small recurrent network with three hidden units. The weights of the convolutional layers, that eventually output feature vectors from raw inputs, and the weights of the recurrent neural network are learned separately. In the former case, 993 weights were optimized to maximize the variance of output representations, encoding them in a real-valued genome evolved using CoSyNE [116]. As for the recurrent neural network, it comprises only 33 weights, which are encoded and evolved using the same mechanism; while fixing a sigmoid activation function.

Also in GECCO 2014, David and Greental [82] proposed a method assisted by a GA to evolve the weights of an autoencoder. In their work, they evolve a stack of five layers, each of them separately, i.e. only when one layer is evolved does the algorithm start with the following. It is noticeable that the architecture is fixed, and only weights are evolved. Authors reported that neuroevolution outperformed backpropagation, although they used the output of the autoencoder with a SVM classifier and attained a test error rate over MNIST of 1.44 %, which is quite large.

Another work was published by Verbancsics and Harguess in 2015 [373], where they proposed a modification on HyperNEAT [344] to support the evolution of convolutional neural networks, by adding a new CNN substrate able to represent this kind of networks. Unfortunately, authors describe their approach with very few detail, and reproducibility of their implementation seems unfeasible. Moreover, their experiments using the evolved convolutional neural network over the MNIST dataset led to a test error rate of 7.9 %, which is one order of magnitude higher than the performance of most CNN-based works. Thus, this work does not seem reliable or successful in their aim of evolving deep neural networks.

In EvoCOP 2015, Desell et al. [89] proposed the evolution of deep recurrent neural networks using ant colony optimization, not observing convolutional layers and working with a fixed topology of five hidden and five recurrent layers. In their proposal, ants selected a forward propagation path given a fully connected network, based on the amount of pheromone in connections.

Also by the end of 2015, Young et al. [404] introduced MENNDL (multi-node evolutionary neural networks for deep learning), a framework for optimizing the hyperparameters of a neural network using genetic algorithms, with a focus on high performance computing. Their approach, however, turns out to be extremely simple as only six hyperparameters are evolved: the number of filters and the filter size for a fixed 3-layers CNN architecture. Thus, the search space is very limited and their proposal remains very inflexible. More recently, in 2017, Young et al. [403] suggested an improvement over their previous version, again with a focus in high performance, with selection being performed as soon as one third of the population has been evaluated. In this improved version, a variable number of layers is supported, by using an on-off bit in the chromosome for each of the layers. Also, different types of layers are supported: convolutional, pooling, and fully connected, and activation functions are evolved as well for each of them.

During 2016, three related works were published. The first was published by Loshchilov and Hutter [225] where they propose using the evolution strategy CMA-ES to evolve the hyperparameters of a deep neural network. In particular, 19 hyperparameters are considered including optimizer hyperparameters (learning rate, momentum, etc.), batch size, dropout rate, number of filters in the convolutional layers or number of units in the fully connected layer. However, the number of layers is fixed prior to the evolutionary process, and most of the hyperparameters involved are related to the optimizer rather than the topology. In fact, neither filter sizes or activation functions are evolved, and recurrent layers are not included in the search space. Finally, though they report the performance on the MNIST dataset, they seem to be using a validation set which is different from the standard test set, and thus a fair comparison is not feasible.

The second work was published by Fernando et al. [100], from DeepMind, suggesting the creation of a differentiable version of a CPPN, called the DPPN. These DPPNs are created using microbial genetic algorithms, and eventually they are able to replicate CNN architectures.

The third work was published by Tirumala et al. [363], who suggest the use of a co-evolutionary algorithm, comparing both a competitive and a cooperative version. Unfortunately, the process is not described with great level of detail, but authors are only evolving the weights of a fixed architecture with 5 fully connected layers. Authors report a test error rate over MNIST of 1.3 % with cooperative co-evolution and 3.7 % with competitive co-evolution. It is worth mentioning that this work does not involve the use of convolutional neural networks.

Most of the works we have been able to find where the topologies of convolutional neural networks were automatically designed or evolved have been published during 2017 and 2018. Moreover, some of these works are only available in pre-print repositories, such as arXiv, and have not been published in peer-reviewed journal or conferences proceedings. Next, we will describe each of these works in further detail.

MetaQNN The work by Baker et al. [10] introduces MetaQNN, where reinforcement learning is used to evolve the CNN architecture. Authors motivate their work by stating that "*at present [...] new architectures are handcrafted by careful experimentation or modified from a handful of existing networks*".

MetaQNN uses Q-learning to search within the space of network architectures, and the training task is performed by sequentially choosing neural network layers. Figure 3.5 shows an overview of the reinforcement learning process, whereas figure 3.6 shows the Markov decision process used for the Q-learning algorithm, where the agent selects a new layer in each action.

Regarding the state space, each state is defined as a tuple including all relevant hyperparameters. Layers can be either convolutional, pooling, fully connected, or final layers (which at the same time can be global average pooling or softmax). The hyperparameters for each of these different layer types are the following:

- Convolution: layer depth ($i < 12$), receptive field size (squared, $f \in \{1, 3, 5\}$), number of receptive fields ($d \in \{64, 128, 256, 512\}$) and representation size ($n \in \{(\infty, 8], (8, 4], (4, 1]\}$).

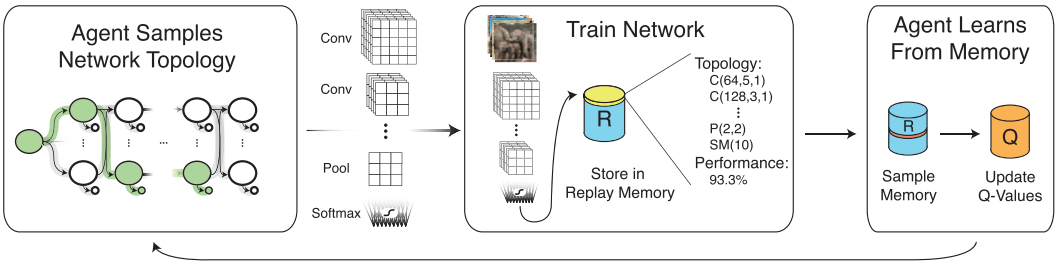


Figure 3.5: *Learning process in MetaQNN. Source: Baker et al. [10].*

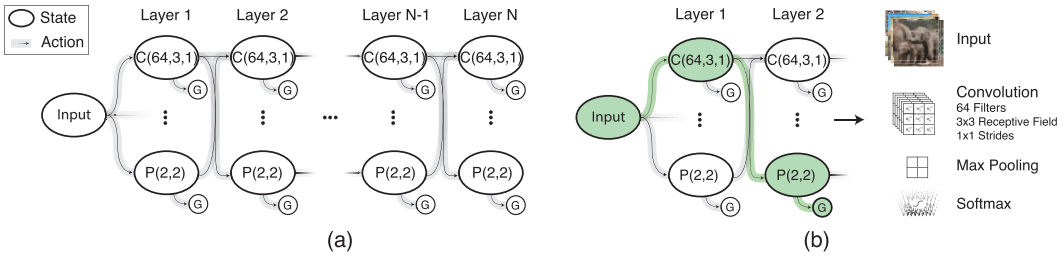


Figure 3.6: *Markov decision process for Q-learning in MetaQNN. Source: Baker et al. [10].*

- Pooling: layer depth ($i < 12$), receptive field size ($f \in \{(5,3),(3,2),(2,2)\}$), stride ($l \in \{(5,3),(3,2),(2,2)\}$) and representation size ($n \in \{(\infty,8],(8,4],(4,1]\}$).
- Fully connected: layer depth ($i < 12$), number of consecutive fully connected layers ($n < 3$) and number of neurons ($d \in \{128,256,512\}$).
- Termination: type (t), which can be either global average pooling or softmax.

As for the action space, the agent is restrained from executing certain actions. More specifically, the next conditions must hold:

- Only transitions from a state with layer depth i to states with layer depth $i+1$ are allowed. For this reason, the maximum number of layers of any kind is set to 12. States with maximum layer depth can only transition to a termination state.
- The maximum number of fully connected layers is set to two, to prevent the number of parameters from growing very quickly. To do so, the agent can only move from one fully connected state to another if n is less than the two.
- The agent in a fully connected state s can only move to a termination state or another fully connected state s' where the number of neurons is smaller ($d' < d$).
- The agent can transition from a convolutional state to any state.
- The agent can transition from a pooling state to any state that is not another pooling state.
- If an agent transitions between convolutional or pooling states, then if the source state s has representation size n, target state s' must have a receptive field size $f' < n$. This condition prevents the convolutional layers from reducing too much the size of the representation.
- The agent can only transition from a convolutional or pooling state to a fully connected state if $n \in \{(8,4],(4,1]\}$, to avoid a very large number of weights.
- The agent can move to the termination state at any time, allowing a variable number of layers.

The reward for the Q-learning algorithm when the agent achieves a terminal state is the prediction accuracy over a validation set. The system performance has been tested over three datasets: CIFAR-100 and CIFAR-10 (with data augmentation), SHVN and MNIST, obtaining a test error rate of 27.14 %, 6.92 %, 2.28 % and 0.44 % respectively, using the best model found. Interestingly, because they find different models during the optimization of CNN architectures, they have decided to build a committee of neural networks with the top-5 performing models. This committee decreases the error rate on SHVN and MNIST to 2.06 % and 0.32 %, though it increases the error in CIFAR-10 to 7.32 % and has not been tested with CIFAR-100.

It is noticeable that MetaQNN does not optimize hyperparameters such as the learning rate or the batch size, does not include recurrent layers as possible states to be included in the architecture, and does not optimize the neurons activation functions. A similar work, including recurrent connections, has been recently published by Zoph and Le [412] from the Google Brain team.

GeNet Also in 2017, Xie and Yuille [393] have worked on a genetic algorithm to evolve the topology of a convolutional neural network to perform visual recognition. Authors considered a constrained case with a limited number of layers, with already predefined building blocks, such as convolution or pooling. Authors recognize the need to perform heuristic search in order to find the optimal topology for their needs, in their own words: *"even under these limitations, the total number of possible network structures grows exponentially with the number of layers. Therefore, it is impractical to*

enumerate all the candidates and find the best one. Instead, we formulate this problem as optimization in a large search space, and apply the genetic algorithm to traversing the space efficiently".

In their work, Xie and Yuille encoded the network structure as a fixed-length binary string. To do so, they divide the network in S stages, where the s-th stage ($s \in \{1, 2, ..., S\}$) contains K_s nodes, denoted as $v_{s,k_s}, k_s \in \{1, 2, ..., K_s\}$. Nodes within each stage are connected and numbered and connections are only allowed from one node to another with a higher number. Each node is translated in the phenotype as a convolutional operator. Before convolution, tensors coming from the different input nodes are aggregated via element-wise summation; and after convolution, batch normalization and ReLU are followed. Convolution operators within the same stage will have the same width, height and number of filters, and stages are separated by pooling operators.

As for the binary representation, each stage is represented with $1 + 2 + ... + (K_s - 1) = \frac{1}{2}K_s\,(K_s - 1)$ bits, where each bit represents the existence or not of a link between two nodes of the stage. Thus, the first bit encodes the existence of a link from $v_{s,1}$ to $v_{s,2}$, the next two bits encode the existence of links from nodes $v_{s,1}$ and $v_{s,2}$ to $v_{s,3}$ respectively; and so on and so forth until the last $K_s - 1$ bits encode the existence of links from $v_{s,1}, v_{s,2}, ...v_{s,K_s-1}$ to v_{s,K_s}.

Figure 3.7 shows an example of the encoding of a network comprising two stages, having four nodes the first stage and five nodes the second. As described before, the binary code represents the links between nodes (convolution operators) within one stage. However, while this approach is innovative because convolutional operators can be performed in a non-linear fashion, it is very restricted because many hyperparameters are predefined prior to the evolutionary process, namely: the number of sequences (S), the number of nodes per each sequence (K_s), the number of filters and their width and height for each sequence, and the size of the pooling operator. As a result, the search space is significantly constrained, and many degrees-of-freedom could be added to the genetic search in the space of possible CNNs.

Also, it is worth noting that GeNet only evolves the structure of the convolutional layers, not evolving the feed-forward or recurrent part of the deep neural network. Moreover, weights are not evolved, but rather learned following a typical backpropagation approach.

Xie and Yuille evaluated the performance of GeNet using the SHVN, CIFAR-10 and CIFAR-100 datasets, obtaining a test error rate of 1.97 %, 7.10 % and 29.03 % respectively. Unfortunately, results obtained by GeNet are not competitive with the state-of-the-art, resulting in a significantly lower recognition accuracy. However, authors state that networks evolved by GeNet are less deep than outperforming convolutional neural networks.

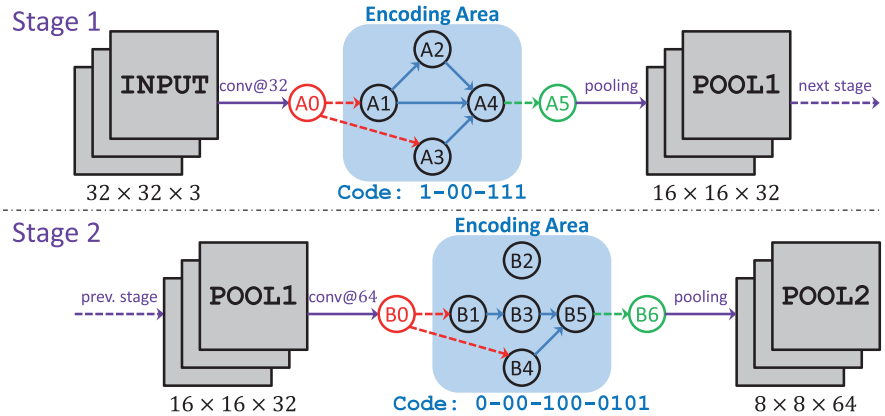


Figure 3.7: *Sample binary genome in GeNET. Source: Xie and Yuille [393].*

Interestingly, authors conclude that "*it is interesting to see that the generated structures, most of which have been less studied before, often perform better than the standard manually designed ones*", thus suggesting that evolution of CNN topologies is a very promising area which is yet to be explored.

CoDeepNEAT In 2017, Miikkulainen et al. [243] have presented CoDeepNEAT, an automated method for evolving deep neural networks. This method is specially interesting since it has been proposed by one of the original researchers involved in the development of NEAT.

When introducing their proposal, Miikkulainen et al. posed a very interesting reflection on this topic which strongly matches the motivation of the current work:

> *"As DNNs have been scaled up and improved, they have become much more complex. A new challenge has therefore emerged: How to configure such systems? Human engineers can optimize a handful of configuration hyperparameters through experimentation, but DNNs have complex topologies and hundreds of hyperparameters. Moreover, such design choices matter; often success depends on finding the right architecture for the problem. Much of the recent work in deep learning has indeed focused on proposing different hand-designed architectures on new problems."*

CoDeepNEAT follows the same working principles than NEAT, yet adapted to work with deep neural networks. In this case, nodes in the chromosome no longer represent neurons, but layers in the DNN. Each node contains a table of real-valued and binary hyperparameters that are subject to mutation. These hyperparameters specify the layer type (convolutional, feed-forward or recurrent) and its properties. Some of these properties are: number of convolutional filters, dropout rate, kernel size, number of neurons, activation function, etc. Also, connections no longer have weights, but just indicate how layers are interconnected. Finally, the chromosome also contains a set of global hyperparameters that do not belong to any specific layer, such as learning rate, momentum, etc. As for the fitness function, the genome is converted into a deep neural network whose weights are learned using a training dataset, and a performance metric is computed.

Because arbitrary connectivity is allowed between layers, then merge layers have to be introduced when one layer has two input layers. Just as in GeNet, this can be done using element-wise summation, though this operation can require downsampling to the minimum layer size.

While the approach described so far is called "DeepNEAT", authors proposed a coevolutionary variant where both modules and blueprints are evolved simultaneously in order to create modular networks, and where each of these modules have a complicated structure among various layers (similar to GeNet's stages). This variant is called "CoDeepNEAT". Also, authors implemented the possibility to include LSTM layers within the network.

Finally, authors tested CoDeepNEAT's performance using the CIFAR-10 domain. After data augmentation, they achieved a test error rate of 7.3 %, which is not the best of the state of the art, though converges much faster than better architectures. It is remarkable that CoDeepNEAT allows learning very complex networks involving convolutional, feed-forward and recurrent or LSTM layers. However, it strongly relies on mutation of these hyperparameters, and does not explore the possibility of using committees of deep neural networks.

EXACT In March 2017, Desell [88] published a work in the pre-print repository arXiv introducing EXACT (evolutionary ecploration of augmenting convolutional topologies), which was later presented in a poster session in GECCO 2017 in Berlin. It is remarkable that most of the work describes how the algorithm is supported by a largely distributed architecture using volunteer

computing. To perform the evolutionary process, the author based his work on NEAT and relied on the fact that the structure of a CNN can be evolved by solely determining the filters sizes and how they are connected. As a consequence, he designed specific mutation and crossover operators.

Regarding mutations, one or several of the following mutation operations is performed at each generation, depending on some user-defined hyperparameters:

- Disabling a random edge in the genome. If this led to unreachability of some output node, then the mutated genome is discarded and a new attempt at mutation is performed.
- Enabling a random edge in the genome.
- Splitting a random edge by creating a new node (just as it was done in NEAT) in the middle, with a filter size which is the mean of the filter sizes of the source and target nodes. Also, a depth value is included which is also the mean of both nodes.
- Adding an edge between two random nodes, given the condition that the edge goes from a node n_i to a node n_o where n_o has larger depth than n_i.
- Changing the filter size in a random node (by increasing or decreasing each dimension in either one or two units).
- Changing the filter size in a random node only in one dimension (similar to the previous one, but acting only in one of the filter dimensions).

As for crossover, edges will be added from the parents to the child with different probabilities depending on whether the parent is the fittest or the less fit of the couple; then, non-selected edges are also added, yet they are disabled in the child. Finally, nodes are added to the child, and when both parents share a node with the same innovation number, then the hyperparameters will be chosen from the fittest parent.

In EXACT, weights are not encoded in the genome, but learned using backpropagation. That enables obtaining complex structures involving many convolutional filters; however, EXACT does

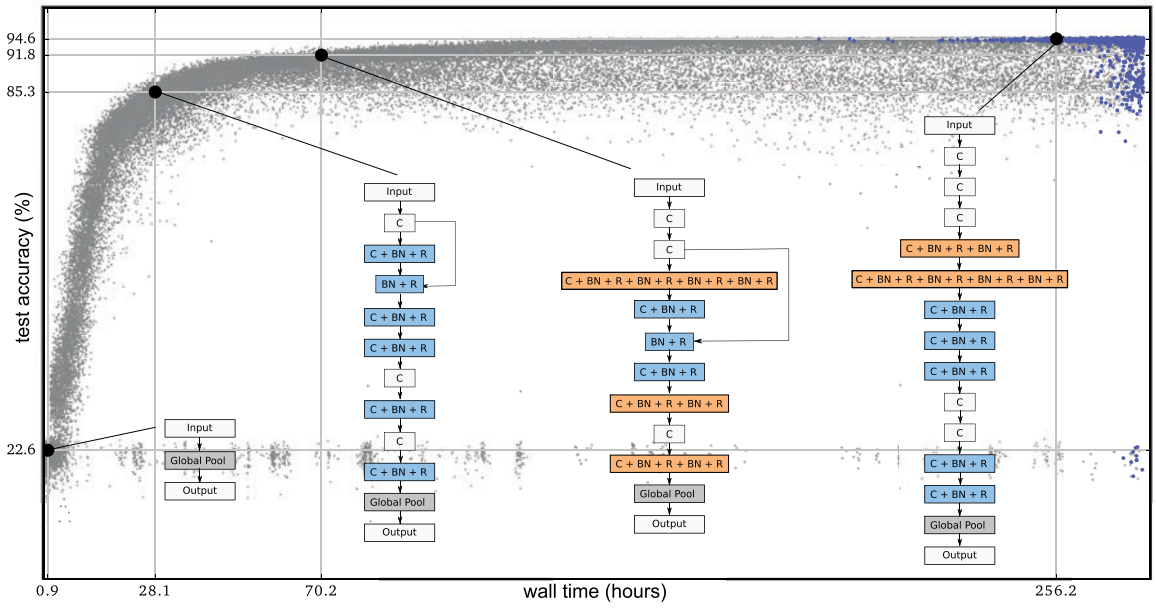


Figure 3.8: *Evolution diagram when evolving CNN topologies using GAs. Source: Real et al. [299].*

not evolve neither pooling operators, activation functions, feed-forward or recurrent layers and other hyperparameters (such as the learning rate). Desell reported an error rate on MNIST of 1.68 %, which unfortunately is significantly worst than most CNN-based approaches, which obtain errors greatly under 1 %.

A similar work has been published by Real et al. [299], from the Google Brain team, although relying mostly in mutation instead of recombination. In this work, authors reported 94.6 % accuracy on CIFAR-10 and 77.0 % on CIFAR-100. Interestingly, authors have also explored building an ensemble to test the models on the CIFAR-10 dataset, improving the accuracy up to 95.5 %. They include an evolution diagram (see figure 3.8) showing how the fitness of the population evolves over time. In that diagram, we can see that very good solutions are found soon during evolution, and then they are slightly improved over the course of generations. This could mean that we could obtain relatively-good, yet not state-of-the-art models after few time. More recently, in 2018, Real et al. [298] have introduced the concept of a regularized evolutionary algorithm, where the oldest model is removed from the population in every generation. When compared against reinforcement learning and random search, authors conclude that neuroevolution and reinforcement learning perform similarly well, although neuroevolution is faster, and both considerably surpass random search. It is worth mentioning that authors report running large-scale experiments in 450 GPUs over a week, and dedicated evolution experiments in 900 TPUv2 during five days.

DEvol Joe Davison, from Microsoft, has recently created an opensource project called DEvol [83], for automated deep neural network design via genetic programming. We have not been able to find a peer-reviewed publication describing this project.

The genome connects several nodes sequentially, each node representing a layer. Hyperparameters for each layer are also evolved, including the number of filters, the dropout rate, the activation function, etc. From the code documentation, it can be inferred that DEvol supports a variable number of convolutional and dense layers. When tested over the MNIST dataset, they have achieved a test error rate of 0.6 %, which is fairly good yet not state-of-the-art.

Also, Suganuma et al. [351] have recently published a work using Cartesian genetic programming to optimize CNN architectures. This paper was presented in GECCO 2017 in Berlin as a best paper nominee. However, in this case their approach only evolves convolutional layers, not observing neither feed-forward or recurrent layers, nor optimization hyperparameters. Authors have reported a test error rate on the CIFAR-10 database of 5.98 % with data augmentation.

CEA-CNN By the end of 2017's summer, Bochinski et al. [31] published a work describing a system based on evolutionary computation to optimize the hyperparameters of convolutional neural networks. In the abstract of their paper, authors already stated one of the key arguments that support the use of neuroevolution of CNNs, as we have insisted throughout this chapter: "*it is not trivial to find the best performing network structure for a specific application because it is often unclear how the network structure relates to the network accuracy*". They also declare that network structures are often chosen by an "educated guess", placing the need for automatically determining the topology.

In their proposal, authors evolve a good number of hyperparameters, including the number of kernels and the kernel size of convolutional layers and the number of neurons in fully connected layers. Pooling is not included as an option, and other training hyperparameters are defined by the authors. An interesting design decision made by the authors involve sorting the evolved layers by descending complexity; i.e., layers are first evolved and then sorted to form the network, so that first layers will be those with more kernels or neurons. As the evolutionary algorithm, authors propose a $(\mu + \lambda)$ evolution strategy.

Besides, authors also suggest how to extend this framework to allow for the joint optimization of committees of convolutional neural networks, by using a fitness function that takes the global classification error of the population.

Finally, authors reported a test error rate in the MNIST database without data augmentation of 0.34 % using an individual model and 0.24 % using a committee of 34 CNNs, this last result being extremely competitive among the state of the art.

EvoCNN In October 2017, Sun et al. [352] published a preprint in arXiv describing the use of a genetic algorithm for evolving deep convolutional neural networks. A remarkable feature of this proposal is that the encoding supports variable-length chromosomes, therefore providing a natural encoding for different numbers of convolutional and fully connected layers.

The hyperparameters evolved by EvoCNN include the filter width and height, the number of filters, the stride width and height and the convolution type for convolutional layers, the kernel width and height, the stride width and height and the pooling type for pooling layers, and the number of neurons for fully connected layers. Besides, weights are also evolved, but instead of evolving all weights, the mean value and standard deviation are evolved instead for each layer, and weights are later randomly assigned following a Gaussian distribution.

Because chromosomes can be of different lengths, a specific crossover operator was implemented in order to support recombination of chromosomes, which involves aligning parent chromosomes of different lengths separating the list of convolutional layers, pooling layers and fully connected layers. Also, a specific environmental selection operator is introduced to promote diversity in the population, which is combined with elitism to help improve the fitness throughout the generations. Also, authors suggest using an efficient fitness computation which reduces the amount of training epochs for each individual, using the mean classification error and standard deviation over the different validation batchs in the last epoch as the fitness value. Standard deviation is only used in case of a tie when sorting individuals by mean error.

Authors apply EvoCNN to a variety of image recognition domains, including many MNIST variants described by Larochelle et al. [195], which due to their nature are not comparable with the MNIST database itself. Authors report results very competitive with the state of the art.

Hierarchical representations Recently in 2018, Liu et al. [223], from Carnegie Mellon University and DeepMind have proposed representing neural networks by means of the computation graph, with a single input and a single output. Therefore, the architecture can be defined as a tuple $(G, \mathbf{o})$, where $\mathbf{o} = \{o_1, o_2, ...\}$ is the set of available operations and G is the graph identified by its adjacency matrix, where $G_{ij} = k$ means that the k-th operation (o_k) will be executed in between nodes i and j.

Then, Liu et al. suggest using a genetic algorithm for evolving individuals using mutation, which allows adding, removing or editing edges in the graph. The set of operations proposed in this work allows the creation of convolutional and pooling layers, yet not feed-forward or recurrent.

IPPSO Another approach to the evolution of convolutional neural networks has been proposed by Wang et al. in 2018 [379]. However, this work is remarkably different from the previous ones since it relies on particle swarm optimization.

In this case, authors propose an encoding that is inspired in CIDR notation for specifying subnets and allows the evolution of convolutional layers (number of kernels, kernel size and stride size), pooling layers (pooling size, stride size and type) and fully connected layers (number of

neurons). Also, they observe the concept of disabled layer in order to support a variable network length. Other aspects such as activation functions, recurrent layers or learning hyperparameters are not observed in the proposal.

Fitness computation is reduced to 10 epochs in order to accelerate the process, and the proposal is tested against MNIST variants described by Larochelle et al. [195], which are not comparable to MNIST dataset. In the three datasets that are common to the evaluation of EvoCNN [352], IPPSO attains slightly better performance in two of them (MNIST Basic and MNIST with Rotated Digits plus Background Images) and significantly worse performance in the third (Convex Sets).

Lamarckian evolution Finally, in late June 2018 a new preprint has been published by Prellberg and Kramer [287], later accepted for presentation in PPSN 2018, which will take place in September. This work is closely related to one previously published by Kramer [189]. In both cases, a (1+1)–EA (evolutionary algorithm) is used to evolve the convolutional layers of a CNN. In the Kramer work, fully connected layers and activation functions are also evolved, and gradient descent with the Adam algorithm is used to learn the weights. The most recent work simplifies the previous one by fixing dense layers at the end of the network as well as activation functions.

In their approach, the evolutionary algorithm relies only on a mutation operator, which is intended to improve one individual one generation at a time. The mutation can add a new building block (convolutional layer) with a random number of filters, a random kernel size and a stride of one, or modify some of the hyperparameters of such building block by adding or removing filters, changing the filter size or the stride. Mutation can also delete a building block.

In this work, a mechanism is introduced to support weight inheritance, so that once a network is trained, its child can reuse weights, except for the new layers and filters resulting from mutation, where parameters are randomly initialized using Glorot initialization [111].

In both works, a niching scheme and mutation rate control is used for supporting the evolutionary procedure. Kramer [189] reported a maximum MNIST accuracy of 99.1 %, and the work by Prellberg and Kramer [287] attained an accuracy of 89.3 % in CIFAR10 and 66.1 % in CIFAR100 without data augmentation. These results are not state-of-the-art, but according to authors they are not intended to be, since they focus on showing the advantages of weight inheritance.

Also in June 2018, Prellberg and Kramer [288] presented an approach where a GPU-optimized evolutionary algorithm evolved the weights (a total of 92,000 parameters) of a CNN, despite they only obtaining a 98 % accuracy on the MNIST dataset.

Non-evolutionary solutions While neuroevolution has proved its value when optimizing the topology of deep and convolutional neural networks, the process is still computationally expensive. Prior to 2016, Jozefowicz et al. [168] performed random search for evaluating recurrent neural networks using LSTM cells. From 2016 onwards, a number of works have arisen presenting different approaches to topology optimization, also known as neural architecture search.

For example, Li et al. [211] presented Hyperband as an approach to hyperparameter optimization modelling the problem as a multi-armed bandit problem. However, in this approach the number of layers remains fixed, although some training hyperparameters (batch size or learning rate) are included in the search space.

Saxena and Verbeek [317] proposed the use of so-called "convolutional neural fabrics", which are a mechanism of generation of convolutional neural networks. They report a test error rate of 0.33 % in MNIST and 7.43 % in CIFAR10, using data augmentation in both cases, although requiring a substantially large number of parameters.

Mendoza et al. [238] presented Auto-Net, which relies on the concept of "automatic machine learning" (AutoML), for optimizing hyperparameters of deep neural networks, including aspects from the training hyperparameters. However, this solution does not observe convolutional or recurrent layers, working only for feed-forward fully connected networks. Still, authors state that their work led to the first fully-automatically-tuned neural network to beat the human expert track of the ChaLearn AutoML Challenge [135].

Albelwi and Mahmood [2] have also proposed to optimize the architecture of CNNs by introducing a new objective function that combines the quality metric (e.g., error rate) and information learnt by feature maps obtained from deconvolution networks, and using the Nelder-Mead method, better known as "simplex" [264], to optimize such objective function.

Jaderberg et al. [163] from DeepMind have suggested a population-based training of both models and topology, using random hyperparameter search [27], though they focus on deep reinforcement learning and generative adversarial networks.

Negrinho and Gordon [263] presented DeepArchitect, which consists of an extensible and modular language to represent search spaces over architectures and hyperparameters, which can be expressed in the form of trees, therefore enabling optimization using a variety of search algorithms, such as Monte Carlo tree search or sequential model-based optimization.

Hazan et al. [144] have proposed a spectral approach to hyperparameter optimization, assuming that the function mapping the hyperparameters to the error metric can be approximated by a sparse and low degree polynomial in the Fourier basis, and then approximating it with a decision tree.

Kandasamy et al. [171] have proposed NASBOT, which relies on Bayesian optimization to perform neural architecture search, using distance metric in the space of neural network architectures. This distance, named OTMANN, tries to match the computation at the layers of one network to the layers of the other, and can be computed efficiently using a optimal transport program.

In recent years, many authors have put the focus on efficiency. For example, Cai et al. [48] proposed EAS (standing for "efficient architecture search"), whose more remarkable contribution is that allows the modification of a previous architecture by performing network transformation operations, relying on function-preserving transformations [60] to initialize the weights of the new network so that it performs the same function that the previous one. This results in a very efficient approach, since slight mutations can be performed without requiring to re-initialize the network from scratch. The whole optimization process is orchestrated using reinforcement learning. Authors report a test error rate on CIFAR10 of 4.66 %, a competitive result within the state of the art. A different approach is taken by Brock et al. [45] with SMASH, where they describe a method for generating weights automatically conditioned by the network architecture (based on HyperNetworks [136]). Then, since the learning procedure is cheaper (by avoiding multiple steps of backpropagation), the process of searching through several architectures turns faster, and authors suggest using gradient descent for optimizing the network architecture itself.

Additionally, Wistuba [391] from IBM Research proposed the use of Monte Carlo planning for optimizing CNNs hyperparameters within one day of GPU computing, taking advantage of Net2Net [60] to gain speed, attaining a test error rate in MNIST of 0.31 %, and in CIFAR10 of 6.45 %. Elsken et al. [96] have decreased this time to 12 hours with a single GPU, using hill climbing and short optimization runs by cosine annealing, attaining a test error rate in CIFAR10 of 5.7 %, which decreased down to 5.2 % in one day and 4.4 % using 4 GPUs. Zoph et al. [413] from the Google Brain team have designed a novel search space, namely the "NASNet search space" that enables transferability from architectures optimized on small datasets to larger datasets, basing the optimization process on their previous work using reinforcement learning [412], and reporting a test error rate on CIFAR10 of 2.4 %, which is state-of-the-art. Liu et al. [222] have reported up to 5 times more efficiency than Zoph et al. using progressive neural architecture search, which relies

on sequential model-based optimization for searching for architectures of increasing complexity. A similar work has been published by Pham et al. [282], proposing ENAS (efficient neural architecture search), where policy gradient is used to select a subgraph from a large computation graph that maximizes the expected reward in a validation set, and according to authors this procedure can be up to 1000 times faster than standard neural architecture search. Kamath et al. [170] have introduced EnvelopeNets, which are founded on the idea that statistics obtained from the feature maps during training can be used to compare the utility of filters within a network, therefore reducing the number of evaluations required for reaching convergence.

Finally, it is worth mentioning the work of Bello et al. [22], which focuses on optimizing a learning function instead of the network architecture. To do so, they use reinforcement learning to train a controller that generates a string that describes a mathematical update equation based on a list of primitive functions. As a result, authors propose two novel optimizers, namely PowerSign and AddSign, which outperform Adam, RMSProp or SGD when learning a CNN for CIFAR10.

Industrial solutions Apparently, some cloud vendors offering machine learning services are also providing some options to automatically tune some of the models hyperparameters. It is the case of Google Cloud Machine Learning Engine, which sits on top of the TensorFlow framework for deep learning. According to their presentation website [124], they offer a feature called "HyperTune" able to automatically tune the hyperparameters of machine learning models. However, acording to the HyperTune documentation [123] it seems that, while there is a high flexibility in the hyperparameters to optimize (the user can define as many as wanted and of diverse types), the tuning is performed by a full-scan search of all combinations. While this is not explicitly specified in the documentation, it is inferred from the next description: "*Hyperparameter tuning works by running multiple trials in a single training job. Each trial is a complete execution of your training application with values for your chosen hyperparameters set within limits you specify. The Cloud ML Engine training service keeps track of the results of each trial and makes adjustments for subsequent trials. [...] Every hyperparameter that you choose to tune has the potential to exponentially increase the number of trials required for a successful tuning job. When you train on Cloud ML Engine you are charged for the duration of the job, so careless assignment of hyperparameters to tune can greatly increase the cost of training your model*". As a result, it is more an approach to automate a tedious trial-and-error process than to perform a guided, efficient search; yet offering high versatility to optimize many aspects of the training process.

Meanwhile, BigML has presented OptiML [5], which uses Bayesian hyperparameter optimization to test different models (not necessarily deep neural networks, but any kind of parameterizable machine learning model) and select the optimal. Google has also presented Cloud AutoML [122], which is currently (as of June 2018) in an Alpha release. Although details about how AutoML works are not provided by Google, a blog post by Le and Zoph [199] cites neuroevolution as one way of doing so. The term "AutoML" transcends Google's product and is used to refer to techniques or solutions that are able to apply some learning technique to improve a machine learning pipeline or task, a field that encompasses, among others, neural architecture search.

AutoML has also been named "Machine Learning for Machine Learning", or "ML4ML", and some applications have been publicly presented in conferences or seminars, such as in 2018's edition of T3chFest, which took place in Universidad Carlos III de Madrid [355]. Amazon Web Services has also very recently (as of June 2018) announced another application of ML4ML in its cloud product SageMaker [159], enabling hyperparameter tuning in deep neural networks. According to Forbes [164], AutoML is a key aspect of the future of artificial intelligence.

Finally, the term neuroevolution has been explicitly included in some press releases of relevant media and companies in late 2017, such as O'Reilly [342], Uber Engineering [343] or KDnuggets [369], the latter one including the source code for a simple Torch-based implementation genetic programming. This media coverage shows the relevance of neuroevolution.

While in this section we have reviewed the few works we could find about automatic design of deep and convolutional neural networks, it can be expected that new works be published during the development of this thesis and soon after its publication.

3.6 Summary

In this chapter we have reviewed the state of the art regarding techniques for automatic design of neural networks. A quite extensive subset of these techniques involves "*neuroevolution*", which refers to the application of evolutionary algorithms in order to evolve some aspect of neural networks, either the topology, the parameters, learning rules, or several of them at a time.

The first works on neuroevolution arose in 1989 and, since then, neuroevolution has been used successfully for almost three decades. Because this field is really extensive, in this chapter we have described some basic concepts and the most relevant neuroevolution techniques.

Only in the latest years some applications of neuroevolution have appeared in order to evolve deep and convolutional neural networks. Since this field remains largely unexplored, we have done our best to thoroughly survey all the works done so far in this research area. Despite being a very new field, it is gaining momentum in the last months; a proof of this fact is that few days before the deposit of this dissertation (in June 16, 2018), a Springer book by Hitoshi Iba has been published online entitled "Evolutionary Approach to Machine Learning and Deep Neural Networks: Neuro-Evolution and Gene Regulatory Networks", which contains a chapter whose title is "Evolutionary Approach to Deep Learning" [160].

This thesis makes a contribution in the area, as its objectives involve the application of evolutionary computation to the automatic design of deep convolutional neural networks. To the best of our knowledge, our proposal differs in its approach from previous works. In particular, some works have performed an extensive and very flexible evolution of convolutional operators, but fixing the feed-forward layers, the activation functions, and the optimization hyperparameters. Other works perform a evolution of the network topology yet significantly reducing the search space by defining a fixed number of layers or other hyperparameters a priori. Most works do not observe the possibility of including recurrent layers, or combining different neural network models within a committee or ensemble. Finally, some approaches are quite complete but do not use evolutionary computation for the optimization process, but reinforcement learning or classical search.

Table 3.1 shows a brief comparison of the system proposed in this thesis against closely related works. The abbreviations shown in the table header stand for the next criteria:

- **Var. Ly.**: whether the proposal supports a variable number of layers (either convolutional, feed-forward, recurrent, etc).
- **Conv.**: whether the proposal evolves convolutional layers or some of their hyperparameters.
- **FC**: whether the proposal evolves fully connected layers or some of their hyperparameters.
- **Rec.**: whether the proposal observes the inclusion of recurrent layers or LSTM cells.
- **Act. Fn.**: whether the proposal evolves the activation function instead of hardcoding it.
- **Opt. HP**: whether the proposal supports the evolution of optimization hyperparameters (learning rate, momentum, batch size, etc).
- **Ens.**: whether the proposal supports the construction of an ensemble of neural networks.
- **W**: whether the proposal evolves the weights of the network.

Work	Technique	Var. Ly.	Conv.	FC	Rec.	Act. Fn.	Opt. HP	Ens.	W
Koutník et al. [186]	CoSyNE								•
David and Greental [82]	GA								•
Verbancsics and Harguess [373]	GA (NEAT)		•		•				
MENNDL [404]	GA		•						
Loshchilov and Hutter [225]	CMA-ES		•	•			•		
Fernando et al. [100]	DPPN	•	•	•					•
Tirumala et al. [363]	Co-Ev GA								•
MetaQNN [10] ∗	RL	•	•	•				•	
Zoph and Le [412] ∗	RL	•	•	•	•	•			
GeNet [393]	GA	•	•						
CoDeepNEAT [243]	GA (NEAT)	•	•	•	•	•	•		
EXACT [88]	GA (NEAT)	•	•						
Real et al. [298,299]	GA (NEAT)	•	•					•	
DEvol [83]	GP	•	•	•		•			
Suganuma et al. [351]	CGP	•	•						
CEA-CNN [31]	ES	•	•	•				•	
EvoCNN [352]	GA	•	•	•					•
Young et al. [403]	GA	•	•	•		•			
Liu et al. [223]	GA	•	•						
IPPSO [379]	PSO	•	•	•					
Kramer [189]	(1+1)–EA	•	•	•		•			
Prellberg and Kramer [287]	(1+1)–EA	•	•						
This thesis	GA/GE	•	•	•	•	•	•	•	

Table 3.1: *Brief comparison of this thesis's features with related works.*

In this table, a star (∗) is depicted next to two of the works in order to point out that these two works have been included due its relevance, but are not using evolutionary computation techniques in order to search for optimal CNN topologies.

It is important to note that table 3.1 is only summarizing the details of each technique. For example, works are considered to evolve convolutional layers when *any* aspect of convolutional layers is evolved. However, there are some works that enable very complex graphs of convolutional operators, whereas others are only evolving few hyperparameters and fixing all of the others, thus resulting in very constrained approaches. The reader is referred to the previous section in order to better understand the scope and features of each of the works included in this table.

As for this thesis, it can be seen how it is one of the most complete works as it covers most of the comparison criteria. The only exception is the evolution of weights, which are instead learned using backpropagation-based optimizers. The reason for this decision is that effective and efficient optimizers have appeared in the last years in order to learn the parameters of deep convolutional neural networks, which address many of the limitations of classical backpropagation. Also, it can be seen how very few related works evolve the network weights, and those approaches are very limited or deal with very small networks in order to reduce as much as possible the search space. If flexible topologies are allowed comprising several convolutional layers (with hundreds of filters each), and several feed-forward or recurrent layers with thousands of units, then the evolution of all the parameters involved (which can be in the order of millions) is extremely computationally expensive, and probably ineffective given the existence of better solutions.

Chapter 4

Proposal

As we have introduced before, when proposing deep learning solutions to tackle classification problems, their behavior is very sensitive to the design of the topology. In particular, when the solution involves convolutional neural networks, there are a variety of hyperparameters that can be specified, and some of these are critical hyperparameters whose setup can have a significant impact in the network performance. To put it simply, for a certain problem there may be a CNN architecture that provides very accurate results, however, if the kernel size of one pooling layer is increased, even a little, the whole architecture may become useless.

Those hyperparameters regarding the setup of convolutional layers are often critical. Previously in this document we described convolutional layers as feature extractors that transform the input data to generate an output of local features. These features will be low-level in the first convolutional layer and become more and more abstract as the data passes through subsequent layers. Because CNNs can be used to classify a very diverse set of data types (including, but not constrained to images, sound waves, biological or electrical signals, text, etc.), the optimal design of convolutional layers will be very sensitive to the domain of the data fed to the network. This not only applies to the number of convolutional layers, but also to the kernel size or pooling layers.

Nevertheless, not only hyperparameters affecting the convolutional setup are important. Hyperparameterss such as the learning rate or the batch size can affect how the model weights evolve over time, thus impacting the performance.

There is not a rule of thumb for designing CNN topologies, and handcrafting them is mostly a matter of trial-and-error. Thus, coming up with a successful, or at least acceptable solution, can take a lot of time. Also, even if a good topology be found, it is hard to tell whether a different architecture could provide better performance.

4.1 Formal Definition

In this thesis, we propose the development of a system which is able to automatically design convolutional neural networks. More specifically, we expect program p to, given a training set T and a validation set V, output a set of hyperparameters A defining a valid CNN topology, so that when this topology is trained with T, classification performance over V in terms of a given metric m is optimal. Equation 4.1 provides a formal definition of what this system is expected to do:

$$p : T, V \rightarrow A, \text{ so that } \underset{A}{\operatorname{argmin}} \left(m \left(A(T), V \right) \right) \tag{4.1}$$

The program p must remain mostly agnostic to the domain and specifics of the input data, namely T and V. Each of these datasets should be structured in a common machine learning format, which comprises a tensor of features $X_{i \times d_1 \times \cdots \times d_n}$, and a vector of labels y_c. Tensor X will have $n+1$ dimensions, where n is the number of dimensions of each instance.

4.2 System Features

A more specific explanation on how the tensor dimensions are interpreted would require domain knowledge. For example, a 2-dimensional instance could represent a grayscale picture or a time series comprising several channels, a 3-dimensional instance could represent a RGB image or a grayscale video, etc. Program p should only be aware of the dimensionality of tensor X and, in order to look for suitable CNN configurations, a description of which of these dimensions present data locality. This knowledge is important as convolutional kernels are only applied over those dimensions presenting locality. For example, if we had a 2D picture, then 2D kernels should be applied, but if we had a time series with several channels, then we should apply 1D kernels, even if tensors in the two problems have the same dimensionality.

This system could be able to provide a CNN topology which outperforms the best one designed by a human for a certain problem. Even if it does not, it could still save time when designing deep-learning solutions, providing a competitive classifier in short periods of time.

Also, this system provides an additional advantage: in the case data changes over time, the system could adapt the architecture gradually, without requiring a manual redesign. Of course, this could happen if the domain evolves in a way where the characteristics of the input data change significantly. This system would prevent the performance to degrade over time, requiring an architect or engineer to come up with a new topology. If this degradation is fast, once a successful redesign is achieved manually, the previous topology could be providing a non-acceptable performance.

4.3 Challenges

Program p has to solve an optimization problem. In computer science, there is a high variety of computational techniques able to solve this kind of problems; however, some can be more convenient than others to solve the problem at hand.

The problem of optimizing the architecture of a convolutional neural network poses several challenges that must be addressed with the choice of a suitable optimization algorithm:

1. The model learnt given an architecture A and a training set T is stochastic, as weights are usually initialized in a random fashion. Because metric m depends on this model, the performance of a certain architecture will vary.
2. Very often, a small change in the architecture A will have a very small effect on its performance. However, if a critical hyperparameter is modified, then the performance can be severely affected, even leading to a useless model.
3. The existence of a sole optimal solution is unlikely. It will often happen that several solutions will exist, and some of them can be very different.

These challenges will have some implications which should be considered before deciding the most appropriate algorithm.

Regarding challenge (1), we can now define the concept of dominated and non-dominated solutions. An architecture A_1 would be dominated by an architecture A_2 if, upon an infinite number of experiments with a fixed training set T and validation set V, $m(A_2(T), V)$ is always higher than $m(A_1(T), V)$. In this case, we can clearly state that A_2 is better than A_1 for the particular problem involving datasets T and V.

In a different case, sometimes the performance of A_1 will be better ($m(A_1(T), V) > m(A_2(T), V)$) and other times A_2 will outperform A_1 ($m(A_2(T), V) > m(A_1(T), V)$). In this case, we can say that neither architecture is dominated by the other.

However, we must admit that even if we had two different topologies, and neither of them were dominated by the other, it can still happen that one of them be considered better. In this case, we would not be considering their performance based only on the values of m given one execution of the CNNs defined by the topologies, but instead would run each CNN enough times as to compare both distributions of m values and test whether one is significantly better than the other.

Challenge (2) implies a very irregular landscape of function m given the architecture A. As a result, a good topology may be surrounded with poor or average architectures, and yet another good topology could be found far in the solutions space. Precisely, functions with complicated contours are the most challenging to optimize.

Finally, challenge (3) has a lot to do with the concepts we just described. The existence of a sole optimal solution A^* would require that no other solution A exists which dominates A^*, and this is a very rare scenario. Even if we relax the domination constraint, it can be difficult that a single solution exist that is better (in average) than all other topologies. Also, it can happen that very good solutions are found far apart in the space of solutions.

More interestingly, it could happen that a system aggregating the decisions of different CNNs provide even better results than those provided by any individual network. This concept is known as ensemble or committee of machines [191], and have been proved to outperform individual classifiers for many problems. When dealing with committees, we are interested in having solutions as diverse as possible. This seems a feasible condition given what we stated regarding challenge (2).

4.4 *Analysis of the Problem*

Given the prior challenges and implications, we must now consider which algorithms are more suitable for solving this problem.

The field of artificial intelligence is rich in search and optimization algorithms and techniques. Many of these algorithms involve heuristic search; i.e., in order to work properly they require the definition of a heuristic function that somehow expresses how far we are from the optimal solution. However, the definition of this heuristic is non-trivial and even turns out to be inconsistent with challenge (2) described above: we can know how good a solution is, but there is little information about which is the best path leading to better solutions.

Once heuristic methods are discarded, we can still considered uninformed search (or *blind* search), where a solution is searched through enumeration of all possible solutions. One problem with these techniques is that the search space can be very high (potentially infinite), and thus enumeration is not a suitable option given the combinatorial explosion of possible CNN topologies. More importantly, there is an additional issue: these search algorithms often stop when a solution is found; however, in this problem there are as many solutions as valid CNN topologies. The goal is to find a good solution, but it can be difficult to define what a *good* solution is ahead of starting

the search. Then, unless we can state a value for metric m that we consider acceptable, uninformed search is not a suitable approach to solve this problem.

Finally, the last type of search algorithms are those based on metaheuristics. These algorithms perform an informed search, just like heuristic search; however, unlike that type, they do not require an explicit heuristic function. Instead, they only need information about how good a certain solution is, and we can provide that information given the existence of metric m. Also, metaheuristics are approximate search methods, this meaning that they do not guarantee finding an optimal solution. Nevertheless, this should not constitute a problem, given that in challenge (3) we stated that the existence of one single optimal solution that dominates every other possible solution is extremely unlikely, especially since the value of metric m for a solution is stochastic.

A important set of techniques based on metaheuristics are those involving evolutionary computation. Evolutionary computation is a biologically-inspired field of knowledge, and its techniques, often referred to as evolutionary algorithms, resemble the evolutionary processes based on Darwin's theory of evolution and natural selection.

As we have discussed in the previous chapter, neuroevolution is the name given to a field of study that applies evolutionary computation to the design of some aspect of neural networks, either the optimization of their weights, their topology, their learning rule, etc. Neuroevolution has been used for over three decades (since as back as 1989, only three years after the rise of neural networks), and has been proven successful for finding suitable models in many diverse applications.

Since the late 1980s and the early 1990s, researchers noticed that evolutionary computation was convenient for optimizing neural networks. As we already stated in the previous chapter, both Yao [400] and Miller et al. [245] agree in that evolutionary computation is suitable for finding optimal solutions due to the next properties of the search space:

- It is potentially infinite, since the number of possible nodes and connections is unbounded.
- It is non-differentiable, as changes in the number of nodes or connections are discrete but can have a continuous effect on the network performance.
- It is complex and noisy, because the mapping between a network and its performance is indirect and stochastic due to the randomness of initial weights.
- It is deceptive, since similar network architectures can lead to very different performances.
- It is multimodal, since very different architectures can have similar performance.

For most of these three decades, neuroevolution has focused on evolving very simple ANN architectures, often involving only one hidden layer. It was not until 2014 that a research work was published exploring the evolution of convolutional neural networks. This new field of study has led to very sparse works during 2015 and 2016, having a significant growth in 2017 ans 2018. Due to its recency, there are still very few works in this area, and as we concluded in chapter 3, they have significant room for improvement.

Additionally, evolutionary computation techniques have an additional advantage: they involve a population of individuals. This means that, at any given point of time in the evolutionary process, we can find not only one solution but a set of solutions. This is very interesting since it is consistent with challenge (3), enabling us to check different good solutions and even making it possible to build an ensemble out of them.

In this thesis, we have decided to focus on the use of evolutionary computation. To make this decision, we have previously concluded that when several search algorithms are compared, uninformed search and heuristic search are not suitable because of the problem characteristics.

Metaheuristics; however, seem appropriate for tackling our problem, and the choice of evolutionary computation is twofold: first, neuroevolution has been successfully used for almost three decades but its application to convolutional neural networks remains mostly unexplored, and second, the fact that evolutionary algorithms involve a population of individuals is consistent with our desire to find more than one valid CNN topology.

4.5 Design of the Solution

Once we have decided to use evolutionary computation, we need to design the solutions based on these techniques. In this section, we explain all the design decisions, from the hyperparameters of the topology which must be optimized, to the specific evolutionary algorithms that we will be using and some specific details of their implementation.

4.5.1 Aspects of Optimization

As we have seen in chapter 2, the design of a CNN-based neural system is complex, and involves a large number of hyperparameters that can determine the effectiveness of the network to solve a given task. In this work we try to facilitate this task, elaborating a procedure capable of automatically generating a complete design of convolutional neural networks. This implies the ability of evolving many aspects of the architecture: number of layers, connectivity, etc., as well as network operational hyperparameters: activation functions, learning rates, etc.

However, the number of hyperparameters to consider is too high to be efficiently covered. The only way to do an effective search is to make some simplifications that do not reduce the chances of finding good solutions. Following this idea, we have organized the hyperparameters into four groups: input setup, convolutional architecture, dense architecture and learning hyperparameters.

Regarding the input setup, the key hyperparameter to be optimized is the batch size: modifying the size of training minibatches can alter the convergence behavior of the process. Morever, in some cases we will have to make some decisions on the sampling: this will happen when the input data is not naturally divided into samples, but is rather presented as a continuum (like in the case of signals). In this case, we will have to segment the signal into samples, thus having to decide the sample (or window) size.

The convolutional layers must allow us to create any type of architecture following the rules generally established in their design, i.e. sequential layers with neurons of a layer partially connected to a spatially clustered group of the contiguous layer. Each convolutional layer receives values from the preceding layer, or from the input if it is the first layer. The values are grouped by one or more kernels, and then, a pooling process can be performed.

The most relevant hyperparameters to determine the architecture of the convolutional stages are the number of layers, the number of kernels (also known as filters or patches), their size and activation function in each layer, and the pooling size in case pooling is performed after certain convolutional layers. Regarding the activation function, we only consider two options: linear and ReLU, since there is not a significant impact on the use of different non-linear functions, and the computation of ReLU is more efficient when compared to other alternatives. Specific convolutional setups (e.g. padding, dilation, or special strides) will not be included, since they have a small impact in overall performance and significantly increase the space of valid CNN topologies. The same happens with the pooling aggregation function, and in this case max-pooling will be used, as it is the most common approach used in the literature.

Dense levels are less complex. In general, a feed-forward fully connected network can be used along with a backpropagation mechanism for learning the network weights. In this thesis we have considered convenient to also support the use of recurrent architectures, therefore, more powerful and effective models for classification can be generated.

Thus, we will not only parameterize the number of layers and neurons in each layer, which are two critical hyperparameters, but also the connectivity pattern in each layer: feed-forward, recurrent, LSTM or GRU. This will enable the creation of hybrid networks, where each layer may have a different structure, allowing greater richness and complexity in the alternatives considered, and the creation of new models especially suitable for certain problems, if needed. The activation function is also a hyperparameter to be optimized, and as in the case of convolutional layers, it can be a linear function or ReLU. Additionally, we have included the possibility of L1 and/or L2 and dropout regularization after each layer, as it is a very common and profitable mechanism to avoid overfitting in CNNs. If dropout were used, then a fixed rate of 50 % of the units would be dropped, and we have not found a need to modify this rate, since it has little effect.

Finally some general hyperparameters, not directly related to the topology but rather to the learning process, will be optimized as well. In this case, we will focus on the learning rule and the learning rate, which will can significantly affect the way in which weights are learned.

In summary, in our proposal we have considered the following hyperparameters:

- Input setup
  - Batch size
  - Sampling setup (when required)
- Convolutional layers
  - Number of layers
  - Number of kernels in each layer
  - Kernel size in each layer
  - Activation function in each layer
  - Pooling size (if any) after each layer
- Learning hyperparameters
  - Learning rule
  - Learning rate
- Dense layers
  - Number of layers
  - Connectivity pattern of each layer
  - Number of neurons in each layer
  - Activation function in each layer
  - Weights regularization in each layer
  - Dropout (none or 50 %) in each layer

Once the main hyperparameters for CNN design have been identified, an efficient search procedure for these hyperparameters is required.

4.5.2 *Basic Concepts of Evolutionary Computation*

The chosen optimization techniques belong to the field of evolutionary computation, which is a subset of the so-called biologically-inspired artificial intelligence. Evolutionary computation techniques resemble natural selection and biological evolution as described in the Darwinian theory. To put it simply, a population of several individual exists. An individual is encoded via a chromosome, also known as genotype in evolutionary computation, which is a sequence of genes. The individual's phenotype is its "real" interpretation, in this case, the phenotype would be the CNN topology. Individuals have a fitness, and fittest individuals will have a higher chance to reproduce and generate offspring that will go through to the next generation. As in Darwinian theory of evolution, in evolutionary computation techniques the survival of the fittest is present, and a couple of good individuals will often reproduce leading to even fitter offspring. Across generations, it is expected that the population fitness will improve, eventually obtaining remarkably good individuals.

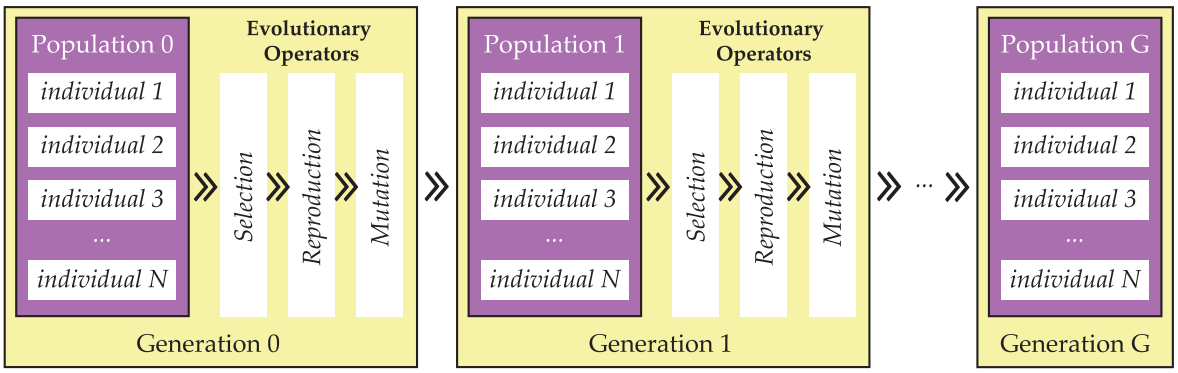


Figure 4.1: *Simplified workflow of an evolutionary algorithm.*

Figure 4.1 shows a general schema of how evolutionary computation techniques work. Evolutionary operators are those responsible for building a new population from the previous one, and most commonly involves selection, reproduction and mutation:

- Selection resembles natural selection, and is a mechanism to choose some of the fittest individuals. While there are several selection strategies, a tournament is one of the most commonly used. In a tournament, τ individuals are randomly chosen and brought into an arena. They will fight between them and the fittest individual will win the tournament. Selection is done twice to choose another individual, and then they will reproduce, generating offspring. This whole process is repeated until there are enough offspring as to fill the next population.
- Reproduction is a mechanism for individuals to generate offspring, keeping the genetic material of parents. Because there is a mapping between phenotype and genotype, offspring will share some similarities with their parents. A common way of reproduction is crossover, where parents' genotypes are combined in some manner to generate new genotypes.
- Mutation is an operator to randomly modify the genetic material of an individual. In nature, mutations happen with some probability, and it serves for introducing new material not previously found on ancestors. In evolutionary computation, mutations allow a broad exploration of the search space, yet often are produced with a small probability. A common type of mutation consists on randomly modifying one or more genes of an individual.

Also, elitism can be introduced. In that case, the best e individuals of the population will be maintained in the next generation, in order to keep their genetic material alive.

4.5.3 *Neuroevolution Procedure*

In our approach, we will use evolutionary algorithms to optimize the previously described hyperparameters of convolutional neural networks. The use of evolutionary computation with the purpose of evolving any aspect of neural networks is known in the literature as "neuroevolution".

In general, the evolutionary algorithm comprises a population of individuals, represented by a genotype, whose encoding depends on the specific technique and in some cases must be manually designed. Additionally, there must be a mechanism able to translate it into a phenotype, which will be a definition of a CNN topology by means of the previously specified hyperparameters.

Once we have a phenotype, we can evaluate its performance. To do so, we will train a neural network model with the given topology using a training dataset. Then, once the model has been

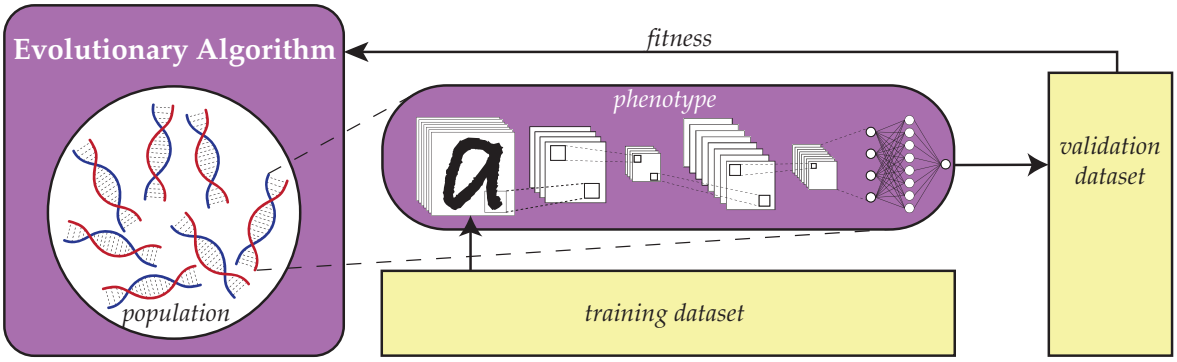


Figure 4.2: *General framework for neuroevolution of CNN topologies.*

learned, we will stop the training and will test its performance over a validation set, which is disjoint with the training set using a given metric (e.g. accuracy, precision, F1 score, etc). The value of this metric will be the fitness value of the individual, and this fitness will be provided as feedback to the evolutionary algorithm. A general framework for neuroevolution is displayed in figure 4.2.

However, such a procedure will have to deal with two fundamental problems:

- The large (potentially infinite) range in which hyperparameters can be tuned.
- The enormous time needed to evaluate each of the alternative designs.

To solve the first problem, we have decided to discretize the range of possible values that each hyperparameter can take. This decision reduces the search space while still making feasible to find good solutions, without reducing too much their potential quality. In most cases, intermediate values does not make significant differences in the design of the network, nor in its effectiveness. For instance, regarding the number of kernels we have chosen a maximum number of 256. However, we do not think that small differences in the number of kernels would have a significant impact on the network performance. Therefore, intermediate values of number of kernels have been ignored, giving the possibility to choose only between more significant values as 2, 4, 8, 16, 32, 64, 128 or 256. In most cases, only some few significant values have been chosen as alternative, regardless of whether finer adjustments can be performed in successive stages, if necessary.

The time required for the evaluation of each of the alternative solutions is one of the major disadvantages for considering a search procedure, even if it is very efficient. When dealing with population-based search methods, almost unanimously used in evolutionary computing, there cannot be a considerable improvement unless taking into account a large number of alternatives, which implies a large number of evaluations. In the present case, each evaluation involves the complete training of a convolutional neural network and its exploitation, making it unfeasible to handle enough number of alternatives for the system to produce significant improvements.

In order to overcome this drawback, we have decided to make estimations of the effectiveness of the networks rather than to carry out a thorough evaluation. Therefore, networks will be trained using only a reduced sample of existing data, and for a small number of training epochs. By doing so, the time required by the evaluation process is greatly reduced, at the cost of obtaining less accurate evaluations, an approach known as "fitness approximation". Nevertheless, estimating the effectiveness of networks with few samples, and few iterations, although providing poorer results, will rarely affect networks in an irregular manner. It is reasonable to assume that such estimations will not include significant biases towards particular architectures, nor will penalize in a particular way certain others. Furthermore, when using a selection operator based on the principles of natural

selection, a common property of most evolutionary computing techniques, accurate evaluations are not a requirement. What it is needed instead is a fair comparison between solutions to know which ones are better than the rest. Hence, the way estimations are performed in this work, are valid and effective in the scope of evolutionary searching.

To check whether this scheme is invariant to the evolutionary procedure involved, two different evolutionary methods have been used: a genetic algorithm (GA) and grammatical evolution (GE). We have decided to first use a GA as the evolutionary procedure for evolving CNN architectures, given its popularity and since it is a well-known evolutionary computation technique, which can be used as a baseline. Later, we have tackled the problem using GE in order to reduce the redundancy present in the GA binary encoding and to provide a more flexible definition of the CNN topology.

4.5.4 *Common Design Decisions*

As we have stated before, in order to optimize the CNN architecture we will run either genetic algorithms or grammatical evolution. Both techniques need:

- An encoding to represent individuals (potential solutions).
- A fitness function to evaluate the performance of an individual.
- A set of hyperparameters that controls the execution flow of the optimization algorithm.

Regarding the encoding, in this section we will explain its structure for each technique. However, the particular encoding will be described individually for each problem in chapter 5.

Also, we want the fitness function to be the metric m, which will often be maximized (unless it is an error metric which needs to be minimized). The specific fitness function will again be described for each individual problem in the next chapter.

For deciding the population size, the maximum number of generations and the stop condition, we needed to establish a trade-off that would guarantee an acceptable level of convergence reducing the time required by the evolutionary algorithm. After some preliminary experiments, we found that it was unlikely that an improvement was achieved after 30 generations without improvements; and, given this stop criterion, the algorithm would in most cases finish before 100 generations occurred. Finally, the population size will involve 50 individuals, so running an experiment with either GA or GE will involve at most 5,000 fitness evaluations. Of course, convergence is not guaranteed under these terms, but it remains a good approximation that establishes an acceptable upper bound for the time required by the algorithm to complete.

Finally, we have decided to implement a hall-of-fame of 50 individuals. This hall-of-fame will store the best 50 individuals found so far during the evolutionary process. By doing so, we will not only consider the best solution found, but also additional solutions which are close to the best in terms of quality. The reason to store this hall-of-fame is twofold: First, since the fitness function is only a proxy of the quality metric using a reduced dataset and few epochs, it could happen that the best individual in terms of fitness does not necessarily correspond the best topology. Second, since we are interested in exploring the behavior of committees of CNNs, we need to store several topologies in order to test this approach.

4.5.5 *Accelerating Fitness Computation*

As previously mentioned, we have performed simplifications to reduce the training time. In particular, in each fitness computation the network is trained only during 5 epochs, using a random

sample of the training set in each epoch. The obtained result is used as a proxy of a more exhaustive training, taking only a fraction of the time required to train a network with the whole training set.

This simplification results in a pessimistic estimation of the classification capacity of the network, but will not affect the evolutionary process, since nothing suggests that the proposed mechanism may introduce some bias or preferences in the evaluation of some individuals over others. What is important in the evolutionary process is not the precision of the evaluations, but rather to allow a fair comparison between the different alternatives. In addition, in this case, to avoid any possible bias, the tournament selection operator has been used, in which the probability of selecting individuals to generate successors does not depend on the difference of the values of the evaluations, but on the position in a ranking.

4.5.6 *Preserving Genetic Diversity*

Two measures were taken in order to guarantee genetic diversity across generations. First, it should be noted that many CNN architectures are invalid (cannot be executed because the number or size of the convolutional or pooling layers would require a larger input). Due to their inability to be trained, we have assigned all these individuals a fitness of 0. For this reason, if the initial population were initialized randomly, many of the individuals would have have zero-fitness, and as a result, because only a few individuals in the initial population are suitable, the genetic diversity would decrease significantly. To tackle this issue, we have iteratively randomly re-initialized those individuals in the initial population with zero-fitness until all individuals are suitable.

Secondly, we have implemented a niching strategy, as we have found that otherwise many individuals converge to a single solution, an issue that could lead to local optima and would prevent us from building a committee of neural networks from the population. We address the niching scheme by having two separate fitness values for each individual: the nominal fitness and the adjusted fitness. The nominal fitness is the fitness as we have described so far: the value of m over a dataset disjoint from the training set. The adjusted fitness is computed from the nominal fitness and will decrease if the individual is "very similar" to the rest of individuals of the current population.

To better understand how the adjusted fitness is computed, we first have to define the concept of "similarity" between individuals. Equation 4.2 provides a formal mathematical definition of similarity between two individuals, i_i and i_j:

$$sim(i_i, i_j) = \begin{cases} \frac{|p(i_i) \cap p(i_j)|}{|p|}, & \text{if } i_i[n_c] = i_j[n_c] \text{ and } i_i[n_d] = i_j[n_d] \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

Let us explain equation 4.2 in further detail. First, we will only consider two individuals to have certain degree of similarity when they have the same number of convolutional layers and of dense layers, otherwise they will be considered as completely different. In case both n_c and n_d match for the two individuals, we will check how many properties they have in common. A property is a certain optimizable hyperparameter of the CNN topology, e.g. the batch size (B), the optimizer (f), the number of neurons in the first dense layer (dn_1) or the learning rate (η), to mention a few. In equation 4.2, $|p(i_i) \cap p(i_j)|$ refers to the cardinality of the intersection of the properties set of both individuals, i.e., the number of properties that both have in common, whereas $|p|$ is the total number of properties. It should be noted that, because both individuals have the same number of layers, they will also have the same number of properties.

Once we have defined the concept of similarity between a pair of individuals (note that the similarity function is reciprocal), then we can specify how the adjusted fitness (f_a) is computed from the nominal fitness (f_n). The mathematical description is provided in equation 4.3.

$$f_a(i_i) = f_n(i_i) \times \left(1 - \frac{\sum_{i_j \in P, j \neq i} sim(i_i, i_j)}{|P| - 1}\right) \tag{4.3}$$

In summary, the adjusted fitness is computed just as the product of the nominal fitness by a factor which is inverse to the average similarity of the individual with the rest of individuals of the population. In equation 4.3, P is the set of individuals of the current population. We can notice how, if the individual were completely different than the rest of the population (i.e., its similarity with all other individuals were always zero), then the adjusted fitness and the nominal fitness would be equivalent. On the opposite hand, if all individuals of the population were identical, then all adjusted fitnesss would be zero regardless of the nominal fitnesss. Both extreme cases are very unlikely to happen in a real-world scenario. It is worth noting that this equation is only valid when the fitness function must be maximized, and should be otherwise adapted by replacing the product by a division and adding extra logic to handle the case of a division by zero.

It is worth noting that we expect this increase in genetic diversity to be positive not only for the evolutionary process, by preventing the evolution towards local suboptima, but also for the performance of committees of CNNs, where we expect these committees to perform better when there is a higher variance between the different models involved. Also, it is important to notice that the hall-of-fame will store the best individuals according to their nominal fitness, not their adjusted fitness, since we are interested in preserving the best individuals found in terms of metric m.

4.5.7 Genetic Algorithm

Now that we have introduced some basic terminology and described some design decisions, we can proceed to explain the specifics of the first evolutionary computation technique that will be used in this thesis: the genetic algorithm.

As we stated before, we have chosen to first use a genetic algorithm since it is a well-known technique that has been widely used in many neuroevolution applications. While it poses some disadvantages that we will tackle later when we use grammatical evolution, it constitutes a good start point to test our proposal.

4.5.7.1 Encoding

First, we need to explain the encoding. While the encoding in a genetic algorithm comprises a vector of values belonging to any data type, it is common to use binary vectors (or chromosomes), i.e., a sequence of 0s and 1s. This representation is really convenient as it can actually encode any other data type: integers, floating point numbers, strings, etc; and is the one that we will use in our proposal. An example of a binary chromosome of length 20 is shown in figure 4.3.

In GAs, the researcher must provide an explicit mapping function between the genotype and the phenotype. In our case, a detailed explanation on how the phenotype is obtained from the genotype will be provided in the next chapter, since it is slightly adapted to each particular problem.

0	1	1	0	0	1	0	1	0	0	0	1	1	1	1	0	1	1	0	1

Figure 4.3: *Example of a 20-bits chromosome for a genetic algorithm.*

4.5.7.2 Genetic Operators

In our proposal, we will be using the most common genetic operators usually found in evolutionary algorithms: selection, reproduction, and mutation. Additionally, we have observed the inclusion of elitism in order to keep the best individuals across generations. More specifically, the operators that we will use in our GA implementation are the following:

- Tournament selection, with tournaments of size τ. The individual with the highest adjusted fitness (f_a) wins the tournament.
- Multi-point crossover: because we will deal with long chromosomes when optimizing CNN topologies, we have implemented the reproduction operator by means of a multi-point crossover, with a variable number of points. An example of the procedure for multi-point crossover with 3 points is illustrated in figure 4.4. It can be seen how the genetic material of both parents is combined to create two children. The number of points is randomly chosen between x_{min} and x_{max} following a uniform distribution.
- Bit-flipping mutation, with a mutation rate of α. All individuals are mutated before passing to the next generation, with the sole exception of those individuals belonging to the elite. An example of the bit-flipping mutation operator is shown in figure 4.5, and it should be noted that none, one or more than one bit can be mutated, depending on the value of α and the outcome of a random variable.
- Elitism of size e. By applying elitism, we will keep the best e individuals from one generation in the next generation. It must be noted that elite individuals are chosen based on the nominal fitness (f_n) rather than the adjusted fitness.

It is important to realize that the hall-of-fame is completely unrelated with the use of elitism. In particular, elitism affects the evolutionary process, since it has an impact in the population of the upcoming generations. On the other hand, the hall-of-fame is rather a checkpoint of the process at a given time, expressed as the set of the best individuals found so far; yet these individuals do not have any impact in the evolution itself.

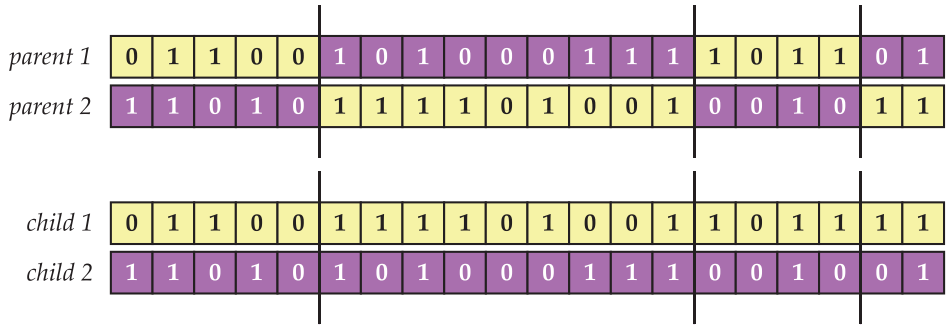


Figure 4.4: *Example of multi-point crossover with 3 points in the GA.*

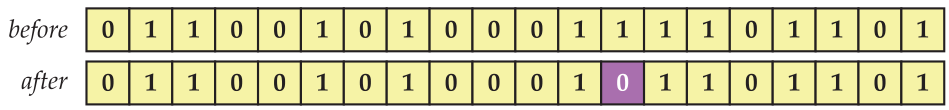


Figure 4.5: *Example of bit-flipping mutation in the GA.*

4.5.7.3 *Procedure*

Regarding the flow of execution of genetic algorithms, high level pseudocode is provided in algorithm 4.1. It can be seen how the algorithm resembles a canonical GA, except for the introduction of the niching scheme.

Table 4.1 outlines the different hyperparameters used in the GA along with their value.

Algorithm 4.1: *Genetic Algorithm*

$P_0 \leftarrow$ random population of S individuals
evaluate individuals in P_0
while $i \in P$ **and** $f_n(i) = 0$ **do** `// ensure` P_0 `has no zero-fitness individuals`
  $P_0 \leftarrow P_0 - \{i\}$
  $n \leftarrow$ new random individual
  evaluate n
  $P_0 \leftarrow P_0 + \{n\}$
end
for $g \leftarrow 1$ **to** G **do**
  $P' \leftarrow$ new empty (intermediate) population
  while $|P'| < S$ **do**
    $P' \leftarrow P' +$ `Tournament(`P_{g-1}, τ`)` `// ind is chosen via a` τ`-sized tournament`
  end
  $P_g \leftarrow$ new empty population
  for $i \leftarrow 1$ **to** S **with step** 2 **do**
    $ps \leftarrow P'[i], P'[i+1]$ `// retrieve the parents`
    $cs \leftarrow$ `Crossover(`ps`)` `// parents reproduce to obtain 2 children`
    $mcs \leftarrow$ `Mutation(`cs, α`)` `// mutate the offspring with mutation rate` α
    $P_g \leftarrow P_g + mcs$ `// add mutated children to` P_g
  end
  evaluate individuals in P_g
  compute adjusted fitness for individuals in Pg `// niching strategy, see eq 4.3`
  remove the e worst individuals from P_g
  $P_g \leftarrow P_g +$ `Elitism(`P_{g-1}, e`)` `// add the` e `fittest ind. from` P_{g-1} `into` P_g
  if `StopCondition(`g, G_s`)` **then**
    break
  end
end

Hyperparameters	Symbol	Value
Population size	$\|P\|$	50
Maximum number of generations	G	100
Number of generations without improvements (stop condition)	G_s	30
Tournament size	τ	3
Minimum number of points in multi-point crossover	x_{min}	3
Maximum number of points in multi-point crossover	x_{max}	10
Mutation rate	α	0.015
Elite size	e	1

Table 4.1: *List of hyperparameters used in the genetic algorithm, with their values.*

4.5.8 Grammatical Evolution

After the implementation of the genetic algorithm, we have decided to use grammatical evolution as a secondary implementation of an evolutionary algorithm to test our proposal. Whereas grammatical evolution has been less used in the literature than genetic algorithms, we have decided to use it because it can overcome some drawbacks of the GA.

Specifically, the genetic algorithm will often present more redundancy in its encoding, this meaning that different genotypes can map into the exact same phenotype. This is specially true since we will be using fixed-length chromosomes but allowing a variable number of layers in the phenotype, as will explain in the following chapter. Also, the genetic algorithm provides less flexibility in the mapping process. The reason is that we will encode a hyperparameter using an integer number of bits, thus meaning that the number of values for any given hyperparameter will be a power of 2. This might be convenient in some cases, but may be forcing us to include some undesired values just to avoid additional redundancy.

Finally, GE could potentially allow us to encode "infinite" individuals, by using recursion in the provided grammar. Nevertheless, we will not use this feature in this thesis, since we want to keep the search space bounded.

We will now proceed to explain the specific encoding, operators and procedure used for our implementation of grammatical evolution.

4.5.8.1 Encoding

As opposed to genetic algorithms, in grammatical evolution the encoding is given by the technique itself. In particular, individuals in GE are encoded as a vector of integer numbers, also called *codons*. The researcher does not have to make any further assumptions about the encoding, and only has to take two decisions: the codon size (the maximum value an integer in the vector can take) and the maximum chromosome length.

Also, in GE the method for decoding the genotype into a phenotype is already given by the technique, unlike the case of the GA where the mapping function must be provided by the researcher. However, in GE the researcher does have to provide a formal grammar that is used by the mapping function in order to generate valid phenotypes from the genotype. In particular, the formal grammar will generate a language, such that the set of words in that language is the set of valid phenotypes. It is worth noting that a language can be potentially infinite if the grammar is recursive. The process for translating the genotype into a phenotype given a chromosome and a formal grammar was described in section 2.11.

In GE, the chromosome will comprise a maximum number of codons, from which not all may be used during the decoding process. As a result, this turns out to be a more natural approach to encode variable-length solutions.

4.5.8.2 Genetic Operators

In GE, we will use the same genetic operators as in the GA, just with some adaptations to better fit the technique. In particular, these genetic operators are the following:

- Tournament selection, with tournament size τ. The individual with the highest adjusted fitness (f_a) wins the tournament.

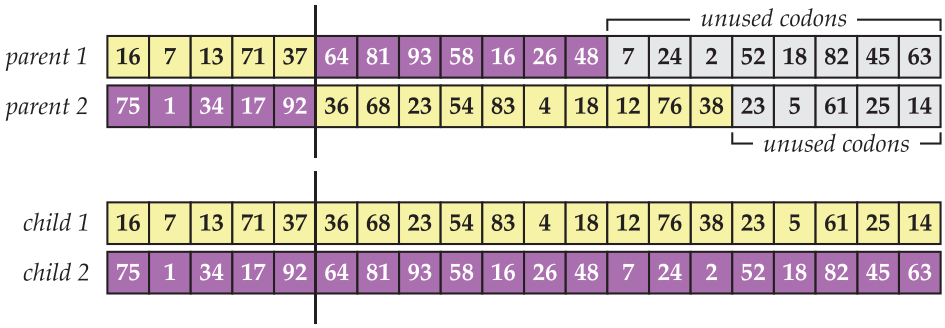


Figure 4.6: *Example of single-point crossover in GE.*

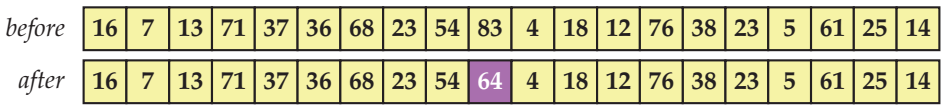


Figure 4.7: *Example of integer-flipping mutation in GE.*

- Single-point crossover: in GE, the reproduction will be performed using only one point for crossover, and will occur with a probability of β. If crossover does not occur, then parents will be present in the following population. Crossover is forced to occur within the subsequences of the chromosomes that are actually used, thus guaranteeing that the crossover has an effective impact in the phenotype. An example of this process is shown in figure 4.6 (it should be noted that we do not know the number of unused codons in the offspring until they are translated into the phenotype). The reason for using single-point instead of multi-point crossover is that the chromosomes are effectively shorter than in the case of the GA.
- Integer-flipping mutation, with a mutation rate of α. The old integer will be replaced by a new random integer, as shown in figure 4.7. All individuals are mutated, with the sole exception of elite individuals. Mutation also affects the parents that are not crossed and are passed to the next generation. None, one or more than one position can be mutated, depending on the value of α.
- Elitism of size e. It must be noted that elite individuals are chosen based on the nominal fitness rather than the adjusted fitness.

4.5.8.3 Procedure

The procedure for our implementation of the grammatical evolution is very similar to the previously described implementation of the genetic algorithm. The main differences between both were already described before when we explained the genetic operators, being the most noticeable difference the crossover operator, which is aware of the number of unused codons in a genotype and is only performed with a crossover rate of β.

The other implementation differences involve the specifics of each technique: in GE, the mapping function for converting the genotype into a phenotype involves the use of a formal grammar $\mathcal{G}$ specified by the researcher. This formal grammar will be specific to each problem and its definition in Backus-Naur form (BNF) will be provided in the following chapter for each different domain.

As in the case of the GA, we will keep a hall-of-fame that will store the best 50 individuals. The niching scheme is also implemented in the same way as in the GA.

The pseudocode for our implementation of grammatical evolution is shown in algorithm 4.2. Because it is described in a rather abstract fashion, the differences with the GA pseudocode are quite subtle, since both procedures follow the same evolutionary schema.

The different hyperparameters used in our proposed implementation of the GE, along with their values, are shown in table 4.2.

Algorithm 4.2: *Grammatical Evolution*

$P_0 \leftarrow$ random population of S individuals
evaluate individuals in P_0 given grammar $\mathcal{G}$
while $i \in P$ **and** $f_n(i) = 0$ **do** `// ensure` P_0 `has no zero-fitness individuals`
    $P_0 \leftarrow P_0 - \{i\}$
    $n \leftarrow$ new random individual
    evaluate n given grammar $\mathcal{G}$
    $P_0 \leftarrow P_0 + \{n\}$
end
for $g \leftarrow 1$ **to** G **do**
    $P' \leftarrow$ new empty (intermediate) population
    while $|P'| < S$ **do**
        $P' \leftarrow P' +$ `Tournament(`P_{g-1}, τ`)` `// ind is chosen via a` τ`-sized tournament`
    end
    $P_g \leftarrow$ new empty population
    for $i \leftarrow 1$ **to** S **with step** 2 **do**
        $ps \leftarrow P'[i], P'[i+1]$ `// retrieve the parents`
        $cs \leftarrow$ `Crossover(`ps, β`)` `// parents reproduce with crossover rate` β
        $mcs \leftarrow$ `Mutation(`cs, α`)` `// mutate the offspring with mutation rate` α
        $P_g \leftarrow P_g + mcs$ `// add mutated children to` P_g
    end
    evaluate individuals in P_g given grammar $\mathcal{G}$
    compute adjusted fitness for individuals in Pg `// niching strategy, see eq 4.3`
    remove the e worst individuals from P_g
    $P_g \leftarrow P_g +$ `Elitism(`P_{g-1}, e`)` `// add the` e `fittest ind. from` P_{g-1} `into` P_g
    if `StopCondition(`g, G_s`)` **then**
        break
    end
end

Hyperparameters	Symbol	Value
Population size	$\|P\|$	50
Maximum number of generations	G	100
Number of generations without improvements (stop condition)	G_s	30
Codon size		256
Maximum chromosome length		100
Tournament size	τ	3
Crossover rate	β	0.7
Mutation rate	α	0.015
Elite size	e	1

Table 4.2: *List of hyperparameters used in grammatical evolution, with their values.*

4.6 Summary

In this chapter we have explained our proposal for a system which automatically designs the topology of convolutional neural networks. We have concluded that metaheuristics are a suitable approach for tackling this problem and the challenges it poses; in particular, evolutionary computation has been used for almost three decades for similar purposes, and is established as a research field known as "neuroevolution". However, the applications of neuroevolution to CNNs are very recent and scarce, and the field remains mostly unexplored. However, existing works, which were reviewed in chapter 3, look promising, and we have decided to further explore this research line.

After further study, we have decided to explore two techniques within the field of evolutionary computation: genetic algorithms and grammatical evolution. Both techniques are based or the concepts of Darwinian theory of evolution to evolve populations of individuals, eventually achieving fittest solutions. The first technique is a classical approach using a binary encoding of individuals, whereas the second uses a formal grammar in order to generate these individuals, and has been designed to overcome some of the limitations posed by genetic algorithms.

In order to keep the population genetically diverse, we have included a niching scheme in both techniques. This scheme penalizes the fitness of those individuals that are very similar to the rest of the population. By doing so, we can achieve more diverse individuals, and this could be an advantage for building ensembles or committees of CNNs.

In the following chapter we will evaluate the performance of the current proposal using different domains, describing the specific setup for each of them.

This page has been intentionally left blank.

Chapter 5

Evaluation

To validate and evaluate the performance of the proposal described in the previous chapter, we will use different databases. These databases comprise very different domains (e.g.; pictures, sensors, etc.) and have been thoroughly addressed in the literature, thus enabling us to perform an exhaustive comparative evaluation with the best results found so far in the state of the art. We have intentionally chosen a diverse set of domains in order to ensure that our proposal is able to optimize a CNN classifying very different types of data.

First, in section 5.1 we will thoroughly describe the evaluation environment in terms of hardware and software configuration, in order to enable proper reproducibility of the experiments.

Later in this chapter, we will go through each of these databases, describing how the data was acquired and how it is structured, thoroughly analyzing the current state of the art, explaining how the proposed evolutionary computation system is applied to each domain, and discussing the results obtained in our approach.

The first domain will be the well-known database MNIST for handwritten digit recognition, which is discussed in section 5.2. Later, in section 5.3 we will reuse the best topologies found with this database to a new set of data, EMNIST, which contains additional instances and provides a new domain of handwritten letters recognition. Finally, we will try the very different domain of human activity recognition, using the resources available in the OPPORTUNITY dataset, describing the whole process in section 5.4

By the end of the chapter, we will provide some conclusive remarks on how our proposal compares to the state of the art for all the reviewed datasets.

5.1 Evaluation Environment

In this section we will describe the environment used for running the experiment, including both hardware components and the software stack used for training the convolutional neural networks.

5.1.1 Hardware Configuration

We have run all the experiments in two compute nodes with two GPUs each. With this configuration, we could train up to 4 CNNs in parallel. Each compute node comprises the hardware components described in table 5.1.

Component	Description
CPU	Intel Core i7-6700 CPU *LGA1151 4 Cores @ 3.4 GHz* *14 nm Architecture*
Motherboard	ASUS H170 PRO Gaming
Memory	Corsair Vengance LPX (2×8 GB) *DDR4 @ 2133 MHz*
Hard Drive (Primary)	Kingston SSDNow 240 GB SATA3
Hard Drive (Secondary)	Western Digital Caviar Blue 2 TB SATA3
GPU	ASUS STRIX NVIDIA GeForce GTX1080 (×2) *2560 CUDA Cores @ 1784 MHz* *8 GB DDR5X @ 10010 MHz*

Table 5.1: *Hardware components in the evaluation environment.*

Most of the computational power of this hardware configuration resides in the GPUs, each one having 8 GB of DDR5X RAM memory and 2560 CUDA cores, enabling massively parallel computations. The detailed architecture of NVIDIA GeForce GTX 1080 is outlined in appendix A, and a more in-depth analysis of the performance of these devices, including the reasons for choosing this configuration instead of provisioning a specific deep-learning cluster is provided in appendix B. CPU cores have not been used for training CNNs.

5.1.2 Software Stack

The software stack used in the evaluation environment is summarized in figure 5.1, where the bottom part shows the low-level elements and the top depicts the most abstract pieces of software.

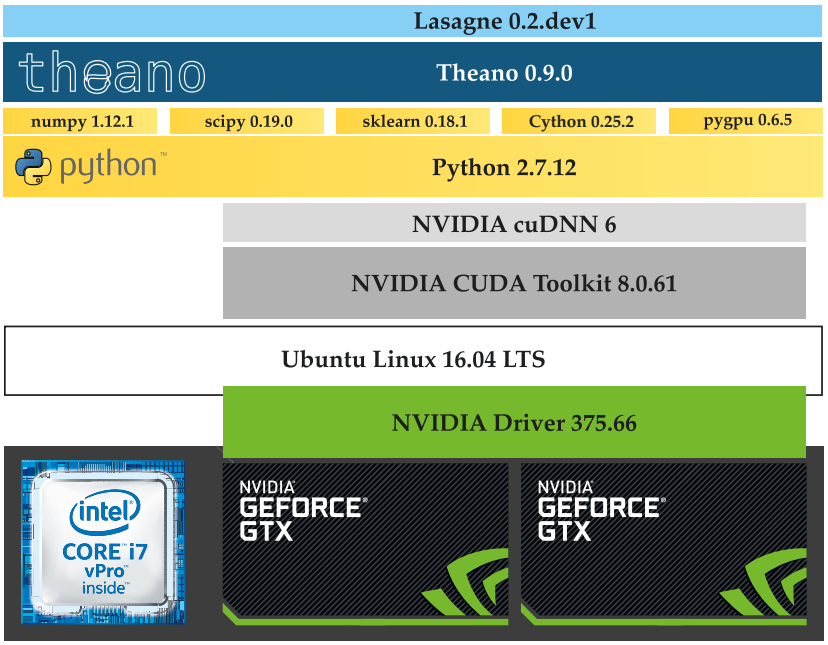


Figure 5.1: *Software stack in the evaluation environment.*

In the bottom, we can find the hardware configuration as described in the previous section. The main computational elements are an Intel Core i7 processor and two NVIDIA GeForce GTX 1080 graphical processing units (GPUs). For the operating system, we have used Ubuntu Linux 16.04 LTS, as LTS (long-term support) releases are often considered to be more stable. In order to be able to use the NVIDIA GPUs, we have installed the latest stable release of NVIDIA proprietary drivers as of May 2017, which is version 375.66.

NVIDIA drivers are not enough in order to run computational programs in the GPU cores. To do so, we require the CUDA Toolkit, which provides GPU accelerated libraries [272] for high-performance computing. The latest version as of May 2017 is CUDA Toolkit 8.0. Besides CUDA, NVIDIA has released a GPU-accelerated library of primitives for deep neural networks, named cuDNN [273]. This library is very convenient as all of the most relevant deep learning libraries (including TensorFlow, Theano, Torch, Caffe, DL4J, etc.) are able to optimize deep learning operations by running them on top of cuDNN. As of May 2017, the latest available version is cuDNN 6, and we have decided to deploy this version in our compute nodes.

For the programming language and interpreter, we will use Python 2.7.12, which is to the date the latest available version in Ubuntu software repositories. We have installed the following libraries using PIP package manager, which are required for developing and running the system: `numpy` 1.12.1, `scipy` 0.19.0, `sklearn` 0.18.1, `Cython` 0.25.2 and `pygpu` 0.6.5. All these libraries are installed in their latest versions as of May 2017. Also, version 5.4.0 of the C++ compiler is installed, as it is required for code optimization.

The chosen deep learning framework is Theano 0.9.0 due its versatility. The bleeding-edge version available as of May 2017 has been installed. Also, Lasagne 0.2.dev1 was used in order to simplify the design of CNN topologies, as it turns out to be a very convenient abstraction for our system, with little overhead and impact on performance.

The code for the genetic algorithm has been written from scratch. For grammatical evolution, we have used a custom implementation based on PonyGE 0.1.5 [275].

5.2 MNIST

The MNIST (Mixed National Institute of Standards and Technology) database was introduced in 1998 by Yann LeCun, Corinna Cortes and Christopher J.C. Burges [204]. Since then, MNIST has been used extensively to test machine learning applications and pattern recognition techniques.

5.2.1 Acquisition

The MNIST database contains 60,000 training samples and 10,000 test samples of grayscale handwritten digits. Half of the data comes from NIST's Special Database 1, which was collected among high-school students. The other half was retrieved from NIST's Special Database 3, which was collected among Census Bureau employees. The set of writers of the training set and the test set is disjoint, and the training set contains samples from over 250 writers.

The original images were normalized to fit in a 20x20 pixel box, while preserving their aspect ratio. Images were originally black and white, though they were converted into grayscale after applying an anti-aliasing filter during the normalization process. Finally, padding was added in order to fit the images in a larger 28x28 pixels figure, so that the center of mass of the pixels matched the center of this 28x28 box.

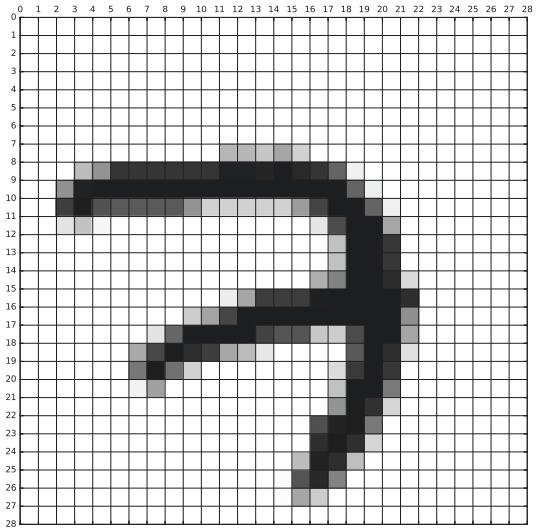


Figure 5.2: *MNIST sample corresponding to the digit '7'.*

Figure 5.3: *10 samples for each digit from the MNIST training set.*

Figure 5.2 shows an example of one sample retrieved from the MNIST training set corresponding to the digit '7' and displaying the 28x28 pixels grid. Meanwhile, figure 5.3 displays 10 samples for each digit between 0 and 9 retrieved from the MNIST training set.

While the task of guessing a handwritten digit may seem easy for a human, some particular samples can be easily confused: for example, a '4' can be mixed up with a '9', and some digits can be hard to recognize (examples can be found in figure 5.3, such as the ninth '2' or the ninth '7').

5.2.2 State of the Art

MNIST database has been used extensively to evaluate the performance of ML classification techniques. For this reason, several rankings have been published in the past, which use MNIST to perform a comparative evaluation. Most of the existing literature in MNIST uses the "test error rate" (in %) as the metric to evaluate the performance of machine learning techniques. This metric is computed as the ratio between the number of incorrectly classified instances and the total number of instances in the test set; thus, it is equivalent to $1 - a$, where a is the classification accuracy.

One of the earliest rankings was published by LeCun et al. themselves [204], which includes references up to 2012. This website provides a taxonomy of classifiers and points out whether each work performed data preprocessing or augmentation (addition of new instances to the training set resulting from distortions or other modifications of the original data). In this ranking, we can verify that the early machine learning approaches used by LeCun et al. [203] included linear classifiers (whose error rate ranges from 7.6 % to 12 %), K-nearest neighbors approaches (ranging from 1.1 % to 5 %), non-linear classifiers (about 3.5 %), support vector machines (from 0.8 % to 1.4 %), neural networks (from 1.6 % to 4.7 %) and convolutional neural networks (from 0.7 % to 1.7 %). It is remarkable that data augmentation leads to better results, in particular, the best error rate achieved using a convolutional neural network with no distortions and no preprocessing in LeCun's work [203] was 0.83 %.

From the different works gathered in LeCun et al.'s ranking, those based in convolutional neural networks outperform the other techniques. However, there are some classical machine learning

techniques which are still able to provide competitive error rates (in this work, we will consider "competitive" those techniques achieving an error rate under 1.0 %). For instance, Belongie et al. [23] achieved 0.63 % and Keysers et al. [180] achieved 0.54 % and 0.52 % using K-NN, Kégl et al. [177] achieved 0.87 % using boosted stumps on Haar features, LeCun et al. [203] achieved 0.8 % and DeCoste and Schölkopf [85] attained results from 0.56 % to 0.68 % using SVM.

When using non-convolutional neural networks, Simard et al. [333] achieved 0.9 % and 0.7 % respectively using a 2-layer neural network with MSE and cross-entropy loss functions respectively. Deng and Yu [87] achieved 0.83 % using a deep convex network without data augmentation or preprocessing. Very interesting results were attained by Meier et al. [237] (0.39 %) using a committee of 25 neural networks and Cireşan et al. [68] (0.35 %) using a 6-layers neural network[1].

On the other hand, works based on convolutional neural networks attained a much better average performance (in fact, the worst result reported in LeCun's ranking was 1.7 %). LeCun et al. [203] combined different convolutional architectures along with data augmentation techniques, obtaining error rates ranging from 0.7 % to 0.95 %. Lauer et al. [197] attained between 0.54 % and 0.83 % using a trainable feature extractor along with SVMs, and Labusch et al. [192] reported an error rate of 0.59 % using a similar technique consisting on a CNN to extract sparse features and SVMs for classification. Simard et al. [333] obtained error rates between 0.4 % and 0.6 % using a CNN with cross-entropy loss function and data augmentation. Ranzato et al. reported results between 0.39 % and 0.6 % using a large CNN along with unsupervised pretraining [297]; and some years later Jarrett et al. [165] reported a test error of 0.59 % with a similar approach technique and without data augmentation. The best results in this ranking are those obtained by Cireşan et al., which reported an error rate of 0.35 % using a large CNN [70], and 0.27 % [69] and 0.23 % [71] using committees of 7 and 35 neural networks respectively, using data augmentation in all cases.

In 2015, McDonnell et al. [235] proposed an approach using 'extreme learning machine' (ELM) [157] algorithm to train shallow non-convolutional neural networks, and they agree on the fact that data augmentation with distortions can reduce error rates even further. In their work, McDonnell et al. compare their results with other previous ELM and selected non-ELM approaches, updated with some CNN-based works as of 2015. The most outstanding ELM-based results achieved test error rates of 0.97 % in the work by Kasun et al. [176], and ranged from 0.57 % to 1.36 % in the proposal by McDonnell et al. [235].

However, most interesting results included in the comparison by McDonnell et al. are not those provided in that work, but those from previous works where convolutional neural networks were used. For example, Wan et al. [378] used CNNs with a generalization of dropout they called DropConnect, and reported an error rate of 0.57 % without data augmentation and as low as 0.21 % with data augmentation. Zeiler and Fergus [408] proposed the use of stochastic pooling achieving an error rate of 0.47 %. Goodfellow et al. [120] described the *maxout* model averaging technique, attaining a test error rate of 0.45 % without data augmentation. In 2015, Lee et al. [207] described "deeply-supervised nets", an approach by which they introduce a classifier (SVM or softmax) at hidden layers, reporting a result of 0.39 % without data augmentation.

A more recent ranking, updated as of 2016, has been made available by Rodrigo Benenson in his GitHub's page [24]. This ranking includes many of the works reviewed so far, as well as other with very competitive results. For example, Sato et al. [316] explore and optimize data augmentation, and attain a test error rate of 0.23 % in the MNIST database using a convolutional neural network. A very interesting result is that reported by Chang and Chen [57], where an error rate of 0.24 % (the best result found as of 2018 in the state of the art without data augmentation) was obtained using a network-in-network approach with a maxout MLP, an approach they named MIN. Also, very competitive performances without data augmentation were reported by Lee et al. [206] using gated

[1]It is worth mentioning that the reproducibility of this result has been put into question in 2016 by C. H. Martin [231] in his blog.

pooling functions in CNNs (0.29 %), by Liang et al. [212] using a recurrent CNN (0.31 %), by Liao and Carneiro when using CNNs with normalization layers and piecewise linear activation units (0.31 %) [214] and competitive multi-scale convolutional filters (0.33 %) [213], or by Graham [126] using fractional max-pooling with random overlapping (0.32 %).

Additional works where state-of-the-art results were obtained without data augmentation include those by McDonnell and Vladusich [236], who reported a test error rate of 0.37 % using a fast-learning shallow convolutional neural network, Mairal et al. [227], who achieved 0.39 % using convolutional kernel networks, Xu et al. [395], who explored multi-loss regularization in CNNs obtaining an error rate of 0.42 %, and Srivastava et al. [341], who used so-called convolutional "highway" networks (inspired by LSTM recurrent networks) to achieve an error rate of 0.45 %. Lin et al. [216] reported an error rate of 0.47 % using a network-in-network approach, where micro-neural networks are used as convolutional layers. Ranzato et al. [296] used convolutional layers for feature extraction and two fully connected ANNs for classification, attaining an error rate of 0.62 %. Bruna and Mallat [46] designed a network that performed wavelet transform convolutions to the input and a generative PCA classifier to obtain an error of 0.43 %. Calderón et al. [49] proposed a variation of a CNN where the first layer applies Gabor filters over the input, obtaining an error of 0.68 %. Also, Le et al. [201] explored the application of limited-memory BFGS (L-BFGS) and conjugate gradient (CG) as alternatives to stochastic gradient descent methods for network optimization, achieving 0.69 % error rate. Finally, Yang et al. [398] developed a new transform named Adaptative Fastfood to reparameterize the fully connected layers, obtaining a test error rate of 0.71 %.

An interesting approach is followed by Hertel et al. [149], in which they reuse the convolutional kernels previously learnt over the ILSVRC-12 dataset, proving that CNNs can learn generic feature extractors than can be reused across tasks, and achieving a test error rate in the MNIST dataset of 0.46 % (vs. 0.32 % when training the network from scratch). In this thesis we will follow a similar approach by reusing the topologies learned on MNIST with an extended version of the dataset. Another approach based on neural architecture search is that by Saxena and Verbeek [317], named "convolutional neural fabrics", which attain a test error rate of 0.33 % using data augmentation; and the work by Wistuba [391] which attained a test error rate of 0.31 % without data augmentation using Monte Carlo planning for optimizing the network hyperparameters.

It can be seen how most recent works are based on convolutional neural networks, due to their high performance. However, some exceptions can be noticed. One example is the work by Wang and Tan in late 2016 [380] where they attained a test error rate of 0.35 % using a single layer centering support vector data description (C-SVDD) network with multi-scale receptive voting and SIFT (scale-invariant feature transform) features. Also, Zhang et al. [409] devised a model (HOPE) for projecting features from raw data, that could be used for training a neural network, that attained a test error rate of 0.40 % without data augmentation. Visin et al. [375] proposed a recurrent neural network (ReNet) that replaced the convolutional layers by recurrent neural networks that swiped through the image, attaining an error rate of 0.45 % with data augmentation.

Additional examples with less performance include the work by Azzopardi and Petkov [9] who used a combination of shifted filter responses (COSFIRE), attaining an error rate of 0.52 %. Also, Chan et al. [55] used a PCA network to learn multi-stage filter banks, and report an error rate of 0.62 %. Mairal et al. [226] used task-driven dictionary learning, reporting a test error rate of 0.54 %, yet using data augmentation. Jia et al. [166] explored the receptive field of pooling, and obtained an error of 0.64 %. Thom and Palm [359] explored the application of the sparse connectivity and sparse activity properties to neural networks, obtaining an error rate of 0.75 % using a supervised online autoencoder with data augmentation. Lee et al. [208] described convolutional deep belief networks with probabilistic max-pooling achieving an error rate of 0.82 %. Min et al. [246] reported an error of 0.95 % using a deep encoder network to perform non-linear feature transformations which are then introduced to kNN for classification. Finally, Yang et al. [396] used supervised translation-invariant sparse coding with a linear SVM attaining an error rate of 0.84 %.

Technique	Test Error Rate
NN 6-layer 5,700 hidden units [68]	0.35 %
MSRV C-SVDDNet [380]	0.35 %
Committee of 25 NN 2-layer 800 hidden units [237]	0.39 %
ReNet [375]	0.45 %†
K-NN (P2DHMDM) [180]	0.52 %
COSFIRE [9]	0.52 %
K-NN (IDM) [180]	0.54 %
Task-driven dictionary learning [226]	0.54 %
Virtual SVM, deg-9 poly, 2-pixel jit [85]	0.56 %
RF-C-ELM, 15000 hidden units [235]	0.57 %
PCANet (LDANet-2) [55]	0.62 %
K-NN (shape context) [23]	0.63 %
Pooling + SVM [166]	0.64 %
Virtual SVM, deg-9 poly, 1-pixel jit [85]	0.68 %
NN 2-layer 800 hidden units, XE loss [333]	0.70 %
SOAE-σ with sparse connectivity and activity [359]	0.75 %
SVM, deg-9 poly [203]	0.80 %
Product of stumps on Haar f. [177]	0.87 %
NN 2-layer 800 hidden units, MSE loss [333]	0.90 %
CNN (2 conv, 1 dense, ReLU) with DropConnect [378]	**0.21 %**
Committee of 25 CNNs [71]	0.23 %
CNN with APAC [316]	0.23 %†
CNN (2 conv, 1 dense, ReLU) with dropout [378]	0.27 %
Committee of 7 CNNs [69]	0.27 %
CNF dense [317]	0.33 %
Deep CNN [70]	0.35 %
CNN (2 conv, 1 dense), unsup pretraining [297]	0.39 %
CNN, XE loss [333]	0.40 %
Scattering convolution networks + SVM [46]	0.43 %
Feature Extractor + SVM [197]	0.54 %
CNN Boosted LeNet-4 [203]	0.70 %
CNN LeNet-5 [203]	0.80 %

Table 5.2: *Side-by-side comparison of the most competitive (error rate < 1 %) results found in the state of the art for the MNIST database with data augmentation or preprocessing.*

Other approach to solve the MNIST classification problem involves so-called "deep Boltzmann machines". The first application of this technique was suggested by Salakhutdinov and Hinton in 2009 [312] and enabled them to report an error rate of 0.95 %. Few years later, Goodfellow et al. [119] introduced the multi-prediction deep Boltzmann machine, achieving an error rate of 0.88 %.

Besides the previous rankings, we have found a work by Mishkin and Matas [251] where the authors work in the weights initialization of the CNN and replacing the softmax classifier with SVM, achieving a test error rate of 0.38 %, and another work by Alom et al. [4] where inception-recurrent CNNs were used to attain an error of 0.29 %.

Finally, in chapter 3 of this thesis we reviewed related works comprising the automatic design of convolutional neural networks. It is worth recalling that MetaQNN [10] had reported an error rate on MNIST of 0.44 %, and 0.32 % when using an ensemble of the best found neural networks, and DEvol [83] obtained an error rate of 0.6 %. Kramer [189] reported an error rate of 0.9 % when

Technique	Test Error Rate
HOPE+DNN with unsupervised learning features [409]	0.40 %
Deep convex net [87]	0.83 %
CDBN [208]	0.82 %
S-SC + linear SVM [396]	0.84 %
2-layer MP-DBM [119]	0.88 %
DNet-kNN [246]	0.94 %†
2-layer Boltzmann machine [312]	0.95 %
CEA-CNN, k=2 [31]	**0.24 %**
Batch-normalized maxout network-in-network [57]	**0.24 %**†
CNN with gated pooling function [206]	0.29 %
Inception-Recurrent CNN + LSUV + EVE [4]	0.29 %†
Monte Carlo Planning (RAVE4NN) [391]	0.31 %†
Recurrent CNN [212]	0.31 %
CNN with norm. layers and piecewise linear activation units [214]	0.31 %
CNN (5 conv, 3 dense) with full training [149]	0.32 %
MetaQNN (ensemble) [10]	0.32 %
Fractional max-pooling CNN with random overlapping [126]	0.32 %†
CNN with competitive multi-scale conv. filters [213]	0.33 %†
IEA-CNN [31]	0.34 %
Fast-learning shallow CNN [236]	0.37 %
CNN FitNet with LSUV initialization and SVM [251]	0.38 %
Deeply supervised CNN [207]	0.39 %
Convolutional kernel networks [227]	0.39 %
CNN with Multi-loss regularization [395]	0.42 %
MetaQNN [10]	0.44 %
CNN (3 conv maxout, 1 dense) with dropout [165]	0.45 %
Convolutional highway networks [341]	0.45 %
CNN (5 conv, 3 dense) with retraining [149]	0.46 %
Network-in-network [216]	0.47 %
CNN (3 conv, 1 dense), stochastic pooling [408]	0.49 %†
CNN (2 conv, 1 dense, ReLU) with dropout [378]	0.52 %
CNN, unsup pretraining [165]	0.53 %
CNN (2 conv, 1 dense, ReLU) with DropConnect [378]	0.57 %
SparseNet + SVM [192]	0.59 %
CNN (2 conv, 1 dense), unsup pretraining [297]	0.60 %
DEvol [83]	0.60 %†
CNN (2 conv, 2 dense) [296]	0.62 %
Boosted Gabor CNN [49]	0.68 %
CNN (2 conv, 1 dense) with L-BFGS [201]	0.69 %
Fastfood 1024/2048 CNN [398]	0.71 %
Feature Extractor + SVM [197]	0.83 %
Dual-hidden layer feedforward network [235]	0.87 %
Evolution of convolutional highway networks [189]	0.9 %
CNN LeNet-5 [203]	0.95 %

Table 5.3: *Side-by-side comparison of the most competitive (error rate < 1 %) results found in the state of the art for the MNIST database without data augmentation or preprocessing.*

evolving convolutional highway networks. More remarkably, Bochinski et al. [31] reported an error rate without data augmentation of 0.24 % using a CNN committee obtained from neuroevolution (against an error of 0.34 % with a single model), a result that would head the ranking.

A summary of all the reviewed works can be found in tables 5.2 and 5.3, which include works with and without data augmentation respectively. The upper sides of the tables show the performance of classical ML approaches, while the lower side displays the results of CNNs. When authors reported different results using the same technique, only the best is shown. Best achieved performances are boldfaced. Also, in order to achieve an exhaustive comparison, some works published in pre-print or code repositories, and therefore not subject to peer-review have been included, and are displayed in the tables along with the † symbol.

5.2.3 *Preprocessing*

In this thesis, we have not performed any further preprocessing or data augmentation of the MNIST database, using the training and test sets out-of-the-box. For this reason, in order to achieve a fair comparison the results obtained in this work will be compared with those reported in table 5.3.

5.2.4 *Encoding*

In this section we will discuss the encoding for both genetic algorithms and grammatical evolution. We will explain how the phenotype is built from the genotype in order to create suitable individuals.

5.2.4.1 *Genetic Algorithm*

The chromosome of the GA for the MNIST dataset consists of a 69-bit binary string using Gray encoding. A brief summary of the genotype's structure is shown in figure 5.4. Next, we will explain this structure with further detail, as well as how the genotype is converted to a phenotype. The chromosome encodes the network setup, being x the integer corresponding to the Gray binary. The first hyperparameter defines the input configuration:

- B: the batch size (2 bits), which can take values $B \in [25, 50, 100, 15]$ ($x = 0, 1, 2, 3$ respectively).

The following five hyperparameters define the setup of the convolutional layers:

- n_c: the number of convolutional layers (2 bits), computed as $n_c = 1 + x$, thus taking values between $n_c = 1$ and $n_c = 4$.

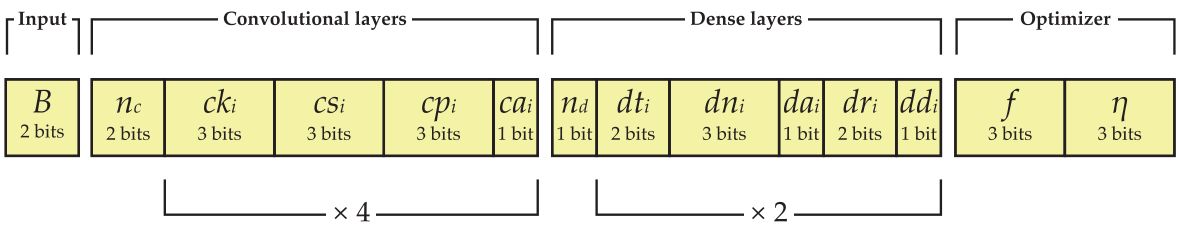


Figure 5.4: *Definition of the chromosome in the GA for the MNIST dataset.*

- ck_i: the number of kernels in the i-th convolutional layer (3 bits), computed as $ck_i = 2^{(x+1)}$, thus taking the values $ck_i \in [2, 4, 8, 16, 32, 64, 128, 256]$.
- cs_i: the kernel size of the i-th convolutional layer (3 bits), computed as $cs_i = 2 + x$, thus guaranteeing that the minimum value is $cs_i = 2$ and the maximum value is $cs_i = 9$. Squared kernels are enforced, so cs_i refers to both the number of rows and columns.
- cp_i: the pooling of the i-th convolutional layer (3 bits), computed as $cp_i = 1 + x$, thus guaranteeing that the minimum value is $cp_i = 1$ and the maximum value is $cp_i = 8$. If pooling size if 1, then it is equivalent to not doing pooling over the input. Squared pooling is enforced.
- ca_i: the activation function of the i-th convolutional layer (1 bit), which can be either ReLU ($x = 0$) or linear ($x = 1$).

Because there will be at most 4 convolutional layers, the chromosome repeats the genes for ck_i, cs_i, cp_i and ca_i four times. However, the network will only consider the setup for the first n_c layers, and ignore the rest. The following six hyperparameters define the setup of the dense layers:

- n_d: the number of dense layers (1 bit), computed as $n_d = 1 + x$, thus taking values between $n_d = 1$ and $n_d = 2$.
- dt_i: the type of the i-th dense layer (2 bits), which can be either recurrent ($x = 0$), LSTM ($x = 1$), GRU ($x = 2$) or feed-forward ($x = 3$).
- dn_i: the number of neurons in the i-th layer (3 bits), computed as $dn_i = 2^{(3+x)}$, thus taking the values $dn_i \in [8, 16, 32, 64, 128, 256, 512, 1024]$.
- da_i: the activation function of the neurons in the i-th layer (1 bit), which can be either ReLU ($x = 0$) or linear ($x = 1$).
- dr_i: the regularization applied to the weights of the i-th layer (2 bits), which which can be either none ($x = 0$), L1 ($x = 1$), L2 ($x = 2$) or L1+L2 ($x = 3$).
- dd_i: the dropout probability for the weights in the i-th layer (1 bit), which is computed as $dd_i = x/2$, thus taking the values $dd_i = 0$ (no dropout) or $dd_i = 0.5$.

Because there can be up to 2 dense layers, the chromosome repeats the genes for dt_i, dn_i, da_i, dr_i and dd_i twice. However, the network will only consider the hyperparameters for the first n_d layers.

Finally, the last two hyperparameters store the configuration of the optimization process:

- f: the optimizer or gradient descent update function (3 bits), which can be either SGD ($x = 0$), SGD with momentum ($x = 1$), SGD with Nesterov momentum ($x = 2$), AdaGrad ($x = 3$), AdaMax ($x = 4$), Adam ($x = 5$), AdaDelta ($x = 6$) or RMSProp ($x = 7$).
- η: the learning rate (3 bits), which can be either $1 \cdot 10^{-5}$ ($x = 0$), $5 \cdot 10^{-5}$ ($x = 1$), $1 \cdot 10^{-4}$ ($x = 2$), $5 \cdot 10^{-4}$ ($x = 3$), $1 \cdot 10^{-3}$ ($x = 4$), $5 \cdot 10^{-3}$ ($x = 5$), $1 \cdot 10^{-2}$ ($x = 6$) or $5 \cdot 10^{-2}$ ($x = 7$).

5.2.4.2 *Grammatical Evolution*

Figure 5.5 shows the definition of grammar used for generating individuals in the MNIST problem, in Backus-Naur Form (BNF). We can see how the grammar can generate phenotypes which are very similar to those encoded in the GA. However, in GE we have more freedom to specify a variable number of values for each hyperparameter. This is due to how the GA encoding worked: in order

```
<dnn>         ::= <input> <conv_lys> <dense_lys> <opt_setup>

<input>       ::= <batch_size>
<batch_size>  ::= 25 | 50 | 100 | 150

<conv_lys>    ::= <conv> | <conv> <conv> | <conv> <conv> <conv>
<conv>        ::= <n_kernels> <k_size> <act_fn> <pooling>
<n_kernels>   ::= 8 | 16 | 32 | 64 | 128 | 256
<k_size>      ::= 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<pooling>     ::= null | <p_size>
<p_size>      ::= 2 | 3 | 4 | 5 | 6

<dense_lys>   ::= <dense> | <dense> <dense> | <dense> <dense> <dense>
<dense>       ::= <d_type> <n_units> <act_fn> <reg_fn> <dropout_r>
<d_type>      ::= rnn | lstm | gru | feedforward
<n_units>     ::= 32 | 64 | 128 | 256 | 512 | 1024
<act_fn>      ::= relu | linear
<reg_fn>      ::= null | l1 | l2 | l1l2
<dropout_r>   ::= 0 | 0.5

<opt_setup>   ::= <opt_type> <learn_rate> <batch_size>
<opt_type>    ::= sgd | nesterov | momentum | adagrad | adamax | adam |
                  adadelta | rmsprop
<learn_rate>  ::= 5E-1 | 1E-1 | 5E-2 | 1E-2 | 5E-3 | 1E-3
```

Figure 5.5: *Definition of the grammar in Backus-Naur Form for the MNIST dataset.*

to avoid redundancy, we decided that each hyperparameter would have a set of values whose cardinality is always a power of 2. In GE, we no longer need to enforce this condition, so we can feel free to set any number of values; for instance, the number of units in the dense layers can take seven values, there are five possible values for the learning rate, six possible values for the number of kernels, etc. As a result, we have decided to remove values which are uncommon in the winning individuals of the GA (see section 5.2.6.1), thus reducing the search space. For the same reason, the maximum number of convolutional layers has been reduced to three.

Also, while the GA had some redundancy in the number of convolutional and dense layers (if not all the layers were used, then the chromosome contained genetic information that was ignored when building the phenotype), this redundancy is naturally removed when using GE. As a result, we expect GE to converge faster than GAs.

5.2.5 Experimental Setup

In this section we will describe the experimental setup, both for the GA and GE.

The fitness function to maximize is the accuracy of the convolutional neural network whose architecture is defined by the individual phenotype. It should be noticed that this is equivalent to minimizing the error rate, which is the metric in which most state-of-the-art works are reported. Because the CNN weights are randomly initialized, the fitness presents some stochasticity, and the same individual will likely obtain different fitness values when evaluated more than once.

Also, because fitness evaluations are expensive, as they require to train and evaluate the CNN, we have reduced the training time by running only 5 training epochs, and using only a randomly-

chosen 50 % sample of the training set in each epoch. The obtained result is used as a proxy of a more exhaustive training, taking only a 8.3 % of the time required to train a network with the whole training set over 30 epochs.

The parameterization of the genetic algorithm and grammatical evolution has been already described in sections 4.5.7 and 4.5.8 respectively.

In order to obtain significant results and avoid bias caused by the random initialization of the initial population, each experiment will be repeated 10 times. As specified in the previous chapter, a run will stop after 100 generations, or 30 generations without improvements in the best individual.

It is worth recalling that we have a hall-of-fame of size 50 for each technique. The best 50 individuals across all runs will be stored in the hall-of-fame. This will enable us to check not only the best individual, but up to the best 50, and will also allow to build an ensemble of CNNs.

5.2.6 Results

In this section we will discuss the results obtained for both the GA and GE. We will first describe the results of the optimization process using each evolutionary computation technique, and later explain how competitive individuals are obtained by thoroughly training the fittest individuals. We will also discuss the performance of ensembles built by combining several of the top-performing convolutional neural networks.

5.2.6.1 Genetic Algorithm

Figure 5.6 shows how the median fitness has evolved over time for one of the 10 runs of the GA. In this figure, we can see how the fitness grows rapidly in the first ten generations. Then, the fitness rises very slowly for the next 50 generations until convergence. This pattern is consistently shown across all runs of the GA.

The fitness values and architectures (phenotypes) for the top-10 individuals in the hall-of-fame are shown in table 5.4. The symbols 'c' and 'd' span the convolutional and the dense layers respectively, in order to ease their identification. As it can be seen, the best individual has a fitness of

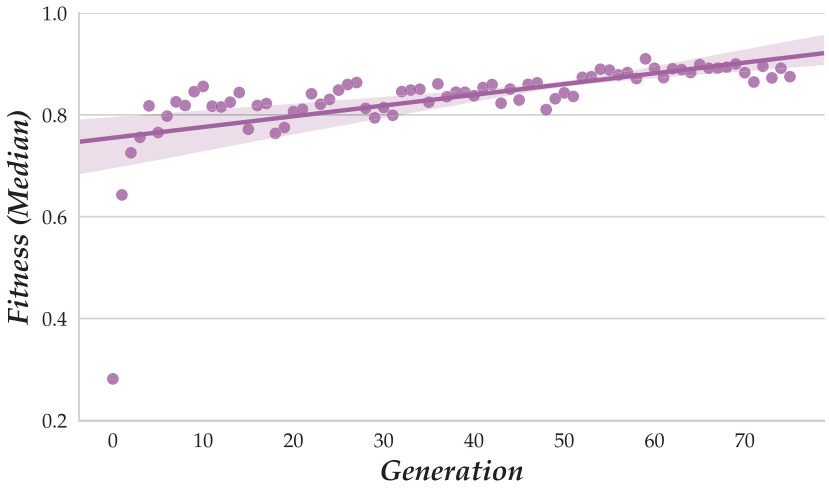


Figure 5.6: *Evolution of the median fitness in one run of the GA with the MNIST dataset.*

#	Fitness		Architecture				
1	0.9932		$B = 25$	$f =$ AdaMax	$\eta = 0.001$		
		c	$ck_1 = 64$	$cs_1 = 6$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 128$	$cs_2 = 5$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 128$	$cs_3 = 8$	$cp_3 = 1$	$ca_3 =$ ReLU	
		d	$dt_1 =$ feed-forward	$dn_1 = 128$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
2	0.9916		$B = 50$	$f =$ Nesterov	$\eta = 0.001$		
		c	$ck_1 = 64$	$cs_1 = 4$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 128$	$cs_2 = 4$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 256$	$cs_3 = 4$	$cp_3 = 2$	$ca_3 =$ ReLU	
		d	$dt_1 =$ feed-forward	$dn_1 = 128$	$dd_1 = 0.5$	$da_1 =$ linear	$dr_1 =$ none
		d	$dt_2 =$ feed-forward	$dn_2 = 512$	$dd_2 = 0$	$da_2 =$ linear	$dr_2 =$ none
3	0.9915		$B = 150$	$f =$ Nesterov	$\eta = 0.05$		
		c	$ck_1 = 128$	$cs_1 = 5$	$cp_1 = 2$	$ca_1 =$ ReLU	
		c	$ck_2 = 256$	$cs_2 = 5$	$cp_2 = 4$	$ca_2 =$ linear	
		d	$dt_1 =$ feed-forward	$dn_1 = 32$	$dd_1 = 0$	$da_1 =$ linear	$dr_1 =$ none
4	0.9915		$B = 50$	$f =$ RMSProp	$\eta = 0.001$		
		c	$ck_1 = 32$	$cs_1 = 3$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 64$	$cs_2 = 5$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 32$	$cs_3 = 8$	$cp_3 = 1$	$ca_3 =$ ReLU	
		d	$dt_1 =$ feed-forward	$dn_1 = 256$	$dd_1 = 0.5$	$da_1 =$ linear	$dr_1 =$ none
5	0.9915		$B = 25$	$f =$ Momentum	$\eta = 0.01$		
		c	$ck_1 = 64$	$cs_1 = 4$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 256$	$cs_2 = 8$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 128$	$cs_3 = 7$	$cp_3 = 3$	$ca_3 =$ linear	
		d	$dt_1 =$ feed-forward	$dn_1 = 128$	$dd_1 = 0$	$da_1 =$ linear	$dr_1 =$ none
6	0.9915		$B = 50$	$f =$ AdaMax	$\eta = 0.005$		
		c	$ck_1 = 8$	$cs_1 = 4$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 128$	$cs_2 = 6$	$cp_2 = 5$	$ca_2 =$ linear	
		d	$dt_1 =$ feed-forward	$dn_1 = 512$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
7	0.9914		$B = 25$	$f =$ AdaGrad	$\eta = 0.01$		
		c	$ck_1 = 32$	$cs_1 = 6$	$cp_1 = 2$	$ca_1 =$ ReLU	
		c	$ck_2 = 128$	$cs_2 = 4$	$cp_2 = 2$	$ca_2 =$ linear	
		d	$dt_1 =$ rnn	$dn_1 = 32$	$dd_1 = 0$	$da_1 =$ linear	$dr_1 =$ l2
		d	$dt_2 =$ feed-forward	$dn_2 = 1024$	$dd_2 = 0.5$	$da_2 =$ ReLU	$dr_2 =$ none
8	0.9914		$B = 50$	$f =$ AdaGrad	$\eta = 0.01$		
		c	$ck_1 = 128$	$cs_1 = 7$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 256$	$cs_2 = 2$	$cp_2 = 2$	$ca_2 =$ ReLU	
		c	$ck_3 = 128$	$cs_3 = 9$	$cp_3 = 2$	$ca_3 =$ ReLU	
		d	$dt_1 =$ feed-forward	$dn_1 = 32$	$dd_1 = 0$	$da_1 =$ linear	$dr_1 =$ none
		d	$dt_2 =$ feed-forward	$dn_2 = 64$	$dd_2 = 0$	$da_2 =$ linear	$dr_2 =$ none
9	0.9914		$B = 25$	$f =$ AdaGrad	$\eta = 0.01$		
		c	$ck_1 = 64$	$cs_1 = 8$	$cp_1 = 2$	$ca_1 =$ ReLU	
		c	$ck_2 = 256$	$cs_2 = 5$	$cp_2 = 3$	$ca_2 =$ ReLU	
		d	$dt_1 =$ feed-forward	$dn_1 = 256$	$dd_1 = 0.5$	$da_1 =$ ReLU	$dr_1 =$ none
10	0.9913		$B = 50$	$f =$ SGD	$\eta = 0.05$		
		c	$ck_1 = 64$	$cs_1 = 7$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 128$	$cs_2 = 7$	$cp_2 = 4$	$ca_2 =$ linear	
		d	$dt_1 =$ feed-forward	$dn_1 = 256$	$dd_1 = 0$	$da_1 =$ linear	$dr_1 =$ none
		d	$dt_2 =$ feed-forward	$dn_2 = 512$	$dd_2 = 0.5$	$da_2 =$ linear	$dr_2 =$ none

Table 5.4: *Architecture and fitness of the top 10 individuals in the hall-of-fame for the GA in the MNIST dataset.*

0.9932, which is already significantly larger than the fitness of the second individual, of 0.9916. The following individuals' fitnesss barely vary, going as low as 0.9913.

More interestingly, we can look for some meaning in the architectures of these individuals. As shown in the table, the topologies differ among them, an effect that can be attributed to two causes: (1) the use of niching to preserve genetic diversity and (2) the execution of 10 different runs with different initial populations. Nevertheless, some common patterns arise when taking a closer look at some of the values. For instance, none of the top-10 individuals has neither one nor four convolutional layers, resulting in all of them having either two or three. This can happen because one layer is insufficient to extract meaningful features from the data, and four layers may be too much given the small size of the input.

We can also see that only one individual applies L2 regularization to one of its dense layers. This would mean that L1 or L2 regularization is not useful for this domain. Not the same behavior is found in dropout, a different form of regularization, as some of these architectures involve dropout of 50 % in one of their dense layers.

Another common pattern that can be found is the lack of recurrent layers along the fully connected layers, which does not seem to be required for properly solving the problem at hand. This makes sense as the input is fairly small and does not involve a temporal dimension.

Also, all individuals have at least one of its layers with a non-linear activation function (ReLU). This seems reasonable, as it is possible that a low error rate cannot be attained using only linear transformations of the input data.

Regarding the learning rate, it is never smaller than $\eta = 1 \cdot 10^{-3}$, even if the encoding allows values as low as $\eta = 1 \cdot 10^{-5}$. It is quite likely that such small learning rates be unable to provide an accurate model in as few as five epochs.

It is worth recalling that these fitness values are obtained using only 5 training epochs and a random 50 % sample of the training set. This simplification was made in order to reduce the cost of fitness computation. Of course, we expect these results to improve significantly after each topology

#	Mean	Std. Dev.	Median	Minimum	Maximum
1	0.5130	0.037290	0.515	0.41	0.58
2	0.5380	0.031221	0.535	0.49	0.60
3	0.5485	0.035582	0.545	0.49	0.65
4	0.5875	0.045291	0.580	0.50	0.67
5	0.6115	0.049340	0.620	0.46	0.67
6	0.5010	0.031772	0.505	0.45	0.56
7	0.5760	0.035004	0.575	0.50	0.64
8	0.6085	0.038289	0.605	0.53	0.70
9	0.4795	0.031702	0.485	0.40	0.53
10	0.6090	0.032428	0.600	0.57	0.67
11	0.5600	0.040262	0.570	0.49	0.63
12	0.5850	0.034259	0.570	0.54	0.65
13	0.7095	0.045361	0.720	0.60	0.78
14	0.6045	0.037902	0.615	0.54	0.66
15	0.5780	0.036216	0.580	0.47	0.64
16	0.6265	0.056965	0.625	0.51	0.72
17	0.6010	0.035968	0.600	0.51	0.67
18	0.6815	0.033604	0.685	0.62	0.73
19	0.4805	0.036343	0.485	0.40	0.53
20	0.5930	0.037290	0.585	0.52	0.66

Table 5.5: *Summary of errors (in %) of the best 20 GA individuals after full training in MNIST.*

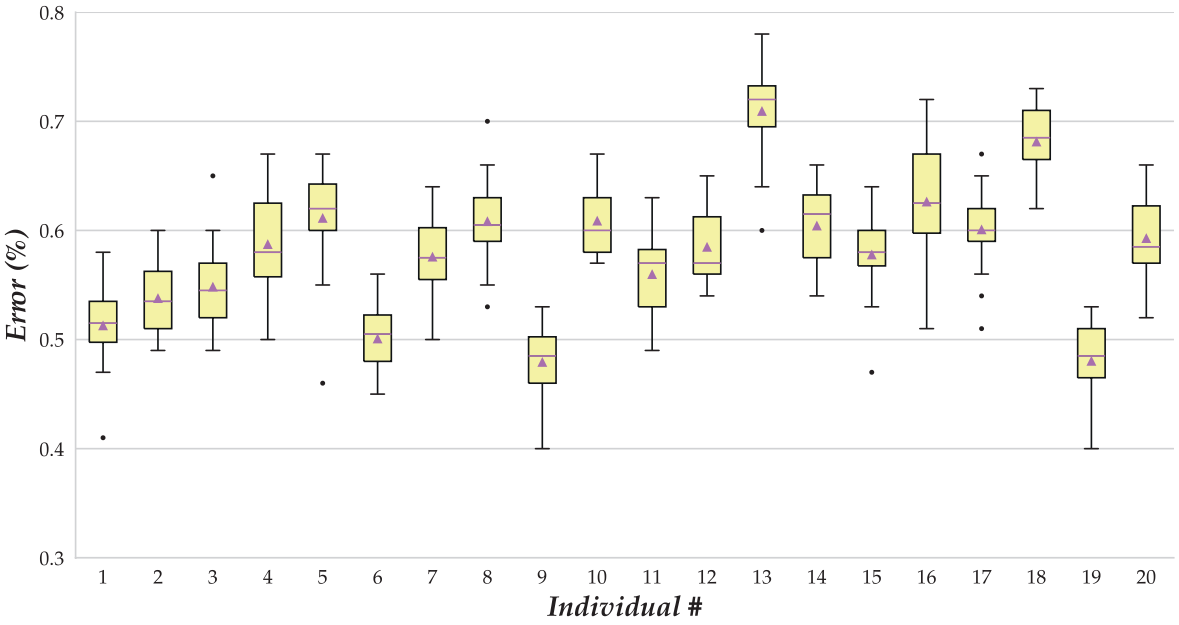


Figure 5.7: *Boxplot showing the distribution of errors of the best 20 GA individuals after full training in MNIST.*

is trained during more epochs and with the full training set. For this reason, we have decided to retrain each of the top-20 individuals in the hall-of-fame (from whom only the top-10 are described in table 5.4) 20 times, using 30 epochs in each run and using the full training set. In each run, the architecture is trained from scratch and evaluated over the test set.

Table 5.5 provides a statistical summary of the errors' distribution obtained for each of these individuals expressed as a percentage, including the mean, standard deviation, minimum, median and maximum. Also, figure 5.7 depicts this distribution as a boxplot, including the mean, which is shown as a small triangle within each box. Notice that we will not refer to this value as "fitness" anymore, as it was not directly optimized by the genetic algorithm.

From these data, we can extract two interesting conclusions. First, as we had assumed earlier, the accuracy is significantly larger than the fitness value, an improvement achieved due to the use of the whole training set and a larger number of training epochs. In fact, the minimum accuracy for each individual is still higher than the fitness value when using 5 epochs and sampling in all cases. Secondly, we find that there is not a direct correlation between the position in the hall-of-fame and the accuracy, i.e., not always the fittest individuals in the GA are showing the best performance in terms of accuracy. As a result, we can see how individuals 9th and 19th outperform the others, having a very similar behavior (as it can be seen in table 5.5).

The minimum error found so far is 0.40 %, equivalent to an accuracy of 99.60 %. Comparing these results with the ones shown in the bottom side of table 5.3 (our benchmark for fair comparison), this would place our best individual in the 17^{th} position of the ranking, or the 12^{th} position if only peer-reviewed papers are considered.

Finally, we will evaluate how ensembles or committees of convolutional neural networks behave. Our decision to use ensembles is based on the following facts and hypotheses:

- To optimize the topologies, we have run 10 executions of a population-based evolutionary algorithm, achieving a large population of individuals to choose from.

- Because we have implemented a niching strategy to preserve the genetic diversity, resulting individuals are significantly different among themselves, except for those hyperparameters clearly leading to better results which are similar.
- When we have several classifiers which are diverse enough, combining them may enable some to cover the classification errors made from others. As a result, the accuracy of the classifiers combined may be higher than the accuracy of the best.

Because we do not know how ensembles will perform or which is the optimal number of classifiers to involve, we will follow the next steps in order to find the best committee:

1. We have serialized the best model found for each of the 20 individuals previously discussed. By "serialize", we mean we have stored all the network parameters in a file, so an identical network can be rebuilt from these parameters, enabling reproducibility of the results.
2. We have sorted the 20 models in increasing order of error rate. After this sorting process, the first individual will have an error rate of 0.40 % and the last individual of 0.62 %.
3. We have built ensembles by increasingly adding a new model to the previous ensemble. In other words, we start with the best individual model, and build one ensemble by adding the second-best model, and so on and so forth until we have tested 20 different ensembles. Finally, we can decide which one performs best.
4. The ensembles work as follows: each classifier will produce its own prediction for the test set. Then, all the predictions will be aggregated by majority-voting.

The error rate of the 20 ensembles is shown in figure 5.8, in the purple line. Yellow diamonds in the figure shows, just for reference, the average error rate of all the models included in the ensemble. Because classifiers are sorted by ascending error, this average increases as more classifiers are added.

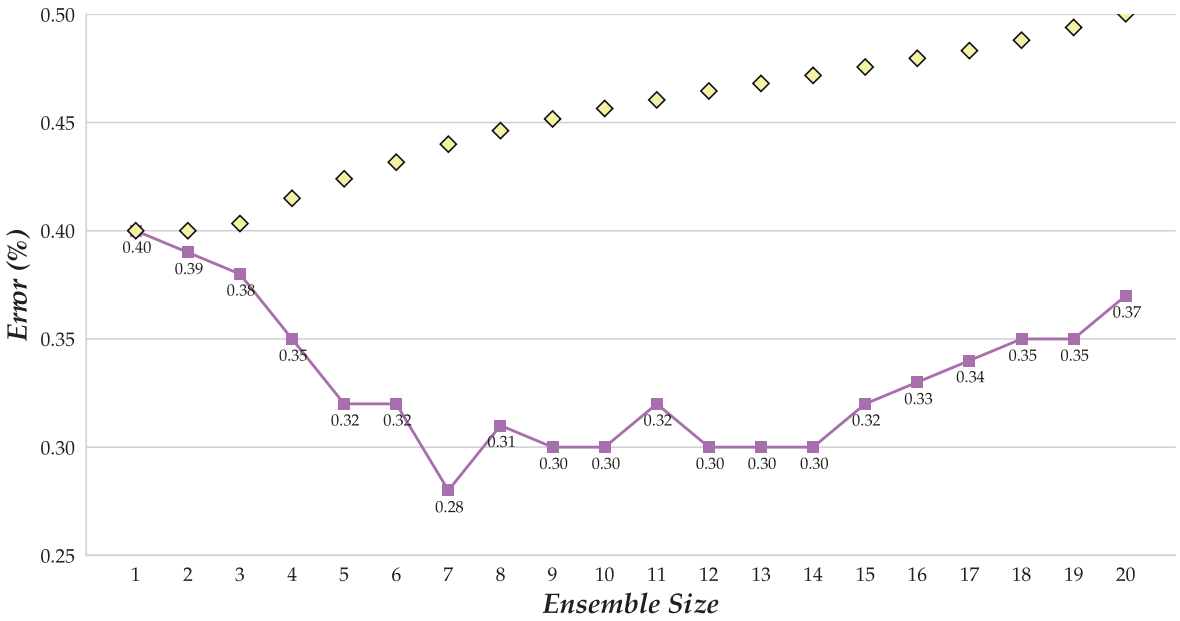


Figure 5.8: *Evolution of the error rate of the incremental ensembles using the best 20 individuals from the GA with MNIST.*

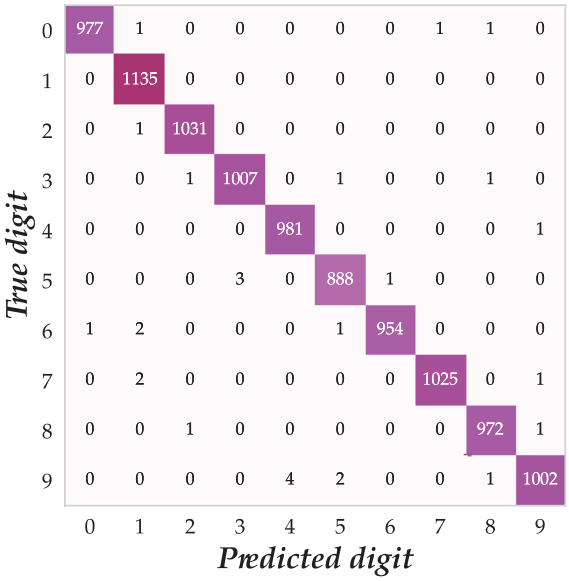


Figure 5.9: *Confusion matrix of the best ensemble using the GA individuals with MNIST.*

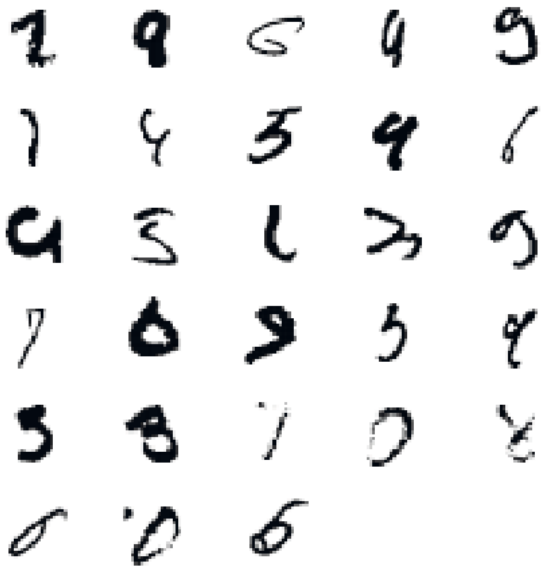


Figure 5.10: *The 28 misclassified images in the MNIST test set with the best ensemble obtained with GA.*

Interestingly, it can be seen how the test error ratte attained by an ensemble is significantly better than the average error rate of its components in all cases without exception. It can be seen how the introduction of a few models decreases the error down to 0.28 % (when 7 classifiers are used), and then stabilizes for a while until it starts to increase again when using more than 14 classifiers). This can be explained as the latter classifiers perform worst, thus having a negative effect in the global performance of the ensemble.

The best committee found in our research, the one involving the best 7 classifiers, classifies the MNIST test set with an error rate of 0.28 %, or equivalently produces an accuracy of 99.72 %. Compared with the ranking in table 5.3, our model would be placed in the third position or, if only peer-reviewed papers are considered, it would reach the second position in the ranking. As a result, as we expected, ensembles outperformed individual models significantly, allowing the performance to climb a large number of positions in the ranking.

Finally, an error rate of 0.28 % over a test set of 10000 samples results in 28 incorrectly classified samples. The confusion matrix of the best model is shown in figure 5.9. The interpretation of this confusion matrix is hugely interesting: we can see how the most frequent error involves the number '9' being classified as a '4'. Some other common mistakes involves recognizing a '3' instead of a '5' or a '1' instead of a '7'. Mixing up these numbers might be acceptable even for humans in the case that some of these were poorly written.

To be more specific, the 28 images that were misclassified are depicted in figure 5.10. We can see how those manuscript digits are indeed very unclear or badly written, and therefore can lead to confusion very easily. For example, the fourth image in the first row could be either a '4' or a '9' and the third image in the second row could be a '3' or a '5'. In some cases, some digits seem to be poorly digitalized, such as the third image in the fifth row, which seems like a '7' that became more similar to a '1' after passing through the scanner. Some other digits are really hard to identify, such as the second from the last row. We should acknowledge that it can difficult even for humans to properly recognize these numbers.

5.2.6.2 Grammatical Evolution

Figure 5.11 shows how the median fitness has evolved over time for one of the 10 runs of the GE. As it happened with the GA, the fitness grows quite rapidly in the first generations. However, this fast improvement is even more clear in GE, as the median fitness is better than in the GA in early generations. Then, the fitness rises slowly over the course of the generations, eventually converging.

The fitness values and architectures (phenotypes) for the top-10 individuals in the hall-of-fame are shown in table 5.6. When compared with the winning architectures of the GA, we can see in a first glance that in GE, the fitness is consistently better. This makes sense as we designed an encoding that removed redundancy, thus reducing the search space. In this case, the best individual has a fitness of 0.9934, whereas the best individual in the GA had a fitness of 0.9932. However, this value was an outlier in the GA, as the second-best had a fitness of only 99.16. In GE, the best 20 individuals all have a higher fitness than the second-best individual from the GA.

Again, it is worth looking into the architecture of the best individuals. As in the case of the GA, the topologies differ among them, though showing some common patterns. Nevertheless, it can be seen that some architectures are very similar, e.g., individuals 1, 2, 4 and 7 share the same number of kernels, kernel sizes and pooling setup in the three convolutional layers. The reason for this similarity may be due a good individual being slightly mutated across generations, with a few impact in its fitness, and thus inserted several times in the hall-of-fame.

Again, no individual has only one convolutional layer, which seems to reinforce the idea that one layer is insufficient to build useful features from raw data, already present in the case of the GA. However, unlike in the case of the GA, all individuals except for one have three convolutional layers, the other one having two. Thus, it seems that three convolutional layers are the most convenient setup for achieving the best results in the MNIST database.

Also regarding convolutional setup, all individuals have the maximum number of kernels (256) and a non-linear activation function (ReLU) in at least one of their layers. This behavior is consistent with the GA, and again, can be due linear transformations not being sufficient to extract valid features from raw data. While many individuals implement pooling, they do not apply this reduction in more than one layer either, maybe because otherwise the network structure would be invalid.

As in the GA, there are not recurrent layers, which seems consistent with the fact that data does not present a temporal dimension. All individuals contain only one dense layer, except for one that comprises two layers. The number of neurons in the feed-forward layers is always larger than 128.

L1 or L2 regularization is uncommon, with only one individual implementing L2 in one of its layers. Dropout is found more often, thus proving useful in certain cases.

Finally, all the optimizers in the best 10 individuals are either *adagrad*, *adamax* or *adadelta*. Learning rate is always 0.5 for *adadelta*, 0.001 for *adamax*, and $0.005 - 0.01$ for *adamax*.

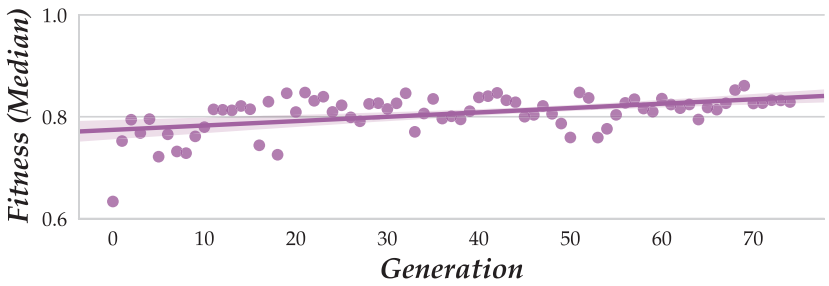


Figure 5.11: *Evolution of the median fitness in one run of the GE with the MNIST dataset.*

#	Fitness		Architecture				
1	0.9934		$B = 50$	$f =$ AdaGrad	$\eta = 0.005$		
		c	$ck_1 = 64$	$cs_1 = 4$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 256$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 256$	$cs_3 = 7$	$cp_3 = 6$	$ca_3 =$ linear	
		d	$dt_1 =$ feed-forward	$dn_1 = 128$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
2	0.9929		$B = 50$	$f =$ AdaGrad	$\eta = 0.005$		
		c	$ck_1 = 64$	$cs_1 = 4$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 256$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 256$	$cs_3 = 7$	$cp_3 = 6$	$ca_3 =$ linear	
		d	$dt_1 =$ feed-forward	$dn_1 = 128$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
3	0.9928		$B = 100$	$f =$ AdaGrad	$\eta = 0.005$		
		c	$ck_1 = 256$	$cs_1 = 7$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 128$	$cs_2 = 7$	$cp_2 = 4$	$ca_2 =$ ReLU	
		d	$dt_1 =$ feed-forward	$dn_1 = 256$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ L2
		d	$dt_2 =$ feed-forward	$dn_2 = 1024$	$dd_2 = 0.5$	$da_2 =$ linear	$dr_2 =$ none
4	0.9927		$B = 100$	$f =$ AdaMax	$\eta = 0.001$		
		c	$ck_1 = 64$	$cs_1 = 4$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 256$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 256$	$cs_3 = 7$	$cp_3 = 6$	$ca_3 =$ linear	
		d	$dt_1 =$ feed-forward	$dn_1 = 1024$	$dd_1 = 0.5$	$da_1 =$ ReLU	$dr_1 =$ none
5	0.9926		$B = 25$	$f =$ AdaDelta	$\eta = 0.5$		
		c	$ck_1 = 8$	$cs_1 = 3$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 256$	$cs_2 = 9$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 32$	$cs_3 = 8$	$cp_3 = 1$	$ca_3 =$ ReLU	
		d	$dt_1 =$ feed-forward	$dn_1 = 1024$	$dd_1 = 0.5$	$da_1 =$ ReLU	$dr_1 =$ none
6	0.9924		$B = 50$	$f =$ AdaGrad	$\eta = 0.01$		
		c	$ck_1 = 64$	$cs_1 = 4$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 8$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 256$	$cs_3 = 7$	$cp_3 = 6$	$ca_3 =$ linear	
		d	$dt_1 =$ feed-forward	$dn_1 = 512$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
7	0.9924		$B = 100$	$f =$ AdaGrad	$\eta = 0.005$		
		c	$ck_1 = 64$	$cs_1 = 4$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 256$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 256$	$cs_3 = 7$	$cp_3 = 6$	$ca_3 =$ linear	
		d	$dt_1 =$ feed-forward	$dn_1 = 128$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
8	0.9922		$B = 50$	$f =$ AdaDelta	$\eta = 0.5$		
		c	$ck_1 = 64$	$cs_1 = 6$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 256$	$cs_2 = 8$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 32$	$cs_3 = 4$	$cp_3 = 4$	$ca_3 =$ ReLU	
		d	$dt_1 =$ feed-forward	$dn_1 = 256$	$dd_1 = 0$	$da_1 =$ linear	$dr_1 =$ none
9	0.9921		$B = 25$	$f =$ AdaDelta	$\eta = 0.5$		
		c	$ck_1 = 64$	$cs_1 = 5$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 256$	$cs_2 = 8$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 32$	$cs_3 = 4$	$cp_3 = 4$	$ca_3 =$ ReLU	
		d	$dt_1 =$ feed-forward	$dn_1 = 256$	$dd_1 = 0.5$	$da_1 =$ linear	$dr_1 =$ none
10	0.9921		$B = 50$	$f =$ AdaGrad	$\eta = 0.01$		
		c	$ck_1 = 64$	$cs_1 = 4$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 8$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 256$	$cs_3 = 7$	$cp_3 = 6$	$ca_3 =$ linear	
		d	$dt_1 =$ ReLU	$dn_1 = 128$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none

Table 5.6: *Architecture and fitness of the top 10 individuals in the hall-of-fame for GE in the MNIST dataset.*

#	Mean	Std. Dev.	Median	Minimum	Maximum
1	0.4680	0.026675	0.470	0.41	0.51
2	0.4765	0.032650	0.470	0.42	0.53
3	0.5310	0.040510	0.535	0.47	0.60
4	0.4300	0.025752	0.420	0.37	0.49
5	0.5545	0.030517	0.555	0.47	0.61
6	0.5265	0.054219	0.520	0.42	0.65
7	0.4615	0.032971	0.465	0.40	0.51
8	0.5830	0.030279	0.580	0.50	0.63
9	0.5530	0.037850	0.550	0.48	0.62
10	0.5330	0.031473	0.535	0.48	0.59
11	0.4985	0.030997	0.510	0.44	0.55
12	0.4815	0.041584	0.480	0.42	0.58
13	0.5090	0.046668	0.495	0.42	0.57
14	0.5680	0.049161	0.550	0.50	0.66
15	0.5095	0.021879	0.500	0.48	0.55
16	0.5245	0.105555	0.500	0.45	0.93
17	0.5525	0.042904	0.555	0.48	0.65
18	0.6015	0.040036	0.610	0.52	0.66
19	0.5100	0.028470	0.510	0.46	0.58
20	0.5525	0.032098	0.550	0.48	0.60

Table 5.7: *Summary of errors (in %) of the best 20 GE individuals after full training in MNIST.*

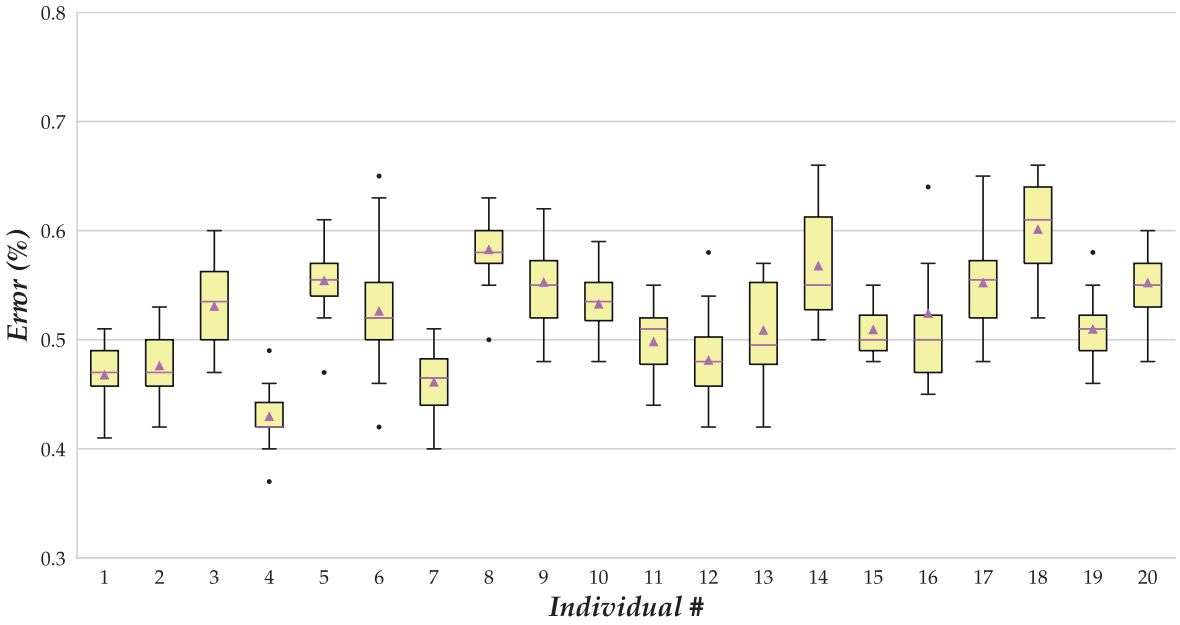


Figure 5.12: *Boxplot showing the distribution of errors of the best 20 GE individuals after full training in MNIST.*

As we did with the GA, we have retrained the top-20 individuals in the hall-of-fame using 30 epochs with the whole training set. A statistical summary of the performance after 20 runs is shown in table 5.7. Also, the error distribution for each individual is depicted as a boxplot in figure 5.12.

Again, we can see how results for each individual are consistently better than when using GAs. For example, there are test error rates smaller than 0.4, whereas no error rate is higher than 0.7 for any of the top-20 topologies. The average and median performance is also better when

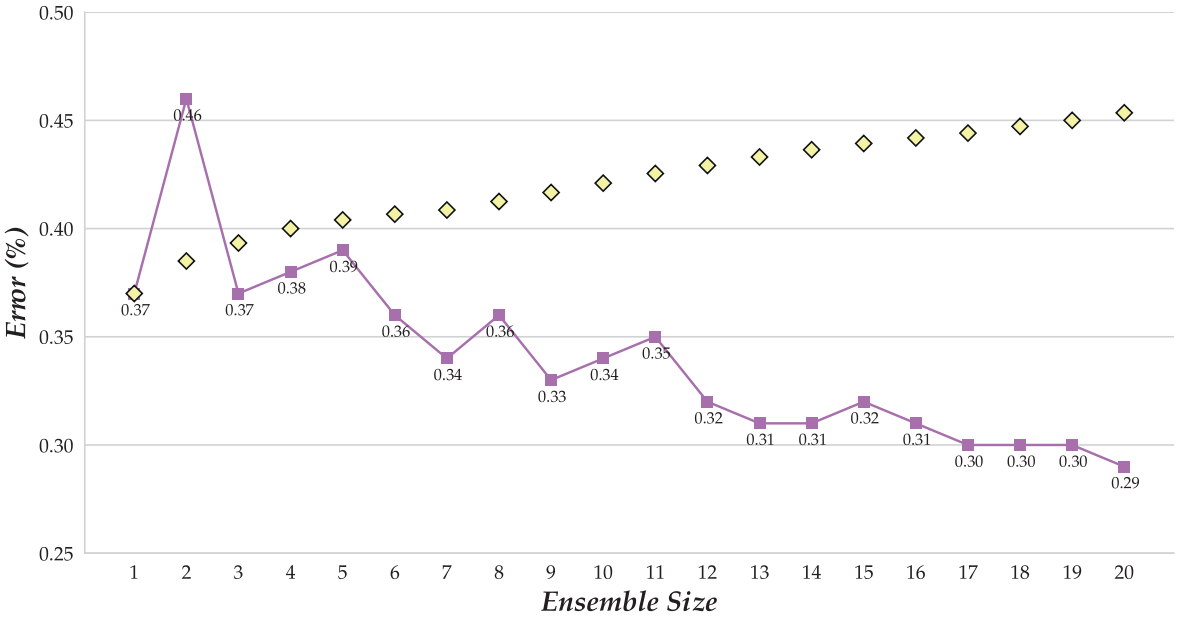


Figure 5.13: *Evolution of the error rate of the incremental ensembles using the best 20 individuals from the GE with MNIST.*

using GE over GAs. We attribute this improvement in the performance to the fact that GE removes redundancy and provides higher flexibility to remove some of the uncommon values for certain hyperparameters of the topology, thus reducing the search space.

After carrying out a full training of the top-20 topologies, we have followed the same approach that we used previously in the GA to build committees: we have serialized the best model for each topology, sorted these models by ascending error, and built ensembles by adding one model at a time. The resulting error for each ensemble is shown in figure 5.13, depicted as a purple line. As in the GA, the yellow diamonds show the average error of the models forming the ensemble, if they were used individually.

In most cases, the performance of the ensemble is better than the average performance of all the CNNs involved. The only exception is the 2-models ensemble, where the error is significantly larger than the average of the error of those two models. From there, the error keeps descending and we find the peak accuracy with the committee involving 20 models.

It is remarkable that, even when the performance of all individual models are better than in the GA, the ensembles perform slightly worst. A reasonable cause for is that the variability between the different models has turned out to be much smaller than in the GA, as we already saw in table 5.6. In fact, this variability is reduced when sorting the models by ascending error; e.g., looking at table 5.5, first model 4, then model 7, then model 1, and so on and so forth. These models have very similar topologies, and because CNNs in the ensemble are quite similar, the performance is affected negatively. This is consistent with our working hypothesis that committees would benefit from a larger diversity in their individuals.

The reason for GE having less diversity than the GA is twofold. First, genomes are shorter in GE, and thus the mutation rate of 1.5 % affects less genes than in the GA, thus mutation in the GE has a larger effect in the phenotype than in the GA. Second, in GE we have set a crossover rate of 70 %, meaning that 30 % of the times parents are introduced in the new population instead of offspring, thus reducing variability.

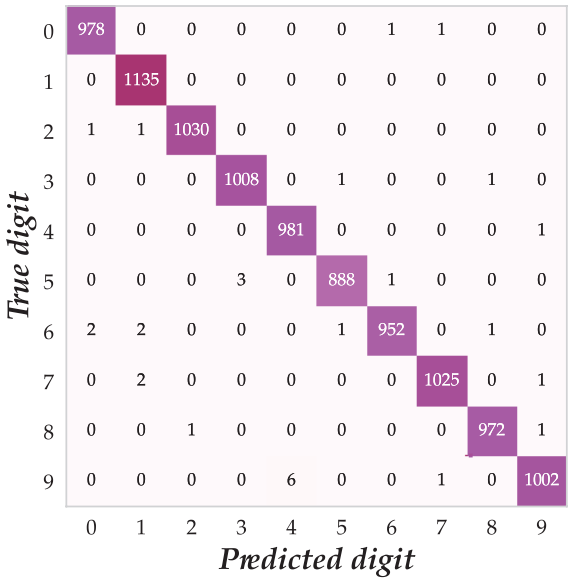


Figure 5.14: *Confusion matrix of the best ensemble using the GE individuals with MNIST.*

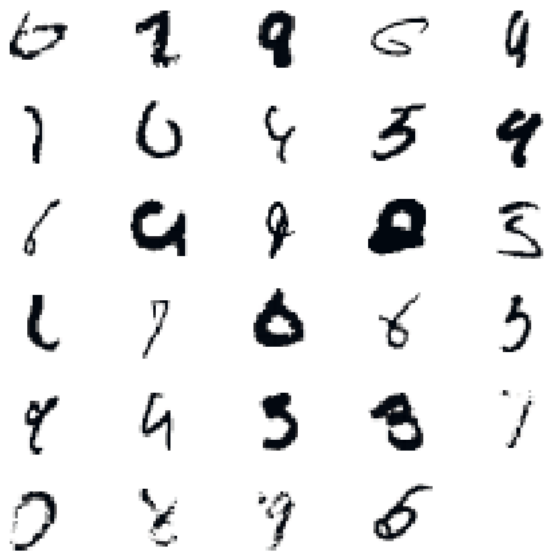


Figure 5.15: *The 29 misclassified images in the MNIST test set with the best ensemble obtained with GE.*

The best-performing model is, as we saw before, the committee comprising the 20 models resulting from fully training the top-20 topologies in the hall-of-fame. An error of 0.29 % translates into 29 misclassified digits. Figure 5.14 shows the confusion matrix for the MNIST dataset with the best committee. It can be seen how the main error involves six digits '9' classified as the digit '4', followed by three digits '5' classified as '3's. The whole set of misclassified digits can be found in figure 5.15. It is noticeable how some '9's are drawn with an open circle in the top, resembling a '4'. Also, some '6's can be easily mixed up with '0's, and in general, it can be acknowledged that those digits are poorly written and hard to recognize even for humans. In the end, the best result would be placed just below the performance obtained by the GA in the ranking.

5.3 EMNIST

EMNIST (Extended MNIST) database has been introduced in 2017 by Cohen et al. [73] and consists of a set of handwritten characters (both digits and letters). This dataset shares structure with the MNIST dataset described in the previous section.

Figure 5.16 shows ten samples for each letter in the EMNIST dataset, including both uppercase and lowercase variants, and two samples for each digit (in the last two columns).

The choice of this dataset is twofold: first, it is very recent (March 2017), and there are few-to-none published works benchmarking different techniques with this dataset. Secondly, and more interestingly, in this case we will not use GAs or GE to obtain the best CNN topologies for this work. Instead, we will reuse the architectures obtained in the previous section, when optimizing for the MNIST dataset, and will learn the weights of each architecture from scratch.

We hypothesize that these architectures must provide accurate classifications, since the data structure is equivalent and the domain is quite similar. Even if better results could be obtained by searching for optimal topologies for this dataset, reusing the previous ones should be a good proxy.

Figure 5.16: *Samples of all letters and digits in the EMNIST dataset.*

5.3.1 Acquisition

EMNIST database is derived from NIST Special Database 19 [267], which contains NIST's (National Institute of Standards and Technology of the US) entire corpus of training materials for handprinted document and character recognition. It contains over 810,000 isolated characters from 3,699 writers [131] who filled a form (see figure 5.17). These characters are labelled after manually checking.

Authors releasing EMNIST admit that in the past years, deep learning and convolutional neural networks have allowed scientists to achieve accuracies over 99.7 % in the MNIST dataset, stating that at that point "*the dataset labeling can be called into question*" [73]. For this reason, they suggest that MNIST has become a non-challenging benchmark.

Even though NIST Special Database 19, from which MNIST was extracted, was available since 1995, it has remained mostly unused as it is difficult to access and challenging to use in modern computers because of the way it was stored. Recently, in 2016, NIST has released a second edition of this database [131] which is easier to access.

Authors have performed a similar processing than the MNIST database, in order to make both compliant in terms of structure. The result is a dataset that contains more instances than MNIST, includes letters apart from digits, and thus is a more challenging benchmark for evaluating the performance of character recognition systems. This processing comprises the next steps: (1) original images in the NIST Special Database 19 are stored as 128x128-pixels BW images, (2) a Gaussian blur filter with $\sigma = 1$ is applied to soften the edges, (3) blank padding is removed, reducing the image to the region of interest (the actual digit), (4) the image is then centered in a square image while preserving the aspect ratio, padding it with a 2-pixels border, and (5) the image is downsampled to 28x28 pixels using bi-cubic interpolation. As a result, each instance in the EMNIST database is a 28x28-pixels grayscale image, where each pixel is a number between 0 and 255.

NIST SD 19 has two different labeling schemata, which have been ported to the EMNIST dataset:

- *By_Class*: in this schema classes are digits [0-9], lowercase letters [a-z] and uppercase letters [A-Z]. Thus, there are 62 different classes.
- *By_Merge*: this schema addresses the fact that some letters are quite similar in their lowercase and uppercase variants, thus both classes can be fused. In particular, these letter are 'c', 'i', 'j', 'k', 'l', 'm', 'o', 'p', 's', 'u', 'v', 'w', 'x', 'y' and 'z'. This schema contains 47 classes.

Besides these two datasets, EMNIST has generated additional datasets:

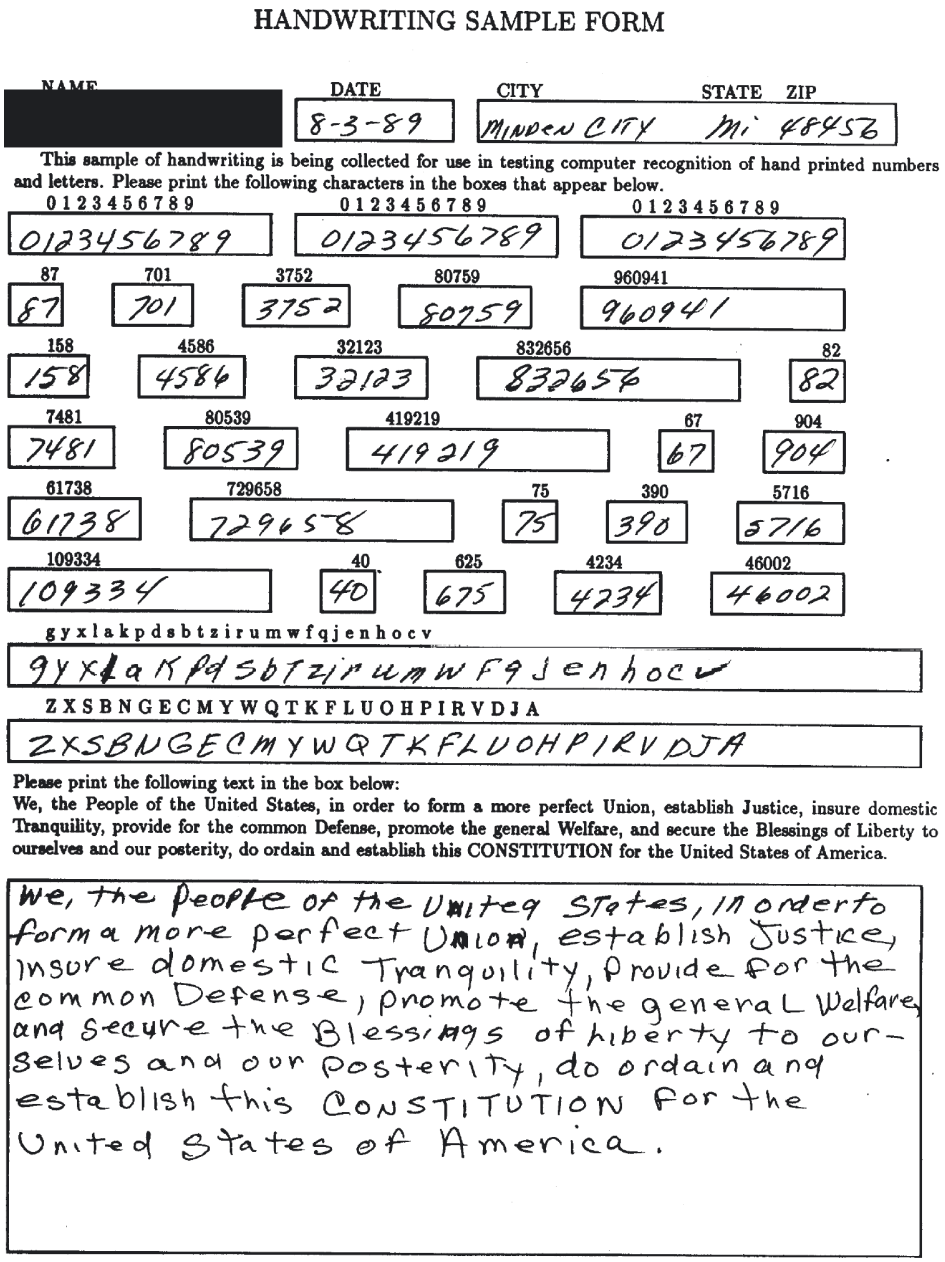


Figure 5.17: *Form filled by writers in the NIST Special Database 19. Source: NIST [131]*

- *Balanced*: both By_Class and By_Merge datasets are very unbalanced when it comes to letters, a fact that could negatively impact the classification performance. This dataset takes the By_Merge dataset and reduces the number of instances from 814,255 (total number of samples in NIST Special Database 19) to only 131,600, guaranteeing that there is an equal number of samples per each label.

- *Digits*: similar to MNIST, but with four times more instances (280,000 instead of 70,000).

- *Letters*: this dataset contains only letters, and a distinction between uppercase and lowercase is not made. As a result, the dataset contains 26 classes, and a total of 145,600 samples.

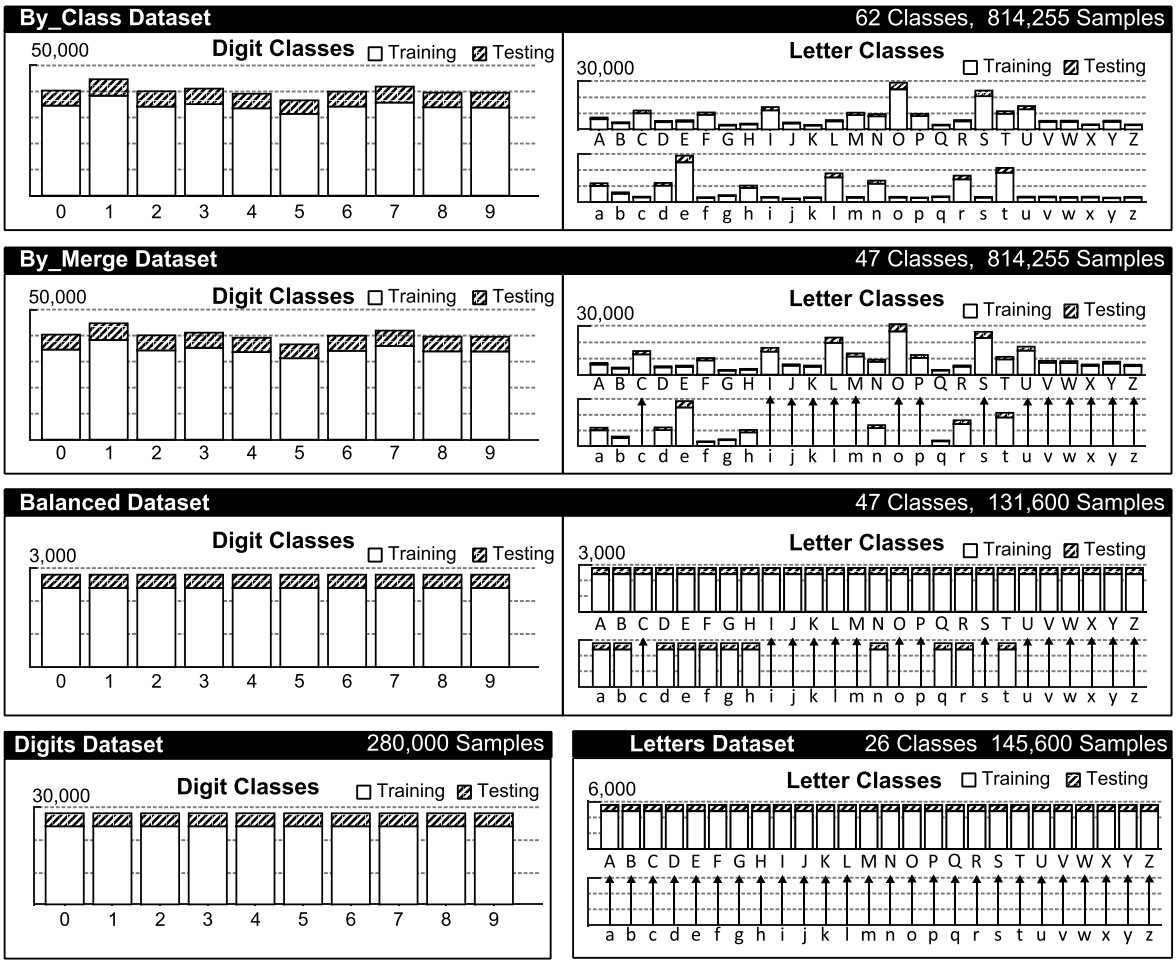


Figure 5.18: *Different datasets within EMNIST. Source: Cohen et al. [73]*

A summary of the different datasets within EMNIST as provided by the authors is shown in figure 5.18. In this work, we will focus on the Digits and Letters datasets, which are the ones located at the bottom of the figure.

5.3.2 *State of the Art*

To the best of our knowledge, the EMNIST dataset is so new that there are very few published works using it as a benchmark. The original EMNIST paper by Cohen et al. [73] includes a baseline using a linear classifier and OPIUM (Online Pseudo-Inverse Update Method), a classifier introduced by van Schaik and Tapson [371]. It is remarkable that authors declare to have selected this technique because it generates an analytical solution which is deterministic, and their aim is not to obtain cutting-edge results. In fact, they report an accuracy of 85.15 % in the Letters dataset and 95.90 % in the Digits dataset using OPIUM. It is worth noting that the performance in the original EMNIST paper is reported in terms of accuracy instead of error rate. For this reason, in this section we will use this metric for reporting the performance of each work.

More recently, Peng and Yin [279] have used Markov random field-based CNNs achieving an accuracy of 95.44 % in the letters dataset and of 99.75 % in the digits dataset. Also, Singh et al. [334]

reported an accuracy of 99.62 % in EMNIST Digits, using a CNN with three convolutional layers and two fully connected layers.

Despite the fact of EMNIST not being used so far in other published works, some researchers have used NIST Special Database 19 in the past. It should be noted that these works are not directly comparable, because the database is not identical; however, results can be extrapolated. For example, Milgram et al. [244] used only digits from this database, taking 195,000 samples for the training set and two different test sets of 60,089 and 58,646 samples respectively. The best result they reported using SVMs with sigmoid function is an accuracy of 99.37 % for the first test set and 98.12 % for the second, resulting in a weighted average accuracy of 98.75 %. Granger et al. [127] also worked with this dataset and benchmarked on the same test sets than Milgram et al., using particle swarm optimization to evolve the topology of neural networks; attaining non-competitive results, resulting in accuracies of 97.90 % and 95.05 % respectively (weighted average accuracy: 96.49 %). Also, Oliveira et al. [276] had previously tested a multilayer perceptron on this dataset, obtaining accuracies of 99.16 % and 97.60 % respectively (weighted average accuracy: 98.39 %)

Other authors have also used letters from NIST Special Database 19. For example, Radtke et al. [293] optimize a classifier using an annealing-based approach called record-to-record travel (RRT), reporting an average accuracy of 96.53 % for digits and 93.78 % for letters. Koerich and Kalva [185] test a multilayer perceptron only with the letters dataset, attaining an accuracy of 87.79 %. Also, Cavalin et al. [53] use hidden Markov models and report an accuracy of 98 % for digits and up to 90 % for letters, though this result is only using uppercase letters, and the accuracy decreases to 87 % when lowercase letters are also considered.

Finally, to the best of our knowledge, Cireşan et al. [69] are the only authors to have used this database for testing the performance of committees of CNNs, attaining accuracies of 88.12 % for the whole database, 92.42 % for letters and 99.19 % for digits.

A summary of the reviewed works is shown in table 5.8. The upper side of the table shows the performance of classical machine learning models and non-convolutional neural networks, whereas the lower side shows those works involving CNNs. Best results are boldfaced. Results marked with a star ($*$) indicate that they refer to works using samples from NIST Special Database 19 which are similar but not equivalent to EMNIST. The † symbol near the work by Cohen et al. [73] means that it is published in an pre-print repository, and thus has not been peer-reviewed.

Technique	By_Class	By_Merge	Balanced	Letters	Digits
Linear classifier [73]	51.80 %†	50.51 %†	50.93 %†	55.78 %†	84.70 %†
OPIUM [73]	69.71 %†	72.57 %†	78.02 %†	85.15 %†	95.90 %†
SVMs (one against all + sigmoid) [244]	–	–	–	–	98.75 %*
Multilayer perceptron [276]	–	–	–	–	98.39 %*
Hidden Markov model [53]	–	–	–	90.00 %*	98.00 %*
Record-to-record travel [293]	–	–	–	93.78 %*	96.53 %*
PSO + fuzzy ARTMAP ANNs [127]	–	–	–	–	96.49 %*
Multilayer perceptron [185]	–	–	–	87.79 %*	–
Markov random field CNN [279]	87.77 %*	90.94 %	90.29 %	**95.44 %**	**99.75 %**
Parallelized CNN [334]	–	–	–	–	99.62 %
Committee of 7 CNNs [69]	88.12 %*	–	–	92.42 %*	99.19 %*

Table 5.8: *Side-by-side comparison of the results for the EMNIST dataset, including works using similar datasets from NIST Special Database 19.*

5.3.3 Preprocessing

We have not performed any further preprocessing or data augmentation of the EMNIST dataset; instead, the Letters and Digits databases have been used out-of-the-box.

5.3.4 Results

As we stated before, for EMNIST we will not run evolutionary computation techniques again, but rather will reuse the architectures obtained when optimizing for the MNIST dataset. We hypothesize that these architectures should work properly, because the data structure is equivalent and the domain, handwritten characters, is very similar. If we achieved good results in the EMNIST dataset, we could validate this hypothesis. If good topologies could be reused for similar domains with good performance, this would unleash a great potential and save time when applying already known topologies to new problems.

In this section we will describe the behavior of the models learned from the architectures resulting from the genetic algorithm and from grammatical evolution. Just as we proceeded in the MNIST experiments, each of the top-20 architectures will be trained 20 times from scratch, using 30 epochs and the whole training set.

5.3.4.1 Genetic Algorithm

Regarding the GA, a statistical summary of the accuracies for each architecture is shown in table 5.9 for the Letters dataset and table 5.10 for the Digits dataset, showing the mean, median, standard

#	Mean	Std. Dev.	Median	Min.	Max.
1	94.3790	0.098457	94.375	94.15	94.60
2	94.4025	0.062482	94.390	94.29	94.53
3	93.6130	0.086700	93.605	93.42	93.76
4	94.0400	0.111308	94.035	93.87	94.21
5	93.3285	0.122572	93.335	93.09	93.54
6	94.5270	0.049852	94.535	94.45	94.61
7	94.7125	0.048869	94.720	94.62	94.82
8	93.9495	0.096925	93.970	93.80	94.15
9	94.6215	0.080018	94.635	94.47	94.76
10	93.5715	0.093711	93.545	93.38	93.73
11	94.5980	0.077974	94.620	94.41	94.72
12	94.0260	0.081137	94.025	93.87	94.18
13	93.4670	0.130590	93.450	93.25	93.69
14	94.2110	0.064064	94.210	94.08	94.36
15	94.2975	0.075315	94.310	94.09	94.41
16	93.8320	0.143696	93.870	93.40	94.03
17	94.1870	0.072555	94.195	94.03	94.30
18	94.1520	0.086304	94.185	94.00	94.29
19	94.6340	0.080289	94.640	94.49	94.77
20	94.2570	0.078680	94.250	94.13	94.42

Table 5.9: *Summary of accuracies of the best 20 GA individuals after full training in EMNIST Letters.*

#	Mean	Std. Dev.	Median	Min.	Max.
1	99.579000	0.014832	99.580	99.55	99.62
2	99.641000	0.011192	99.640	99.62	99.66
3	99.575000	0.025649	99.580	99.49	99.60
4	99.406842	0.039589	99.420	99.32	99.46
5	99.504000	0.021619	99.505	99.47	99.54
6	99.640000	0.015560	99.640	99.61	99.67
7	99.640500	0.011910	99.640	99.62	99.67
8	99.602500	0.020229	99.605	99.56	99.63
9	99.664000	0.010954	99.660	99.64	99.68
10	99.543500	0.017852	99.545	99.50	99.57
11	99.614500	0.015720	99.615	99.59	99.64
12	99.507000	0.016890	99.505	99.48	99.54
13	99.406500	0.041710	99.405	99.34	99.46
14	99.601000	0.018325	99.600	99.57	99.64
15	99.539500	0.018202	99.540	99.50	99.57
16	99.545000	0.020647	99.540	99.51	99.58
17	99.616000	0.015694	99.620	99.59	99.64
18	99.592500	0.014824	99.590	99.57	99.63
19	99.623500	0.017852	99.620	99.60	99.66
20	99.631000	0.015526	99.630	99.61	99.66

Table 5.10: *Summary of accuracies of the best 20 GA individuals after full training in EMNIST Digits.*

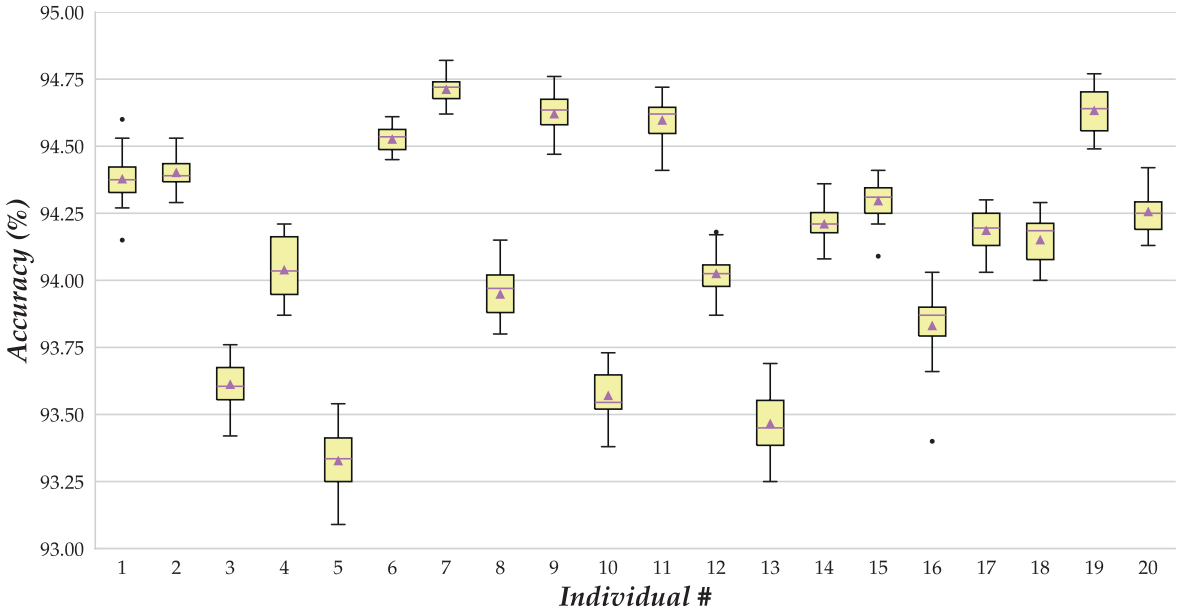


Figure 5.19: *Boxplot showing the distribution of accuracies of the best 20 GA individuals after full training in the EMNIST Letters dataset.*

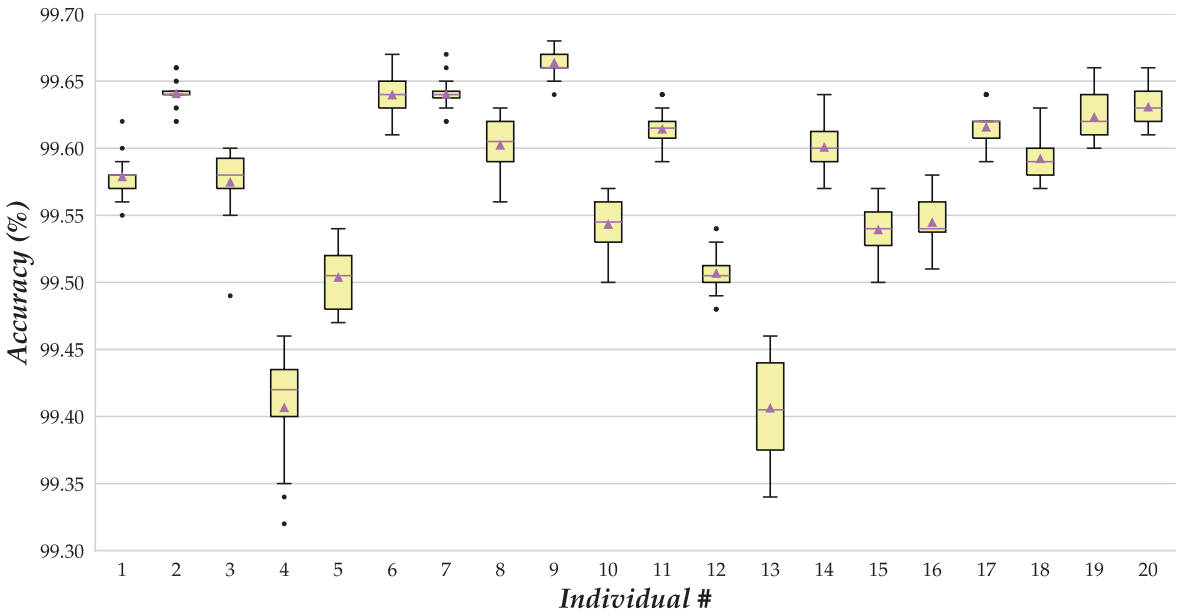


Figure 5.20: *Boxplot showing the distribution of accuracies of the best 20 GA individuals after full training in the EMNIST Digits dataset.*

deviation, maximum and minimum values. Also, the results distributions are also depicted in figures 5.19 and 5.20 respectively. We can see how, unlike the case of the GA with MNIST, these distributions seems more chaotic regarding the individual position in the ranking. This fact may be an indicator that these topologies are not the most suitable for the EMNIST dataset. On the other hand, results seems fairly good, as even the worst values are better than those found in the state of the art for MNIST and NIST Special Database 19 for both the Letters and Digits datasets.

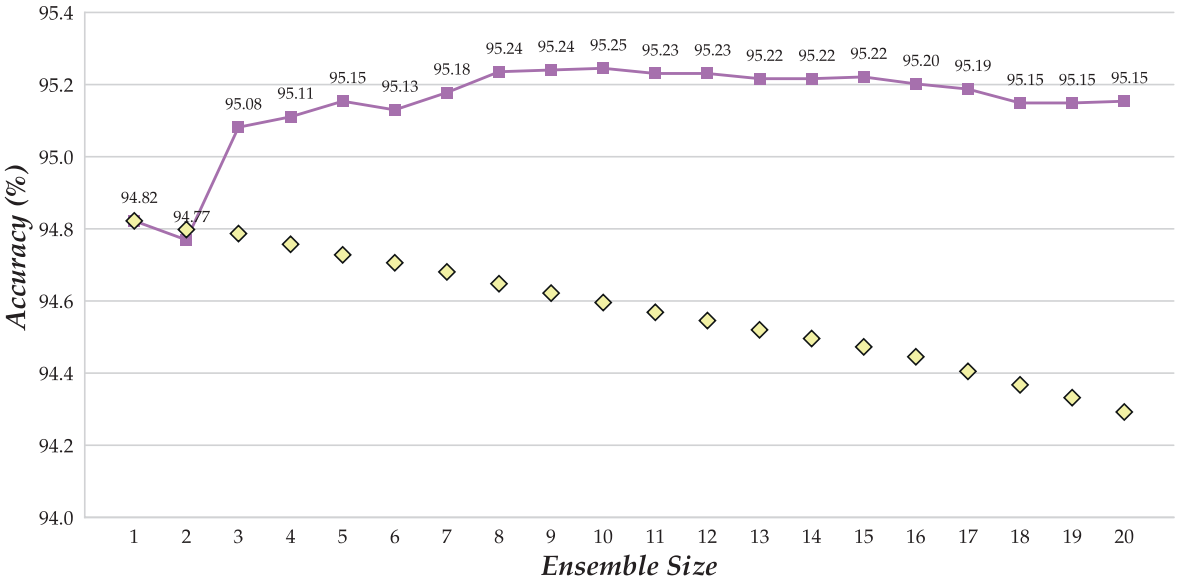


Figure 5.21: *Evolution of the accuracy of the incremental ensembles using the best 20 individuals from the GA with EMNIST Letters.*

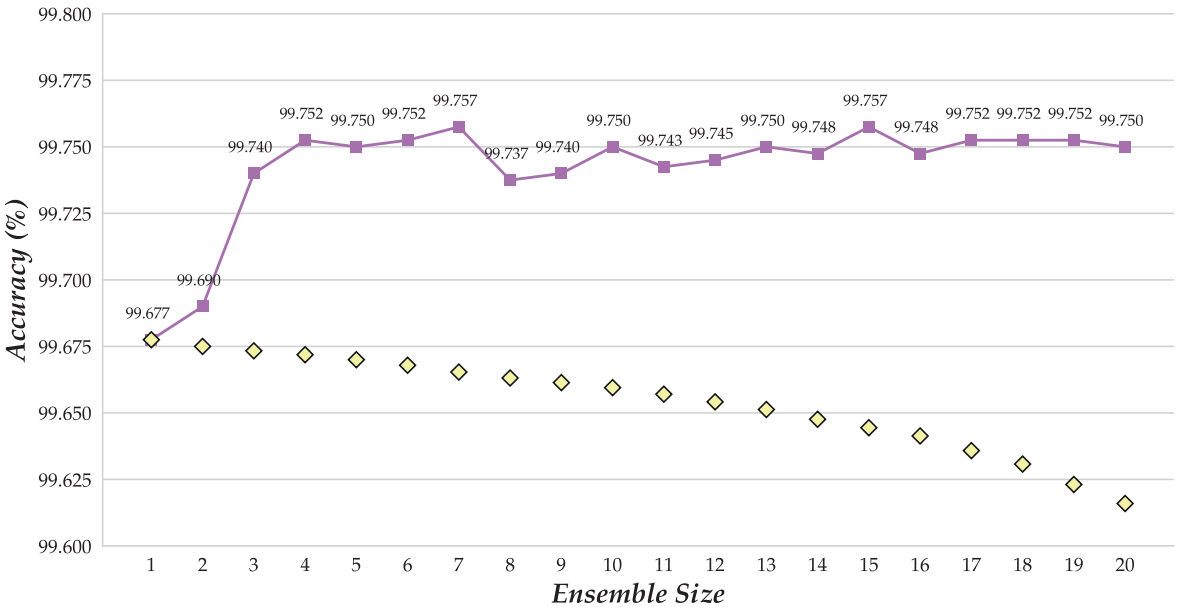


Figure 5.22: *Evolution of the accuracy of the incremental ensembles using the best 20 individuals from the GA with EMNIST Digits.*

Now, we will use these models to work within an ensemble. Figure 5.21 shows the evolution of accuracy for the Letters dataset as new models are added. The best result in EMNIST Letters is an accuracy of 95.25 % (error rate of 4.75 %), with 10 CNNs. It is noticeable that when the ensemble reaches 8 CNN models, adding new classifiers have little effect in the results.

The plot referred to the Digits dataset is shown in figure 5.22. It is noticeable that beyond 4 individuals, the accuracy stabilizes around 99.75 % (error rate of 0.25 %), a behavior consistent with

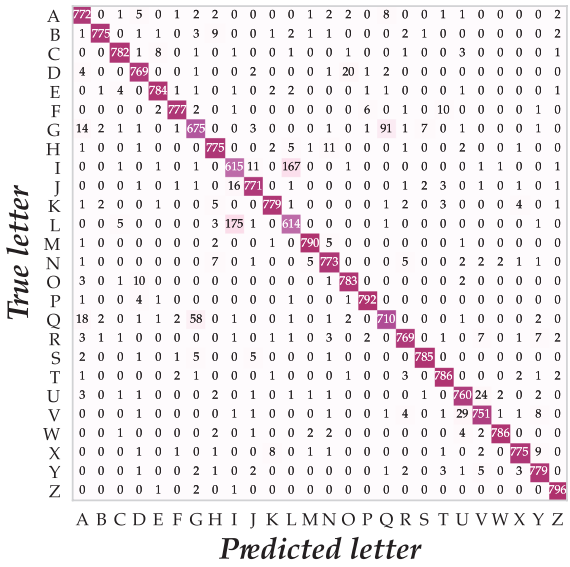


Figure 5.23: *Confusion matrix of the best ensemble using the GA individuals with EMNIST Letters.*

Figure 5.24: *Some misclassified images in the EMNIST Letters test set with the best ensemble obtained with GA.*

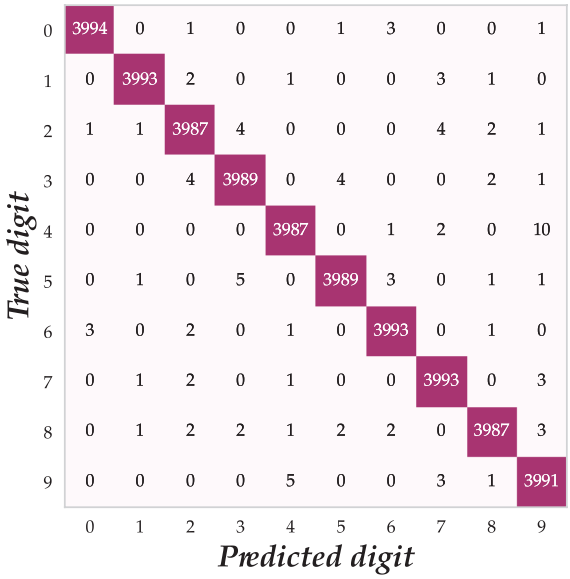


Figure 5.25: *Confusion matrix of the best ensemble using GA individuals with EMNIST Digits.*

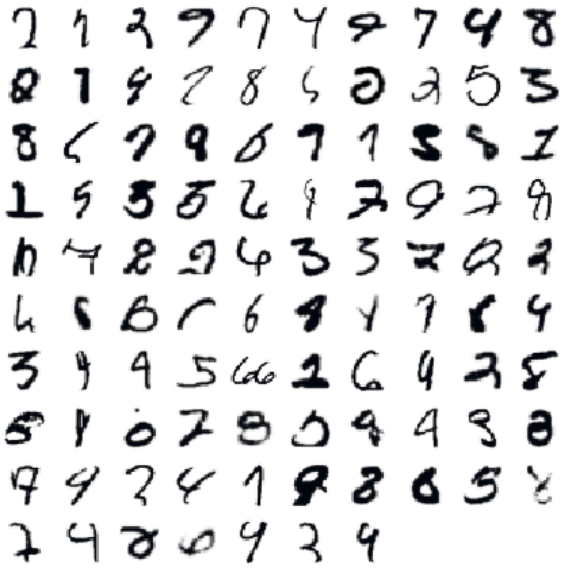


Figure 5.26: *All misclassified images in the EMNIST Digits test set with the best ensemble obtained with GA.*

the Letters dataset. The best value is achieved when the ensemble comprises either 7 or 15 CNNs, with an accuracy of 99.7575 % (error rate of 0.2425 %).

Figure 5.23 shows the confusion matrix for the EMNIST Letters dataset using the best ensemble found. It can be seen that accuracy is almost perfect. Most common mistakes involve mixing up the letters 'I' and 'L', the letters 'G' and 'Q', and to a much lesser extent the letters 'V' and 'U'.

These seem like acceptable mistakes given the high similarity of these characters. Figure 5.24 shows 100 misclassified images in the EMNIST Letters dataset. As we already knew from the confusion matrix, most misclassified samples are vertical bars which could be either an ´L' or an 'I' (notice that both are even more similar when comparing a lowercase 'l' with an uppercase 'I'). Also, it can be seen how some characters are hardly recognizable even by a human.

As for the confusion matrix for the Digits dataset using the best ensemble found, it is depicted in figure 5.25. Again, most values are in the main diagonal, representing an almost perfect accuracy. Most remarkable mistakes involve misclassifying digits '9' and '4', '3' and '5', and '2' and '3'. To a lesser extent, the ensemble also mixes up the digits '6' and '0'. From a test set of 40,000 samples, only 97 were incorrectly classified. The whole set of misclassified instances is shown in figure 5.26. Most of these digits are hardly recognizable. In fact, one instance seems to involve two digits in one (seventh row, fifth column). Others seem to be incomplete, and it is hard to tell whether they are a '5' or a '3' (e.g., eight row, sixth column). Finally, the confusion between '4's and '9's seems to arise either because a digit '4' is very rounded on the top, or the digit '9' seems to be slightly open.

Finally, it is remarkable that the best ensemble found in this work would rank the second when compared to related works in the state of the art for the Letters dataset, and the first for the Digits dataset. In the case of Letters, our ensemble obtained an accuracy of 95.25 %, only outperformed by the work by Peng and Yin [279], with an accuracy of 95.44 %. As for Digits, our ensemble attained an accuracy of 99.7575 %, slightly better than that same work by Peng and Yin [279].

5.3.4.2 Grammatical Evolution

As for grammatical evolution, a statistical summary of the accuracies for each architecture is shown in table 5.11 for the Letters dataset and table 5.12 for the Digits dataset, showing the mean, median, standard deviation, maximum and minimum values. Also, the results distributions are also de-

#	Mean	Std. Dev.	Min.	Median	Max.
1	94.6585	0.074004	94.665	94.51	94.76
2	94.7300	0.070934	94.740	94.61	94.86
3	94.5635	0.084870	94.580	94.41	94.70
4	95.1215	0.049553	95.120	95.02	95.19
5	94.2790	0.064880	94.275	94.16	94.38
6	94.7230	0.068832	94.710	94.63	94.88
7	94.6540	0.053646	94.650	94.53	94.74
8	93.7270	0.094429	93.705	93.60	93.93
9	93.6515	0.070058	93.645	93.50	93.75
10	94.5305	0.139075	94.550	94.18	94.75
11	94.6890	0.045410	94.690	94.61	94.78
12	94.4335	0.071545	94.440	94.28	94.55
13	94.4330	0.075888	94.435	94.28	94.61
14	94.6260	0.132045	94.630	94.24	94.81
15	94.4605	0.062616	94.465	94.35	94.57
16	94.1590	0.093578	94.165	94.00	94.29
17	93.4505	0.156423	93.455	93.14	93.81
18	93.5095	0.149929	93.505	93.25	93.77
19	94.2725	0.074684	94.270	94.13	94.38
20	94.5085	0.070208	94.495	94.41	94.64

Table 5.11: *Summary of accuracies of the best 20 GE individuals after full training in EMNIST Letters.*

#	Mean	Std. Dev.	Min.	Median	Max.
1	99.6835	0.009881	99.680	99.66	99.70
2	99.6880	0.015079	99.685	99.66	99.72
3	99.6155	0.018489	99.610	99.58	99.65
4	99.7145	0.009987	99.720	99.70	99.73
5	99.5420	0.029487	99.540	99.48	99.60
6	99.6780	0.019628	99.680	99.64	99.72
7	99.6765	0.015652	99.680	99.64	99.70
8	99.5410	0.024900	99.540	99.49	99.59
9	99.4655	0.039533	99.475	99.39	99.52
10	99.6570	0.016575	99.650	99.63	99.69
11	99.6785	0.013485	99.675	99.66	99.72
12	99.6115	0.016944	99.610	99.58	99.64
13	99.6665	0.016944	99.665	99.63	99.70
14	99.6205	0.022589	99.620	99.58	99.66
15	99.6560	0.011425	99.660	99.64	99.68
16	99.6305	0.013945	99.630	99.60	99.65
17	99.3050	0.070599	99.320	99.16	99.39
18	99.5220	0.031722	99.525	99.44	99.59
19	99.6480	0.015079	99.650	99.62	99.68
20	99.6015	0.015313	99.600	99.58	99.63

Table 5.12: *Summary of accuracies of the best 20 GE individuals after full training in EMNIST Digits.*

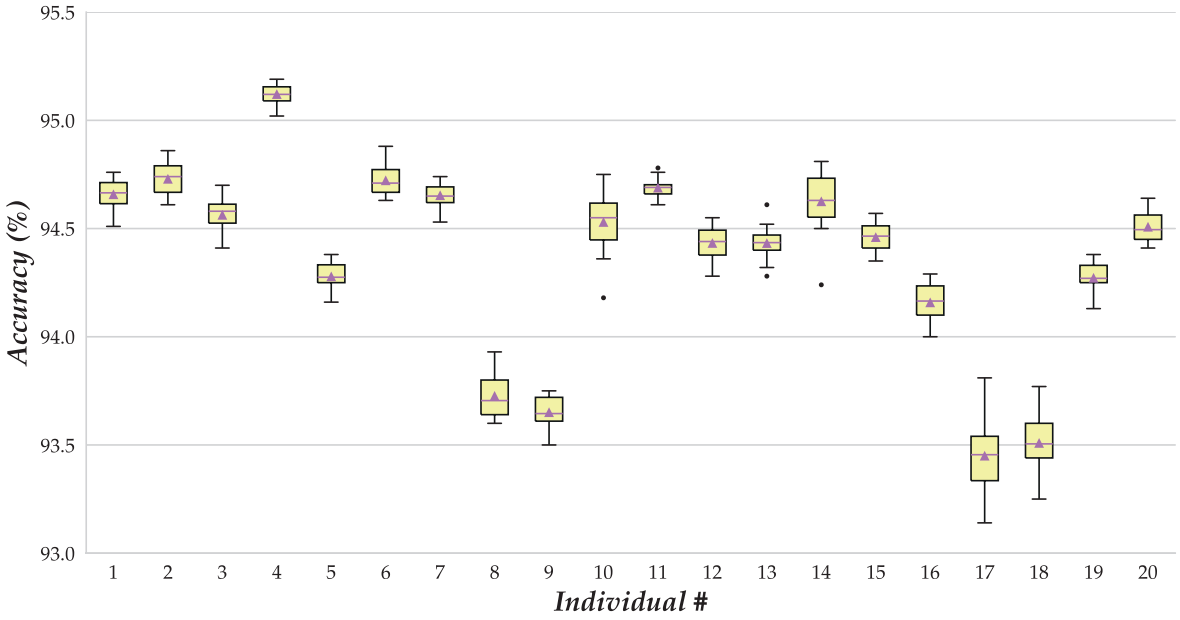


Figure 5.27: *Boxplot showing the distribution of accuracies of the best 20 GE individuals after full training in the EMNIST Letters dataset.*

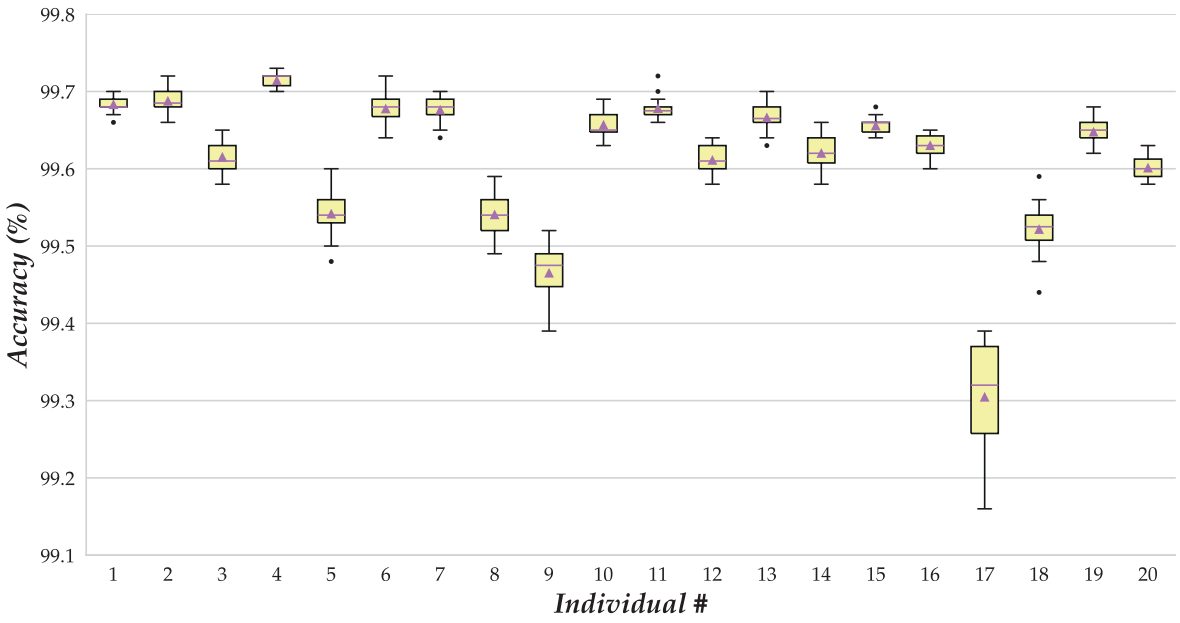


Figure 5.28: *Boxplot showing the distribution of accuracies of the best 20 GE individuals after full training in the EMNIST Digits dataset.*

picted in figures 5.27 and 5.28 respectively. Again, as it happened in the genetic algorithm, it does not seem that first individuals behave better than the rest, pointing out that these topologies are not explicitly optimized for the EMNIST dataset. However, results are very good: both in the Digits and the Letters datasets, the maximum accuracies obtained are over 99.7 % and 95 % respectively, better than the best values obtained with the topologies optimized using the GA and very close to the highest accuracy achieved using ensembles.

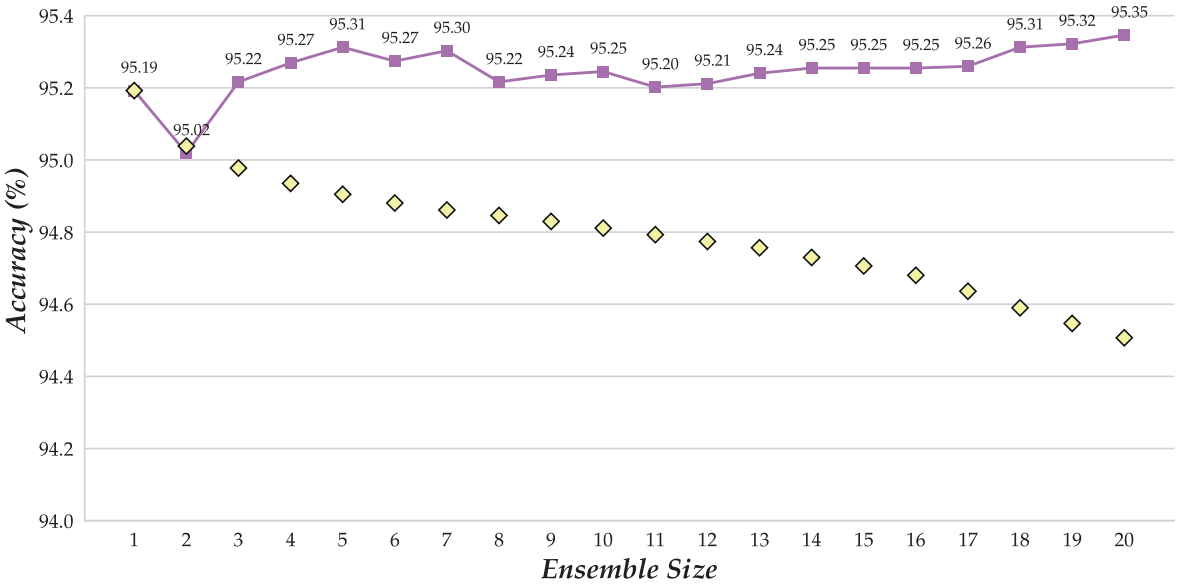


Figure 5.29: *Evolution of the accuracy of the incremental ensembles using the best 20 individuals from the GE with EMNIST Letters.*

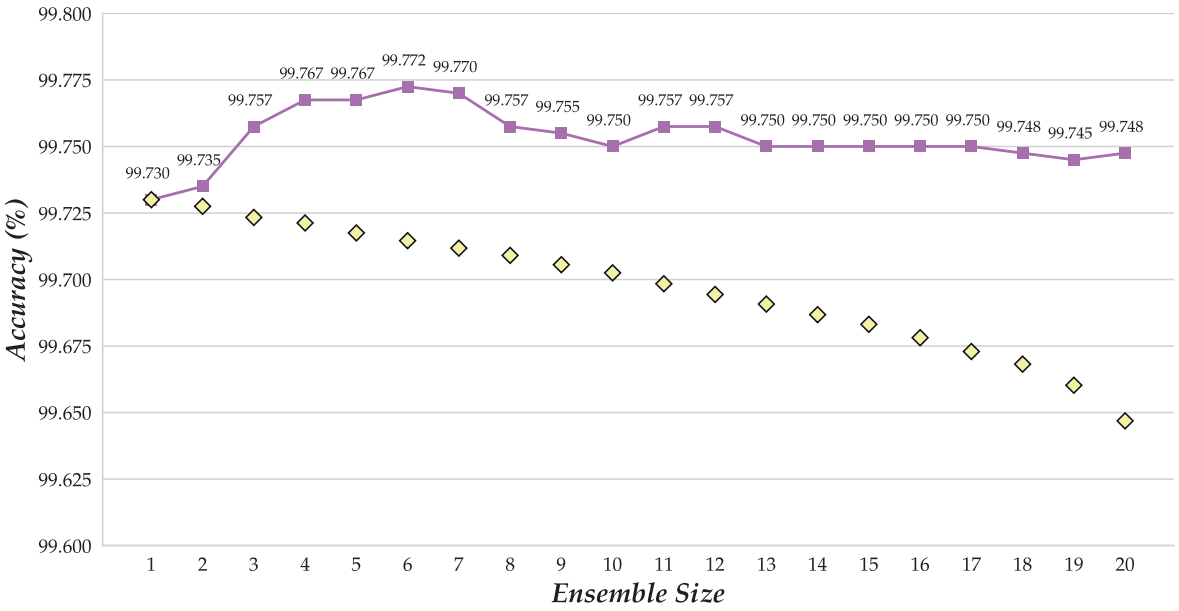


Figure 5.30: *Evolution of the accuracy of the incremental ensembles using the best 20 individuals from the GE with EMNIST Digits.*

Also, in the Digits dataset, the best accuracy found (99.73 %) is already higher than the best result reported for MNIST using ensembles (99.72 %). The performance seems to be consistent across both datasets, i.e., best models in Letters seems to also behave better in Digits.

Again, we will use these models to build a committee of CNNs. Figure 5.29 shows the evolution of accuracy for the Letters dataset as new individuals are added to the ensemble, and so does figure 5.30 for the Digits dataset.

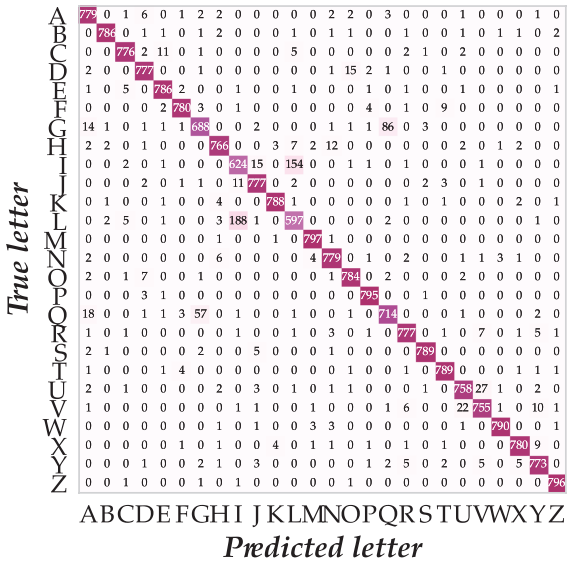


Figure 5.31: *Confusion matrix of the best ensemble using the GE individuals with EMNIST Letters.*

Figure 5.32: *Some misclassified images in the EMNIST Letters test set with the best ensemble obtained with GE.*

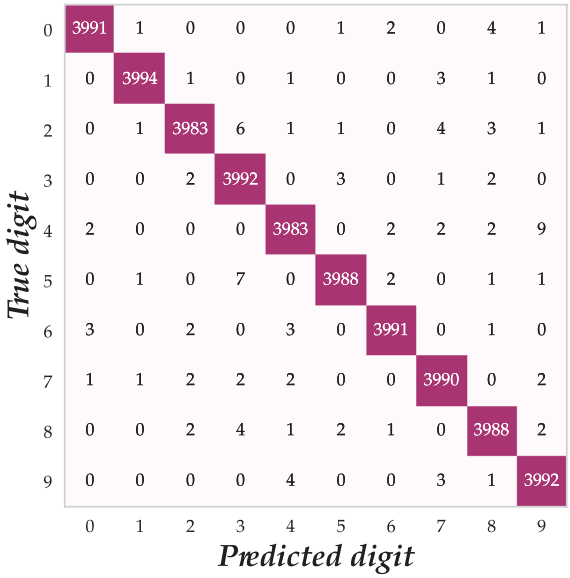


Figure 5.33: *Confusion matrix of the best ensemble using the GE individuals with EMNIST Digits.*

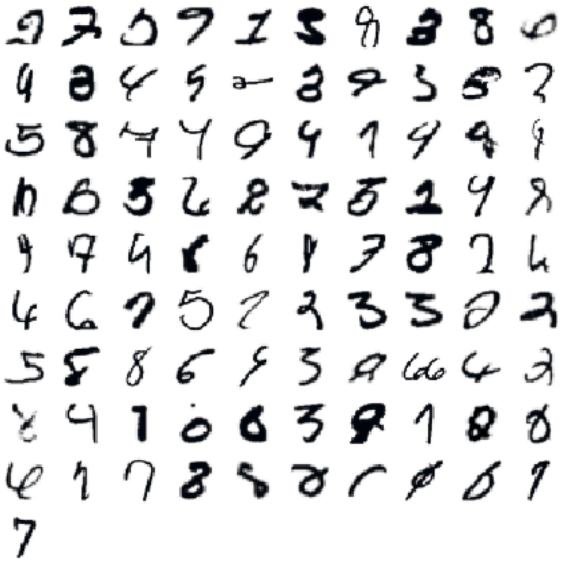


Figure 5.34: *All misclassified images in the EMNIST Digits test set with the best ensemble obtained with GE.*

The best result in EMNIST Letters is an accuracy of 95.35 % (error rate of 4.65 %), with a 20-CNNs ensemble. The accuracy looks very steady when more than three CNNs are involved, yet results improve at the end, attaining better accuracy with a larger number of models. In the Digits dataset, the best accuracy is 99.7725 % (error rate of 0.2275 %), with a committee of six CNNs. The accuracy is very steady as new models are added to the ensemble, with the worst result obtaining only 0.03 percentage points less than the aforementioned result.

Finally, the confusion matrix for EMNIST Letters using the best models achieved with GE is shown in figure 5.31. Once again, most frequent mistakes involve mixing up 'L' with 'I' and 'G' with 'Q'. A random subset of 100 misclassified instances is shown in figure 5.32, which looks consistent with the information reported in the confusion matrix.

We can see how it can be really difficult to identify some of the letters in this sample, some remarkable examples being the ninth letter from the first row, which seems to be a '3' digit rather than a letter, or the sixth letter from the second row, which seems unintelligible. Additionally, there are many vertical bars which are difficult to identify as either 'L' or 'I', which turns out to be the most frequent mistake as reported by the confusion matrix.

As for EMNIST Digits, the confusion matrix is depicted in figure 5.33, showing an almost perfect accuracy, with only a few mistakes mixing up digits '4' and '9' and '3' with '5', which were also the most common mistakes in the MNIST dataset. In fact, only 91 instances from a total of 40,000 in the test set have been incorrectly classified, and these can be seen in figure 5.34.

Also, many of these incorrectly classified digits were also found when the topologies obtained with the GA were tested (see figure 5.26). One of the misclassified samples, which was also present in the GA, involves a wrongly-segmented number, containing two '6' instead of one (eight sample from seventh row). In general, the fact that there is so much consistency between misclassified instances in both the GA and GE could indicate that these samples are specially hard to recognize, regardless how powerful the model is.

Models trained with the topologies evolved using GE outperform those resulting from GA in both Letters (95.35 % vs. 95.25 %) and Digits (99.7725 % vs. 99.7575 %). However, the position of our work in the ranking of the state of the art remains unaltered, since the best result for the Letters dataset is still slightly below the accuracy reported by Peng and Yin [279] (95.44 %) even after this improvement, therefore taking the second position.

5.4 OPPORTUNITY

OPPORTUNITY is a complex human activity dataset introduced by Roggen et al. in 2010 [305]. This dataset results from an EU-funded project whose aim is to recognize human activity from *opportunistically* discovered sensors, therefore the name. Opportunistic sensor configurations imply that sensors are not placed on the body at precise locations. By contrast, sensors which are already present are used, even when their number, placement and setup are not known in advance.

5.4.1 Acquisition

According to the authors, OPPORTUNITY data are recorded in a sensor rich environment consisting of "*a room simulating a studio flat with kitchen, deckchair, and outdoor access where subjects performed daily morning activities*" [305]. 72 sensors belonging to 15 networked sensor systems and comprising 10 modalities were deployed over the environment, the objects and the subject's body.

The purpose of the OPPORTUNITY dataset is to record daily human activities in a realistic manner, trying to keep their execution natural. The authors did not provide specific instructions to subjects on how to perform the activities, thus leaving free interpretation and encouraging them to perform the tasks in the usual way they would do. Authors configured a sensor rich environment, an approach which poses some benefits: activities are sensed by multiple sensors, sensors in close proximity provide robustness against sensor placement variability, and reading sensors from diverse modalities and/or systems allows to assess their performance or reliability.

ID	Sensor system	Location and observation
B1	Commercial wireless microphones	Chest and dominant wrist
B2	Custom Bluetooth acceleration sensors [304]	12 in the body to sense limb movement
B3	Custom motion jacket [347]	Includes 5 commercial RS485-networked XSens inertial measurement units [394]
B4	Custom magnetic relative pos. sensor [284]	Senses distance of hand to body
B5	InertiaCube3 [162] inertial sensor system	One per foot, on the shoe toe box, to sense modes of locomotion
B6	Sun SPOT acceleration sensors	One per foot, right below the outer ankle, to sense modes of locomotion
O1	Custom wireless Bluetooth acceleration and rate of turn sensors	12 objects in the scenario to measure their use
A1	Commercial wired microphone array	4 in each room side to sense ambient sound
A2	Commercial Ubisense localization system	Placed in the corners of the room to sense user location
A3	Axis network cameras	Placed in three locations for localization, documentation and visual annotation
A4	XSens inertial sensor [347,394]	Placed on the table and the chair to sense vibration and use
A5	USB networked acceleration sensors [406]	8 placed on doors, drawers, shelves and the lazy chair to sense usage
A6	Reed switches	13 placed on doors, drawers and shelves, to sense usage providing ground truth
A7	Custom power sensors	Connected to coffee machine and bread cutter to sense usage
A8	Custom pressure sensors	3 placed on the table to sense usage after subjects placed plates and cups on them

Table 5.13: *Sensors used in the OPPORTUNITY dataset, placed over the body, the objects and the environment.*

In particular, six sensor systems were deployed on the subjects' bodies, one sensor system is placed on the objects and eight sensor systems are deployed over the environment. Table 5.13 describes the different systems as provided by authors in the original OPPORTUNITY paper [305].

The fact that many wireless sensors are closely placed also introduced additional technical challenges when building a suitable sensing environment. Because they used some proprietary sensor systems along with custom devices, different systems were required for the data acquisition infrastructure. In particular, authors used 7 computers (6 laptops and a desktop PC), and different proprietary software along with the CRN Toolbox previously introduced by Bannach et al. [17]. These computers were provisioned according to the needs of the sensors systems (e.g. video and audio streams required a dedicated computer, some sensors required a computer to be placed within a range, etc). Room lighting was controlled by closing the blinds to avoid sunlight changes and switching on fluorescent tubes for the whole experiment (except when subjects were required to turn them off as a part of the protocol).

During the experiment, subjects were asked to perform five ADL (activity of daily living) runs and one drill run. The ADL runs comprise the next sequence of activities:

1. Start: lie in the deckchair and then get up.
2. Groom: move across the room, checking that objects are in the right drawers and shelves.
3. Relax: go outside and have a walk around the building.
4. Prepare coffee: use the coffeemaker to prepare coffee with milk (in the fridge) and sugar.
5. Drink coffee: take coffee sips naturally.
6. Prepare sandwich: made of bread, cheese and salami and using the bread cutter along with various knifes and plates.
7. Eat the sandwich.
8. Cleanup: clean the table, store objects back in their place or in the dish washer.
9. Break: lie on the deckchair.

Activities in the ADL run can be considered with different level of abstractions, thus enabling a hierarchy of actions. For example, an abstract activity could be "*prepare a sandwich*", comprising some composite activities such as "*cut bread*", which at the same time can be composed of atomic activities like "*move to bread*" or "*operate bread cutter*". Each ADL lasted about 15-25 minutes, and a break of 10-20 minutes between runs were given to copy data and ensure that devices were charged and running properly. An instructor guided subjects during the first run, yet placing very little constraints. Thus, subjects were asked to follow the high-level action sequence, and allowed to interleave actions (e.g. prepare the sandwich while still drinking coffee), switch hands, etc.

After the five runs, subjects had to complete the drill run, where they were asked to perform 20 repetitions of the following sequence, in order to generate many activity instances:

1. Open and close the fridge.
2. Open and close the dishwasher.
3. Open and close 3 drawers at different heights.
4. Open and close door 1.
5. Open and close door 2.
6. Turn on and off the lights.
7. Clean table.
8. Drink while standing.
9. Drink while sitting.

Drill runs lasted between 20 and 35 minutes. After all the experiments took place, an open source tool was used to annotate the dataset. This annotation was done in four different tracks: the first contains modes of locomotion (sitting, standing, walking...), the following two contain actions of the left and right hand (reach, grasp, release...) along with the object of the interaction and the last contains high level activities (prepare sandwich, prepare coffee...). According to authors, 30 minutes of data recording require between 7 and 10 hours of annotation time.

Deliverable D5.1 of the OPPORTUNITY Project [306] provides a much more detailed description of the dataset, including maps showing the placement of sensors, technical specifications of the sensing devices or signal processing techniques used to compose the final dataset.

5.4.2 State of the Art

A subset of the OPPORTUNITY dataset was used for the OPPORTUNITY Activity Recognition Challenge which took place in the workshop on robust machine learning techniques for human activity recognition of the 2011 IEEE Conference on Systems, Man and Cybernetics [58, 289]. This subset finally comprised four subjects and 113 sensor channels grouped in two different datasets: the locomotion dataset and the gestures dataset.

The locomotion dataset includes four classes: *stand* (1093), *sit* (1095), *walk* (90) and *lie* (40). The number between parentheses indicates the number of instances associated to that label. The gestures dataset includes 17 classes: *open dishwasher* (50), *close dishwasher* (56), *open fridge* (129), *close fridge* (133), *open drawer 1* (50), *close drawer 1* (49), *open drawer 2* (44), *close drawer 2* (44), *open drawer 3* (56), *close drawer 3* (57), *open door 1* (45), *close door 1* (39), *open door 2* (43), *close door 2* (41), *move cup* (184) and *clean table* (33). Additionally, instances in both datasets may not be labelled (i.e., belong to a "*null*" class), and this *null* class has a high prevalence. Numbers show that the first dataset is very unbalanced (most of the times subjects are sitting or standing), whereas the second is pretty much balanced, except for the "*move cup*" class.

In this section, we will report performance in terms of the weighted F1 score metric, whose formula is shown in equation 5.1, with n_c/N being the proportion of samples of class c:

$$F_1 = 2\sum_c \left(\frac{n_c}{N} \frac{precision_c \times recall_c}{precision_c + recall_c} \right) \tag{5.1}$$

Also, we will adhere to the guidelines provided in the OPPORTUNITY challenge for training and testing classifiers, in order to enable a side-by-side comparison:

- Training set: comprises all ADL and drill sessions for subject 1 and ADL1, ADL2 and drill sessions for subjects 2 and 3.
- Test set: comprises ADL4 and ADL5 for subjects 2 and 3.

When results are not available for the whole test set, then the average weighted F1 score for individual subjects 2 and 3 has been computed.

A benchmark of different classification techniques was first published by the dataset authors for the workshop where the OPPORTUNITY challenge took place [311]. In the gestures dataset, best results were achieved using k-nearest neighbors (k-NN) with $k = 3$, leading to an F1 score of 0.85 when considering the *null* class (which drops to 0.56 when the *null* class is removed). 3-NN also outperformed other classifiers in the locomotion dataset, where F1 scores ranging from 0.85 to 0.86 were attained regardless of whether the *null* class was considered or not.

In 2012, Cao et al. [51] proposed an integrated framework for human activity recognition, whose performance was tested with the OPPORTUNITY dataset. This framework obtained an F1 score of up to 0.821 for the gesture recognition problem with *null* class, and 0.927 for locomotion recognition without *null* instances.

Also, a benchmark was published after the challenge took place revising the baseline results and summarizing the most remarkable contributions [58]. In the case of the gesture recognition problem, an F1 score of 0.88 was achieved including the *null* class, dropping to 0.77 when the *null* instances were removed. In both cases, the best classifier was a combination of support vector machines (SVM) with 1-nearest neighbor. Meanwhile, in locomotion recognition the best performance was attained by k-NN when considering the *null* class (F1 score of 0.85) and decision tree grafting [383] when ignoring the *null* class (F1 score of 0.87).

In 2015, Yang et al. [397] proposed the first application of deep learning techniques to learn a classifier for the OPPORTUNITY dataset, yet only focusing on the gesture recognition problem and considering the *null* class, for which they report an average F1 score of 0.818. Results are slightly higher (up to 0.822) when processing data using a technique known as *smoothing*, which was formerly described by Cao et al. [51].

Early in 2016, Ordóñez and Roggen proposed *DeepConvLSTM* [277], a deep learning technique combining convolutional layers with LSTM cells. In this approach, they pass the input data through four convolutional layers with rectified linear units (RELUs), and then through two recurrent dense layers with LSTM cells using a hyperbolic tangent activation function. Finally, the output is introduced to a softmax classifier. Their architecture features an F1 score of 0.915 in the gestures dataset, dropping to 0.866 when the *null* instances were removed. LSTM layers were proved to improve the performance of non-recurrent convolutional, which attained F1 scores of 0.883 and 0.783 respectively. Regarding locomotion recognition, Ordóñez and Roggen attained F1 scores of 0.895 and 0.93 (with and without *null* class respectively) using *DeepConvLSTM*, and 0.878 and 0.912 using non-recurrent convolutional neural networks.

Also in 2016, Hammerla et al. compared different deep learning solutions on the gesture recognition track of OPPORTUNITY with *null* instances [138]. They report an F1 score of 0.927 using bi-directional LSTM networks in a sample-by-sample basis.

In 2017, Guan and Plötz [134] suggested the use of ensembles of deep LSTM learners, combining some models trained using F1 score and others trained using cross entropy as the loss function. They reported an F1 score of 0.726 in the gestures dataset with the *null* class, which is substantially worst than results from Ordóñez and Roggen [277] or Hammerla et al. [138], although authors claimed to outperform previous works, apparently failing to acknowledge the actual results that had been reported in those works.

More recently, in 2018, Moya Rueda and Fink [258] have used a CNN based on the architectures previously presented by Ordóñez and Roggen [277] and Grzeszick et al. [133]. Their main contribution is that they use an evolutionary algorithm for optimizing the attributes that are projected from the sequential data. When they consider a sequence of convolutional layers per each inertial measurement unit, whose outputs are later concatenated before being introduced to the fully connected layers, they report an outstanding F1 score of 0.929 in the gestures dataset with the *null* class. This same idea was also previosly suggested by Moya Rueda et al. [259] attaining an F1 score of 0.9207.

Finally, in 2018, Yao et al. [399] have used fully convolutional, meaning that it does not involve dense or recurrent layers for classification. They report F1 scores of 0.869 and 0.887 for the locomotion dataset (with and withouth *null* class respectively), and 0.890 and 0.596 for the gestures dataset. Authors claim that their proposal outperforms all previous works when replicating their methods, although their attained values do not exceed F1 scores reported in those papers.

Table 5.14 provides an extensive yet comprehensive side-by-side comparison of the most relevant state of the art results attained for the OPPORTUNITY, including both the locomotion and gesture recognition tracks with and without *null* instances when available. The upper side of the table shows the performance of classical machine learning approaches, while the lower side displays the results of deep learning techniques, with the latter often showing better results. The best results are highlighted in boldface. Results marked with a star ($*$) indicate that authors reported the performance in a subject-per-subject basis, and are the outcome of averaging the F1 score for subjects 2 and 3; as a result, those values are only indicative. Also, the † symbol near some works means that they are published in a pre-print repository, and thus has not been peer-reviewed. Techniques are named as in the original papers, and the reader is referred to those papers to understand the specific setup for each technique.

Technique	**Locomotion**		**Gestures**	
	with *null* class	no *null* class	with *null* class	no *null* class
CStar [58]	0.63	0.87	0.88	0.77
1-NN [58]	0.84	0.85	0.87	0.55
SStar [58]	0.64	0.86	0.86	0.70
3-NN [58]	0.85	0.85	0.85	0.56
NStar [58]	0.61	0.86	0.84	0.65
Integrated Framework [51]	–	0.927*	0.821*	–
SPO + 1NN + Smooth. [51]	–	0.917*	0.811*	–
SPO + SVM + Smooth. [51]	–	0.897*	0.804*	–
SPO + SVM [51]	–	0.885*	0.797*	–
SVM [51]	–	0.883*	0.762*	–
SPO + 1NN [51]	–	0.890*	0.777*	–
1NN [51]	–	0.890*	0.705*	–
LDA [58]	0.59	0.64	0.69	0.25
UP [58]	0.60	0.84	0.64	0.22
QDA [58]	0.68	0.77	0.53	0.24
NCC [58]	0.54	0.60	0.51	0.19
MI [58]	0.83	0.86	–	–
MU [58]	0.62	0.87	–	–
NU [58]	0.53	0.75	–	–
UT [58]	0.52	0.73	–	–
attrCNN-IMU evol [258]	0.8975†	–	**0.929**†	–
attrCNN evol [258]	**0.900**†	–	0.9194†	–
b-LSTM-S [138]	–	–	0.927	–
DeepConvLSTM [277]	0.895	**0.930**	0.915	**0.866**
CNN-IMU [259]	0.8905	–	0.9207	–
LSTM-S [138]	–	–	0.912	–
LSTM-F [138]	–	–	0.908	–
CNN [138]	–	–	0.894	–
DNN [138]	–	–	0.888	–
Baseline CNN [277]	0.878	0.912	0.883	0.783
Dense labelling FCN [399]	0.869	0.887	0.890	0.596
CNN + Smooth. [397]	–	–	0.822*	–
CNN [397]	–	–	0.818*	–
MV + Smooth. [397]	–	–	0.788*	–
MV [397]	–	–	0.778*	–
Ensemble LSTM [134]	–	–	0.726	–
DBN [397]	–	–	0.701*	–
DBN + Smooth. [397]	–	–	0.700*	–

Table 5.14: *Side-by-side comparison of the most relevant results provided in the state of the art for the OPPORTUNITY dataset, including both the locomotion and the gesture recognition tracks, with and without null instances.*

5.4.3 Preprocessing

Very few preprocessing has been performed before feeding the OPPORTUNITY data to the CNN. The main aim of this stage is to clean data to remove missing values present in the raw data, and to transform data into a format supported by a tensor processing library (e.g. TensorFlow or Theano).

The first step involves removing features which must be ignored for the OPPORTUNITY challenge. These features are quaternion coefficients obtained from the inertial measurement units (a total of 16 features) and features obtained from sensors placed in objects and in the environment; i.e., only sensors placed in the subjects' bodies were considered (removing another 119 features). The filtered dataset will contain 113 features (channels) plus the class.

Later, we perform linear interpolation in each channel in order to provide an estimation for missing values (most of them arising from Bluetooth sensors disconnecting during the recording). Then, if there are missing values left, we set them to zero.

Finally, data is normalized in the interval $[0, 1]$ in order to avoid saturation of the neural network. This normalization is performed following equation 5.2, where x_c refers to a sample in channel c, and $\hat{x_c}$ is the normalized value:

$$\hat{x_c} = \frac{x_c - \min(c)}{\max(c) - \min(c)} \tag{5.2}$$

The data fed to the input layer of the deep neural network is shaped as a 4-dimensional tensor with dimensions $B \times 1 \times w \times 113$, where B is the batch size (as we are using mini-batching to train the model) and w is the window size. It must be noted that this window is extracted from a sliding window over the input data, with size w and step w_{step}. To obtain the overlapping sliding windows in an efficient manner we have followed the implementation suggested by John Vinyard [374].

5.4.4 Encoding

In this section we will discuss the encoding for both genetic algorithm and grammatical evolution. We will explain how the phenotype is built from the genotype in order to create suitable individuals.

5.4.4.1 Genetic Algorithm

The chromosome of the genetic algorithm for the OPPORTUNITY dataset consists of a 151-bit binary string using Gray encoding. A brief summary of the genotype's structure is shown in Figure 5.35. Next, we will explain this structure with further detail, as well as how the genotype is processed to be converted to a phenotype.

The chromosome encodes the next network setup, being x the integer corresponding to the Gray binary. The first three hyperparameters define the input configuration:

- w: the sliding window length (4 bits), computed as $w = 8\,(x + 1)$, which can take the values $w \in [8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, 128]$.

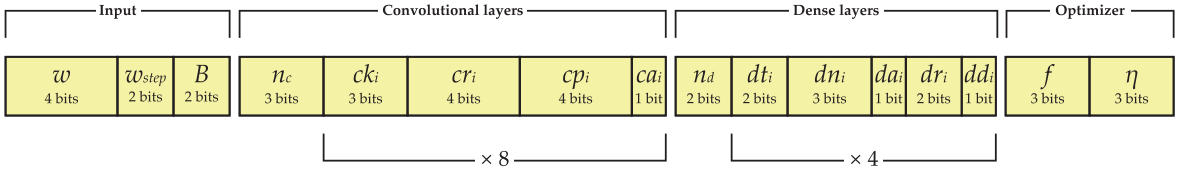


Figure 5.35: *Definition of the chromosome in the GA for the OPPORTUNITY dataset.*

- w_{step}: the sliding window step (2 bits), computed as $w_{step} = w/2^x$, thus taking the values $w_{step} \in [w, w/2, w/4, w/8]$.
- B: the batch size (2 bits), which can take values 25 ($x = 0$), 50 ($x = 1$), 100 ($x = 2$) and 150 ($x = 3$).

The following four hyperparameters define the setup of the convolutional layers:

- n_c: the number of convolutional layers (3 bits), computed as $n_c = 1 + x$, thus taking values between $n_c = 1$ and $n_c = 8$.
- ck_i: the number of kernels in the i-th convolutional layer (3 bits), computed as $ck_i = 2^x$, thus taking the values $ck_i \in [2, 4, 8, 16, 32, 64, 128, 256]$.
- cr_i: the number of rows of the i-th convolutional layer (4 bits), computed as $cr_i = 2 + (x/15)\,(w - 2)$, thus guaranteeing that the minimum value is $cr_i = 2$ and the maximum value is $cr_i = w$.
- cp_i: the pooling of the i-th convolutional layer (4 bits), computed as $cp_i = 1 + (x/15)\,(w - 2)$, thus guaranteeing that the minimum value is $cp_i = 1$ and the maximum value is $cp_i = w - 1$. If pooling size if 1, then it is equivalent to not doing pooling over the input.
- ca_i: the activation function of the i-th convolutional layer (1 bit), which can be either ReLU ($x = 0$) or linear ($x = 1$).

Because there will be at most 8 convolutional layers, the chromosome repeats 8 times the genes for ck_i, cr_i and cp_i. However, the network will only consider the setup only for the first n_c layers, and ignore the remaining. The following six hyperparameters define the setup of the dense layers:

- n_d: the number of dense layers (2 bits), computed as $n_d = 1 + x$, thus taking values between $n_d = 1$ and $n_d = 4$.
- dt_i: the type of the i-th dense layer (2 bits), which can be either recurrent ($x = 0$), LSTM ($x = 1$), GRU ($x = 2$) or feed-forward ($x = 3$).
- dn_i: the number of neurons in the i-th layer (3 bits), computed as $dn_i = 2^{(3+x)}$, thus taking the values $dn_i \in [8, 16, 32, 64, 128, 256, 512, 1024]$.
- da_i: the activation function of the neurons in the i-th layer (1 bit), which can be either ReLU ($x = 0$) or linear ($x = 1$).
- dr_i: the regularization applied to the weights of the i-th layer (2 bits), which which can be either none ($x = 0$), L1 ($x = 1$), L2 ($x = 2$) or L1+L2 ($x = 3$).
- dd_i: the dropout probability for the weights in the i-th layer (1 bit), which is computed as $dd_i = x/2$, thus taking the values $dd_i = 0$ (no dropout) or $dd_i = 0.5$.

Because there can be up to 4 dense layers, the chromosome repeats these genes four times; however, the network will only use the hyperparameters for the first n_d layers, ignoring the others.

Finally, the last two hyperparameters store the configuration of the optimization process:

- f: the optimizer or gradient descent update function (3 bits), which can be either SGD ($x = 0$), SGD with momentum ($x = 1$), SGD with Nesterov momentum ($x = 2$), AdaGrad ($x = 3$), AdaMax ($x = 4$), Adam ($x = 5$), AdaDelta ($x = 6$) or RMSProp ($x = 7$).
- η: the learning rate (3 bits), which can be either $1 \cdot 10^{-5}$ ($x = 0$), $5 \cdot 10^{-5}$ ($x = 1$), $1 \cdot 10^{-4}$ ($x = 2$), $5 \cdot 10^{-4}$ ($x = 3$), $1 \cdot 10^{-3}$ ($x = 4$), $5 \cdot 10^{-3}$ ($x = 5$), $1 \cdot 10^{-2}$ ($x = 6$) or $5 \cdot 10^{-2}$ ($x = 7$).

```
<dnn>         ::= <input> <conv_lys> <dense_lys> <opt_setup>

<input>       ::= <batch_size> <window_len> <window_step>
<batch_size>  ::= 25 | 50 | 100 | 150
<window_len>  ::= 8 | 16 | 24 | 32
<window_step> ::= 1 | 2 | 4 | 8 | 16

<conv_lys>    ::= <conv> | <conv> <conv> | <conv> <conv> <conv> |
                  <conv> <conv> <conv> <conv> | <conv> <conv> <conv> <conv> <conv> |
                  <conv> <conv> <conv> <conv> <conv> <conv>
<conv>        ::= <n_kernels> <k_size> <act_fn> <pooling>
<n_kernels>   ::= 8 | 16 | 32 | 64 | 128 | 256
<k_size>      ::= 2 | 3 | 4
<pooling>     ::= null | <p_size>
<p_size>      ::= 2 | 3

<dense_lys>   ::= <dense> | <dense> <dense> | <dense> <dense> <dense>
<dense>       ::= <d_type> <n_units> <act_fn> <reg_fn> <dropout_r>
<d_type>      ::= rnn | lstm | gru | feedforward
<n_units>     ::= 256 | 512 | 1024
<act_fn>      ::= relu | linear
<reg_fn>      ::= null | l1 | l2 | l1l2
<dropout_r>   ::= 0 | 0.5

<opt_setup>   ::= <opt_type> <learn_rate>
<opt_type>    ::= adam | adamax
<learn_rate>  ::= 1E-2 | 5E-3 | 1E-3 | 5E-4
```

Figure 5.36: *Definition of the grammar in Backus-Naur Form for the OPPORTUNITY dataset.*

5.4.4.2 Grammatical Evolution

Figure 5.36 shows the definition of grammar used for generating individuals in the OPPORTUNITY problem, in Backus-Naur Form (BNF).

As it happened in the MNIST scenario, the grammatical evolution encoding is more flexible as we are now allowed to declare a variable number of values for each hyperparameter (instead of powers of two like in the case of the GA). This time, we have decided to remove uncommon values in the winning individuals of the GA (see section 5.4.6.1), to reduce the search space and accelerate convergence. For example, we will support up to six convolutional and three dense layers, limit the learning rules to only Adam and AdaMax adjusting the learning rates accordingly, and reduce the allowed values for the input setup (sliding window length and step), since small values have shown to work better in all cases in the GA.

5.4.5 Experimental Setup

In this section we will describe the experimental setup. All hyperparameters not covered in this section remain as described in the previous chapter. The fitness function to be maximized is the F1 score over the test set of the convolutional neural network whose architecture is defined by the individual phenotype. F1 score is the main metric reported in the state of the art for OPPORTUNITY.

As we did with MNIST, we will train the CNNs using a reduced version of the dataset. This time, since OPPORTUNITY dataset is larger and experiments require more time to complete, we have decided to use less training data. In particular, 5 epochs will be run, with a randomly-chosen 5 % of the training set each. This will enable us to complete the experiments in an acceptable amount of time, with the fitness function being a reasonable proxy.

5.4.6 Results

In this section the results will be discussed, both for the genetic algorithm and for grammatical evolution. First, we will describe the outcome of the optimization process, and later will check how fully-trained individuals perform. Finally, we will build committees of neural networks and evaluate their performance.

5.4.6.1 Genetic Algorithm

Figure 5.37 shows how the median fitness has evolved over time for one of the 10 runs of the GA. The improvement in the fitness function (F1 score) is slow during the first 40 generations and then accelerates slowly, reaching a stable plateau after 60 generations around an F1-score of about 0.8.

The fitness and topology of the top 7 individuals in the hall-of-fame are shown in table 5.15. Again, symbols 'c' and 'd' span the convolutional and the dense layers respectively, in order to ease their identification. The best individual involves a fitness value of 0.9030, which correspond to the F1-score resulting from evaluating the CNN when a simplified training process comprising only 5 epochs and 5 % of the training data is used. Whereas the best individual is the only one with a fitness over 0.9, the next individuals perform similarly, with a fitness no lower than 0.8962 for the individual ranking the 7th position in the hall-of-fame.

Now, when interpreting the phenotypes of these individuals, we can draw some interesting conclusions regarding the CNN topologies they represent. This is specially true if we remember that specific mechanisms were implemented in order to enhance the genetic diversity (niching and repetition of experiments with different initial populations), and thus consistency between different individuals in the hall-of-fame can be interpreted as a sign that those values perform well.

For example, the batch size does not seem to be a critical factor. On the other hand, the sliding window size and step is unanimous across all individuals: a small window of only 8 samples is always preferred, with a window step of 1.

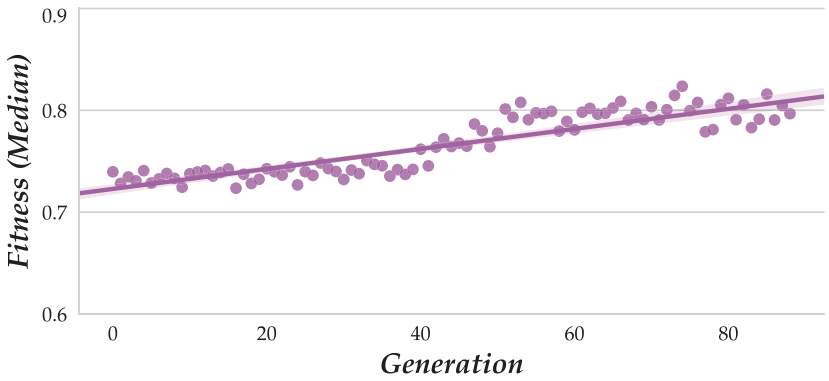


Figure 5.37: *Evolution of the median fitness in one run of the GA in OPPORTUNITY.*

#	Fitness		Architecture				
1	0.9030		$B = 25$	$w = 8$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.001$
		c	$ck_1 = 64$	$cs_1 = 2$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 256$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 64$	$cs_3 = 2$	$cp_3 = 1$	$ca_3 =$ ReLU	
		c	$ck_4 = 32$	$cs_4 = 2$	$cp_4 = 1$	$ca_4 =$ ReLU	
		c	$ck_5 = 8$	$cs_5 = 2$	$cp_5 = 1$	$ca_5 =$ linear	
		c	$ck_6 = 128$	$cs_6 = 2$	$cp_6 = 1$	$ca_6 =$ ReLU	
		d	$dt_1 =$ GRU	$dn_1 = 512$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
2	0.8993		$B = 25$	$w = 8$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.001$
		c	$ck_1 = 128$	$cs_1 = 2$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 32$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ linear	
		c	$ck_3 = 2$	$cs_3 = 2$	$cp_3 = 1$	$ca_3 =$ linear	
		c	$ck_4 = 16$	$cs_4 = 2$	$cp_4 = 1$	$ca_4 =$ ReLU	
		d	$dt_1 =$ LSTM	$dn_1 = 512$	$dd_1 = 0$	$da_1 =$ linear	$dr_1 =$ none
3	0.8992		$B = 25$	$w = 8$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.0005$
		c	$ck_1 = 8$	$cs_1 = 2$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 128$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ linear	
		c	$ck_3 = 16$	$cs_3 = 2$	$cp_3 = 1$	$ca_3 =$ linear	
		c	$ck_4 = 32$	$cs_4 = 2$	$cp_4 = 1$	$ca_4 =$ ReLU	
		c	$ck_5 = 16$	$cs_5 = 2$	$cp_5 = 1$	$ca_5 =$ ReLU	
		c	$ck_6 = 16$	$cs_6 = 2$	$cp_6 = 1$	$ca_6 =$ linear	
		d	$dt_1 =$ GRU	$dn_1 = 512$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
		d	$dt_2 =$ LSTM	$dn_2 = 1024$	$dd_2 = 0$	$da_2 =$ linear	$dr_2 =$ none
4	0.8978		$B = 100$	$w = 8$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.0005$
		c	$ck_1 = 8$	$cs_1 = 2$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 256$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ linear	
		c	$ck_3 = 8$	$cs_3 = 2$	$cp_3 = 1$	$ca_3 =$ linear	
		c	$ck_4 = 32$	$cs_4 = 2$	$cp_4 = 1$	$ca_4 =$ linear	
		c	$ck_5 = 64$	$cs_5 = 2$	$cp_5 = 1$	$ca_5 =$ ReLU	
		c	$ck_6 = 2$	$cs_6 = 2$	$cp_6 = 1$	$ca_6 =$ linear	
		d	$dt_1 =$ GRU	$dn_1 = 512$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
		d	$dt_2 =$ LSTM	$dn_2 = 1024$	$dd_2 = 0$	$da_2 =$ linear	$dr_2 =$ none
5	0.8968		$B = 25$	$w = 8$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.001$
		c	$ck_1 = 32$	$cs_1 = 2$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 32$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ ReLU	
		d	$dt_1 =$ GRU	$dn_1 = 512$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
6	0.8966		$B = 50$	$w = 8$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.001$
		c	$ck_1 = 32$	$cs_1 = 2$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 4$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ linear	
		c	$ck_3 = 8$	$cs_3 = 2$	$cp_3 = 1$	$ca_3 =$ linear	
		c	$ck_4 = 128$	$cs_4 = 2$	$cp_4 = 1$	$ca_4 =$ ReLU	
		c	$ck_5 = 8$	$cs_5 = 2$	$cp_5 = 1$	$ca_5 =$ linear	
		d	$dt_1 =$ LSTM	$dn_1 = 1024$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
		d	$dt_2 =$ LSTM	$dn_2 = 512$	$dd_2 = 0$	$da_2 =$ linear	$dr_2 =$ none
7	0.8962		$B = 25$	$w = 8$	$w_{step} = 1$	$f =$ AdaMax	$\eta = 0.001$
		c	$ck_1 = 2$	$cs_1 = 2$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 256$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ linear	
		c	$ck_3 = 64$	$cs_3 = 2$	$cp_3 = 1$	$ca_3 =$ ReLU	
		d	$dt_1 =$ GRU	$dn_1 = 1024$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none

Table 5.15: *Architecture and fitness of the top 7 individuals in the hall-of-fame for GA in the OPPORTUNITY Gestures dataset.*

Additionally, a large number of convolutional layers is always preferred: three individuals have 6 convolutional layers, one has 5, and another has 4. These models would be extracting very high level features from the raw data in order to increase the classification performance. The other two have three and two respectively, thus showing that the network can still perform properly even with less layers. One convolutional layer; however, seems insufficient to achieve this task.

Regarding the number of dense layers, all individuals implement only one or two of them. This happens because during experimentation, a larger number of layers increased significantly the network complexity and the models did not fit in memory, thus being unable to run and attaining a fitness of 0. Nevertheless, it seems clear that few dense layers can still lead to proper classification; in fact, the two best individuals have only one dense layer.

Also, there is an additional realization at this point: all dense layers are recurrent. This contrasts with the results of MNIST, and strongly suggests that recurrent layers are required to capture the temporal knowledge present in the input data. Additionally, classic recurrent layers were not used at all, as the models only comprised LSTM or GRU layers, something that can be explained as these cells are more effective and efficient than classical recurrent approaches.

The convolutional layers present a very diverse number of kernels, but always involve patches of size 2 and no pooling. This is consistent with the fact that small window lengths are preferred, and thus larger patches or pooling could effectively consume the whole input. The number of neurons in the dense layers is always large, in the order of 512 or 1024.

All topologies include at least one convolutional layer and one dense layer with a non-linear activation function (ReLU). As it happened with MNIST, classification may be unfeasible when only linear transformations are considered. In this case, regularization is not applied in any of the models, neither L1 or L2 nor dropout. This could be a sign that overfitting is not happening during the optimization task, even if the training data size was reduced by sampling and despite its large dimensionality comprising 113 channels. As a result, overfitting should not occur either when a larger training dataset be used.

#	Mean	Std. Dev.	Median	Minimum	Maximum
1	0.909200	0.001571	0.90895	0.9065	0.9139
2	0.907715	0.002269	0.90800	0.9032	0.9115
3	0.909120	0.001787	0.90875	0.9057	0.9122
4	0.903210	0.001604	0.90275	0.9008	0.9062
5	0.906010	0.001989	0.90590	0.9016	0.9097
6	0.908795	0.001782	0.90830	0.9059	0.9120
7	0.908575	0.001600	0.90865	0.9049	0.9115
8	0.909540	0.001749	0.90955	0.9067	0.9132
9	0.906860	0.002531	0.90660	0.9034	0.9135
10	0.909850	0.001613	0.90955	0.9078	0.9133
11	0.905040	0.001539	0.90545	0.9006	0.9077
12	0.908240	0.001498	0.90835	0.9050	0.9108
13	0.905730	0.002349	0.90515	0.9019	0.9106
14	0.909725	0.001374	0.90960	0.9073	0.9129
15	0.906910	0.002202	0.90740	0.9032	0.9102
16	0.908360	0.001662	0.90830	0.9057	0.9120
17	0.910105	0.001856	0.90955	0.9062	0.9140
18	0.911280	0.001906	0.91170	0.9065	0.9149
19	0.910655	0.001480	0.91045	0.9085	0.9135
20	0.909135	0.001974	0.90925	0.9033	0.9143

Table 5.16: *Summary of F1 scores of the best 20 GA individuals after full training in the OPPORTUNITY Gestures dataset.*

Finally, regarding the learning rule, the choice is in most cases Adam, with a single exception preferring AdaMax, pointing out that these optimizers are more suitable for achieving an accurate classification. The learning rate oscillates between $\eta = 1 \cdot 10^{-3}$ and $\eta = 1 \cdot 10^{-4}$.

After re-training each of the top-20 topologies in the hall-of-fame using the entire training set and 30 epochs, the F1 scores described in table 5.16, and depicted in figure 5.38, were obtained. The maximum value reports an F1 score of 0.9149, similar to the second best of the state of the art [277] (0.915), but still substantially worst than the best result reported [258] (0.929).

After the full training of the top-20 topologies, we have built committees of CNNs. The resulting F1 scores for each ensemble are shown in figure 5.39. The performance stabilizes after 11 CNNs are introduced into the committee, and the F1 score reaches a peak of 0.9245 when it comprises 15 CNNs. This would rank the third place, just below the works by Hammerla et al. [138] and

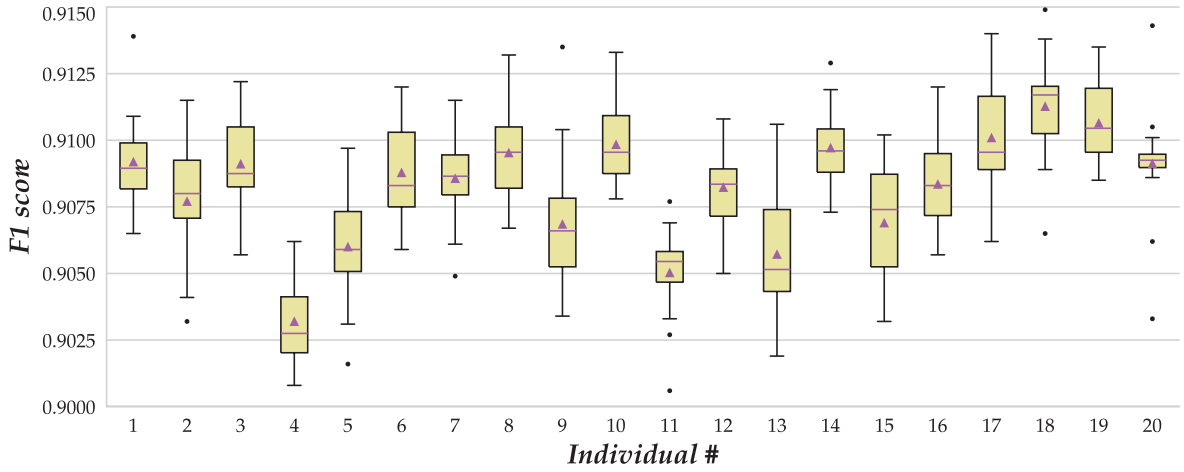


Figure 5.38: *Boxplot showing the distribution of F1 scores of the best 20 GA individuals after full training in OPPORTUNITY Gestures.*

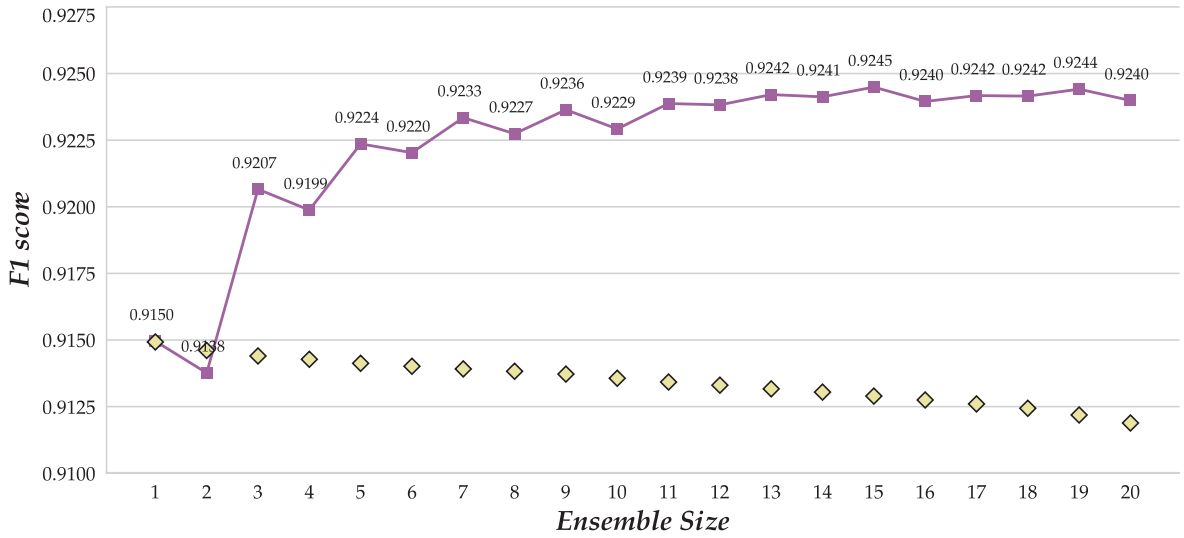


Figure 5.39: *Evolution of the F1 scores of the incremental ensembles using the best 20 individuals from the GA with OPPORTUNITY Gestures.*

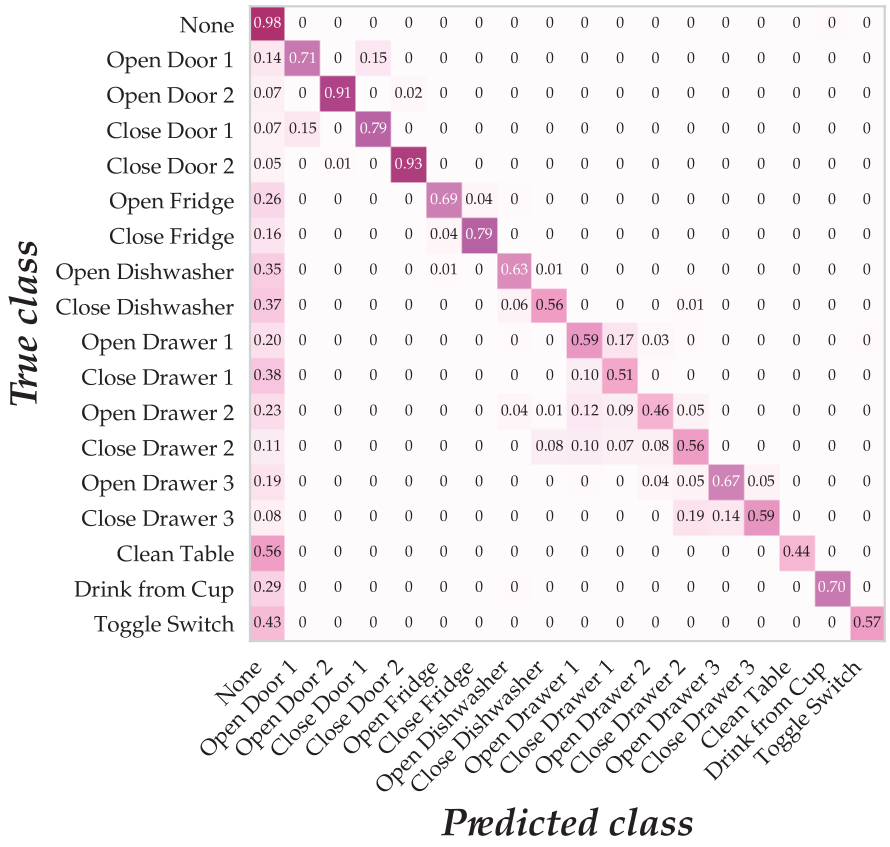


Figure 5.40: *Confusion matrix of the best ensemble using GA individuals with OPPORTUNITY.*

Moya Rueda and Fink [258], who reported respectively F1 scores of 0.927 and 0.929, and almost one percentage point above the result by Ordoñez and Roggen [277], who attained an F1 score of 0.915.

Finally, figure 5.40 displays the confusion matrix for the OPPORTUNITY Gestures dataset using the best ensemble obtained after optimization with the genetic algorithm. Since the classes distribution is unbalanced, we have normalized the values per class in order to ease its visualization. Numbers in a row may not sum up to one due to rounding.

It can be seen that most classes are sometimes classified as "null" (no action). However, this is more frequent in those actions related to opening and closing drawers, and specially noticeable for the gestures of toggle switching and cleaning the table.

While it is not easy to identify why these actions are poorly recognized, intuition suggests that toggle switching may require very few motion; and therefore the system may fail to recognize that the subject has performed an action at all, whereas cleaning the table may be an activity performed differently across subjects since there were not specific instructions on how to clean the table or how much surface needed to be cleaned. On the other hand, actions related with opening and closing the doors and the fridge show a higher level of accuracy; and it is worth noting that these actions are generally performed very similarly across different subjects.

Besides from actions misclassified as "null", other mistakes seem easier to explain. In most cases, opening a certain object is sometimes confused with closing it, and viceversa. For example, 91 % of the times the gesture "open the door 2" is correctly classified, but 2 % of the times it is misclassified as closing the door 2; and on the other direction, 93 % of the times closing the door 2

is correctly classified, with 1 % of the times being misclassified as opening it. For some reason, this mistake becomes more obvious in the case of the first door, where the action is classified correctly about 70-80 % of the times, and incorrectly 15 % of the times.

To different extents, this happens consistently with all objects. However, the confusion is larger in the case of the drawers, where misclassification can involve a different drawer. For example, closing the drawer 3 is recognized 19 % of the times as closing the drawer 2. These mistakes may be acceptable, because the three drawers are located close to each other in the same piece of furniture [58] and the body actions required to open and close them may be similar.

5.4.6.2 *Grammatical Evolution*

Figure 5.41 shows how the median fitness has evolved over time for one of the 10 runs of the GE. The improvement in the fitness function (F1 score) is very subtle across all the generations, and it could be noticed more strongly in the case of the GA. This could be due to the fact that the search space is smaller in the GE. Also, we can see that the median fitness is smaller that in the case of the GA. We have not found a proper explanation to this effect but, as we will see later, the maximum fitnesss are higher when evolving individuals with grammatical evolution.

Table 5.17 shows the topology and fitness (F1 score) of the top 7 individuals of the hall-of-fame. The first relevant aspect we can notice is that the fitness is consistently better than in the genetic algorithm, given that in this case all individuals in the top 7 have a fitness higher than 0.9 (as opposed of only one in the GA).

In many cases, the topologies found after optimization with grammatical evolution are similar to those found with the genetic algorithm. For example, the number of convolutional layers oscillates between 2 and 5, and in all cases there is at least one layer with 128 or 256 convolutional kernels. Small kernel sizes, no larger than 4 positions, are preferred. Pooling is used in some cases, but there seems not to be a trend regarding its presence in the topology.

Regarding fully connected layers, only one or two layers are considered in the topologies. As it happened in the GA case, all these layers observe some recurrent behavior, since they implement either LSTM or GRU cells with no exception. This is consistent with our expectations, given that input data is a time series. The number of hidden units is always either 512 or 1024, a fact that could point out that a smaller number of units is not suitable to properly classify instances. L1 or L2 regularization is never used, and dropout is only found in one case.

All individuals comprise at least one layer with a non-linear activation function (ReLU). This is expected behavior, since this is a complex dataset and the output cannot be approximated using only linear transformations.

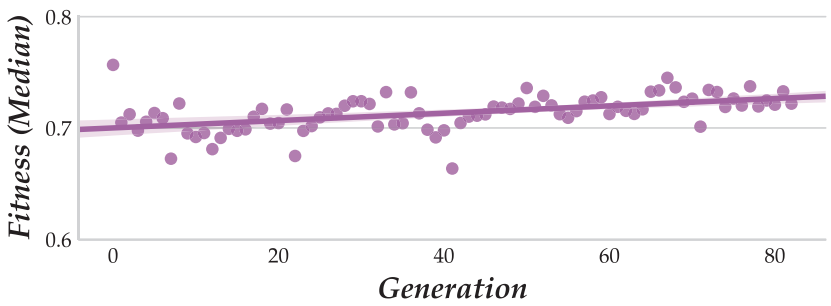


Figure 5.41: *Evolution of the median fitness in one run of the GE in OPPORTUNITY.*

#	Fitness		Architecture				
1	0.9094		$B = 25$	$w = 32$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.001$
		c	$ck_1 = 64$	$cs_1 = 4$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 128$	$cs_2 = 3$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 16$	$cs_3 = 2$	$cp_3 = 3$	$ca_3 =$ ReLU	
		c	$ck_4 = 8$	$cs_4 = 2$	$cp_4 = 1$	$ca_4 =$ linear	
		c	$ck_5 = 32$	$cs_5 = 4$	$cp_5 = 2$	$ca_5 =$ ReLU	
		d	$dt_1 =$ LSTM	$dn_1 = 1024$	$dd_1 = 0$	$da_1 =$ linear	$dr_1 =$ none
2	0.9037		$B = 25$	$w = 32$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.001$
		c	$ck_1 = 256$	$cs_1 = 2$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 32$	$cs_2 = 4$	$cp_2 = 3$	$ca_2 =$ ReLU	
		d	$dt_1 =$ GRU	$dn_1 = 512$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
3	0.9031		$B = 50$	$w = 32$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.0005$
		c	$ck_1 = 8$	$cs_1 = 3$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 128$	$cs_2 = 2$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 128$	$cs_3 = 3$	$cp_3 = 1$	$ca_3 =$ linear	
		c	$ck_4 = 64$	$cs_4 = 4$	$cp_4 = 1$	$ca_4 =$ ReLU	
		d	$dt_1 =$ GRU	$dn_1 = 512$	$dd_1 = 0$	$da_1 =$ linear	$dr_1 =$ none
4	0.9025		$B = 50$	$w = 32$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.001$
		c	$ck_1 = 256$	$cs_1 = 2$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 128$	$cs_2 = 3$	$cp_2 = 1$	$ca_2 =$ ReLU	
		d	$dt_1 =$ GRU	$dn_1 = 512$	$dd_1 = 0.5$	$da_1 =$ linear	$dr_1 =$ none
5	0.9013		$B = 25$	$w = 32$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.0005$
		c	$ck_1 = 16$	$cs_1 = 3$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 256$	$cs_2 = 3$	$cp_2 = 3$	$ca_2 =$ linear	
		c	$ck_3 = 32$	$cs_3 = 2$	$cp_3 = 1$	$ca_3 =$ ReLU	
		d	$dt_1 =$ GRU	$dn_1 = 512$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
6	0.9013		$B = 25$	$w = 32$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.0005$
		c	$ck_1 = 16$	$cs_1 = 3$	$cp_1 = 1$	$ca_1 =$ ReLU	
		c	$ck_2 = 256$	$cs_2 = 3$	$cp_2 = 3$	$ca_2 =$ linear	
		c	$ck_3 = 32$	$cs_3 = 2$	$cp_3 = 1$	$ca_3 =$ ReLU	
		d	$dt_1 =$ GRU	$dn_1 = 512$	$dd_1 = 0$	$da_1 =$ ReLU	$dr_1 =$ none
7	0.9010		$B = 25$	$w = 32$	$w_{step} = 1$	$f =$ Adam	$\eta = 0.001$
		c	$ck_1 = 64$	$cs_1 = 4$	$cp_1 = 1$	$ca_1 =$ linear	
		c	$ck_2 = 128$	$cs_2 = 3$	$cp_2 = 1$	$ca_2 =$ ReLU	
		c	$ck_3 = 16$	$cs_3 = 2$	$cp_3 = 3$	$ca_3 =$ ReLU	
		c	$ck_4 = 8$	$cs_4 = 2$	$cp_4 = 1$	$ca_4 =$ linear	
		c	$ck_5 = 32$	$cs_5 = 4$	$cp_5 = 2$	$ca_5 =$ ReLU	
		d	$dt_1 =$ LSTM	$dn_1 = 1024$	$dd_1 = 0$	$da_1 =$ linear	$dr_1 =$ none

Table 5.17: *Architecture and fitness of the top 7 individuals in the hall-of-fame for GE in the OPPORTUNITY Gestures dataset.*

Interestingly, the preferred window length in this case is of 32, whereas it was 8 in the GA setup. This value is found consistently in all individuals. The window step; however, is always of 1, which makes sense since it involves introducing more data to the convolutional neural network. The batch size is always either 25 or 50. Even when the GA had one individual with a batch size of 100 samples, in this case values larger than 50 are not found. This may be due to the fact that, since windows are larger, there is a chance that the dataset does not fit in GPU memory.

Finally, the chosen optimizer is always Adam, with a learning rate oscillating between $\eta = 1 \cdot 10^{-3}$ and $\eta = 5 \cdot 10^{-4}$. Again, this optimizer seems to be the best choice for achieving high F1 scores, at least when a small number of epochs is imposed.

Later, each of the top 20 topologies from the hall-of-fame are retrained using the full training set (without sampling) and 30 epochs, as we had previously done for the MNIST and EMNIST databases. This full training stage is repeated 20 times for each topology, computing and recording the F1 score over the test set. A statistical summary of these F1 scores for each topology is gathered in table 5.18, and their boxplots are depicted in figure 5.42.

#	Mean	Std. Dev.	Median	Minimum	Maximum
1	0.912150	0.001528	0.91235	0.9091	0.9148
2	0.907910	0.002371	0.90735	0.9029	0.9114
3	0.911230	0.002044	0.91185	0.9066	0.9143
4	0.909540	0.001801	0.91020	0.9065	0.9116
5	0.907695	0.002126	0.90740	0.9040	0.9110
6	0.907805	0.002299	0.90820	0.9030	0.9121
7	0.912155	0.001004	0.91215	0.9102	0.9143
8	0.912635	0.002196	0.91295	0.9079	0.9175
9	0.910900	0.001453	0.91065	0.9090	0.9146
10	0.911030	0.002295	0.91180	0.9058	0.9138
11	0.907885	0.002182	0.90755	0.9043	0.9122
12	0.907995	0.002917	0.90790	0.9040	0.9140
13	0.912040	0.002446	0.91135	0.9072	0.9177
14	0.898005	0.005668	0.89610	0.8918	0.9076
15	0.911005	0.001863	0.91100	0.9070	0.9139
16	0.910800	0.001365	0.91070	0.9082	0.9136
17	0.910945	0.001438	0.91075	0.9085	0.9144
18	0.910730	0.004045	0.91180	0.9005	0.9185
19	0.911695	0.001979	0.91175	0.9091	0.9176
20	0.912015	0.002713	0.91250	0.9047	0.9152

Table 5.18: *Summary of F1 scores of the best 20 GE individuals after full training in the OPPORTUNITY Gestures dataset.*

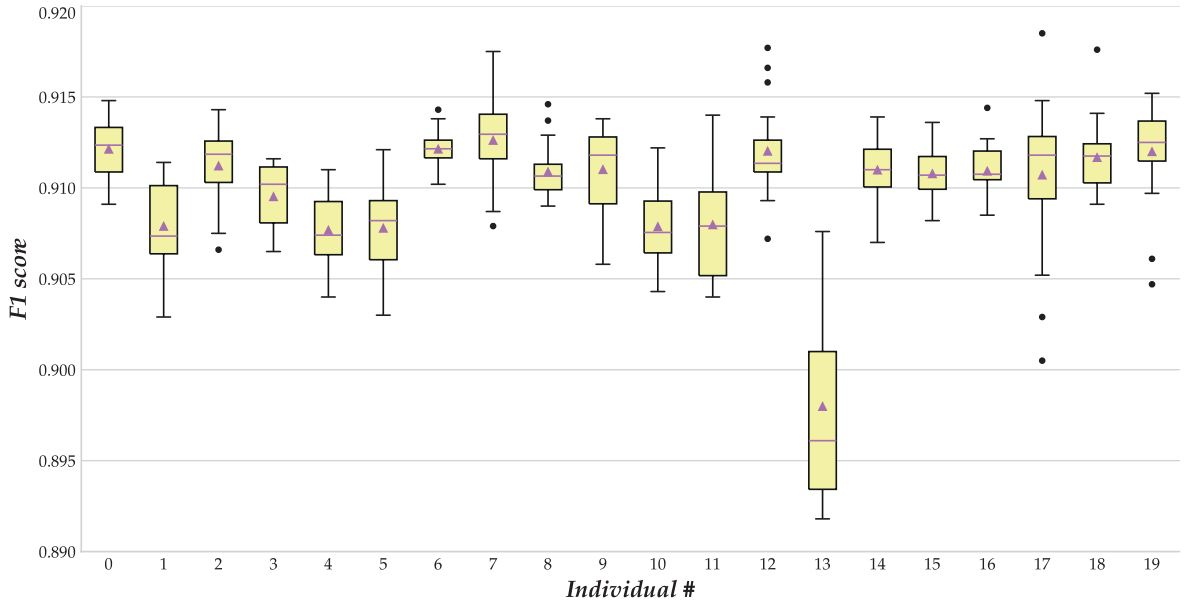


Figure 5.42: *Boxplot showing the distribution of F1 scores of the best 20 GE individuals after full training in OPPORTUNITY Gestures.*

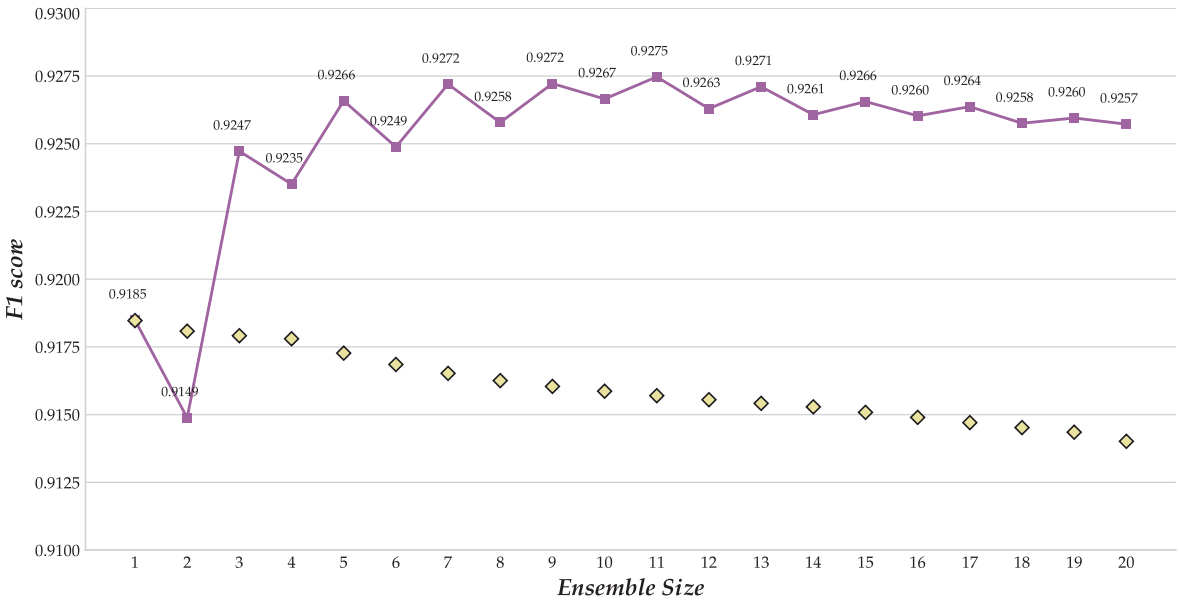


Figure 5.43: *Evolution of the F1 scores of the incremental ensembles using the best 20 individuals from the GE with OPPORTUNITY Gestures.*

The obtained F1 scores are consistently better than those previously obtained with genetic algorithms. In this case, the best individual has an F1 score of 0.9185 (versus 0.9149 with GAs), which is significantly better than the score reported by Ordoñez and Roggen [277] (0.915) but still far from the best result of the state of the art [258] (0.929).

Then, we will build committees of CNNs out of these individuals, following the same procedure that we have used before. The performance of these committees as new models are added to them is shown in figure 5.43. Again, the F1 score shows a large variance with small committees, but stabilizes after more than ten CNNs have been added to it. In this case, the best committee found comprises 11 models, and attains an F1 score of 0.9275. This ensemble would slightly exceed the F1 score reported by Hammerla et al. [138]; reaching the second position in the ranking slightly after the result reported by Moya Rueda and Fink [258] (0.929), whose work is so far only available in a pre-print repository.

Finally, figure 5.44 shows the confusion matrix corresponding to the best committee found for the OPPORTUNITY Gestures dataset. Again, values per class are normalized in order to ease its visualization and they may not sum up to one due to rounding.

The results are consistent with those reported previously with the genetic algorithm (see figure 5.40). Most of the activities are, to a greater or lesser degree, misclassified as "null". This is specially noticeable in the "cleaning table" task.

Again, a common confusion seems that of mixing up opening and closing actions. However, in this case, this happens mostly with the door 1, whereas the second door does not seem affected by this phenomena. Also, the system sometimes fails to discern between drawers 1 and 2. In fact, the only activity with a recall lower than 50 % is opening the second drawer (43 %), and it is remarkable than 21 % of the times it is recognized as opening the first drawer. This seems a common mistake, easily explainable by the fact that drawers are located next to each other.

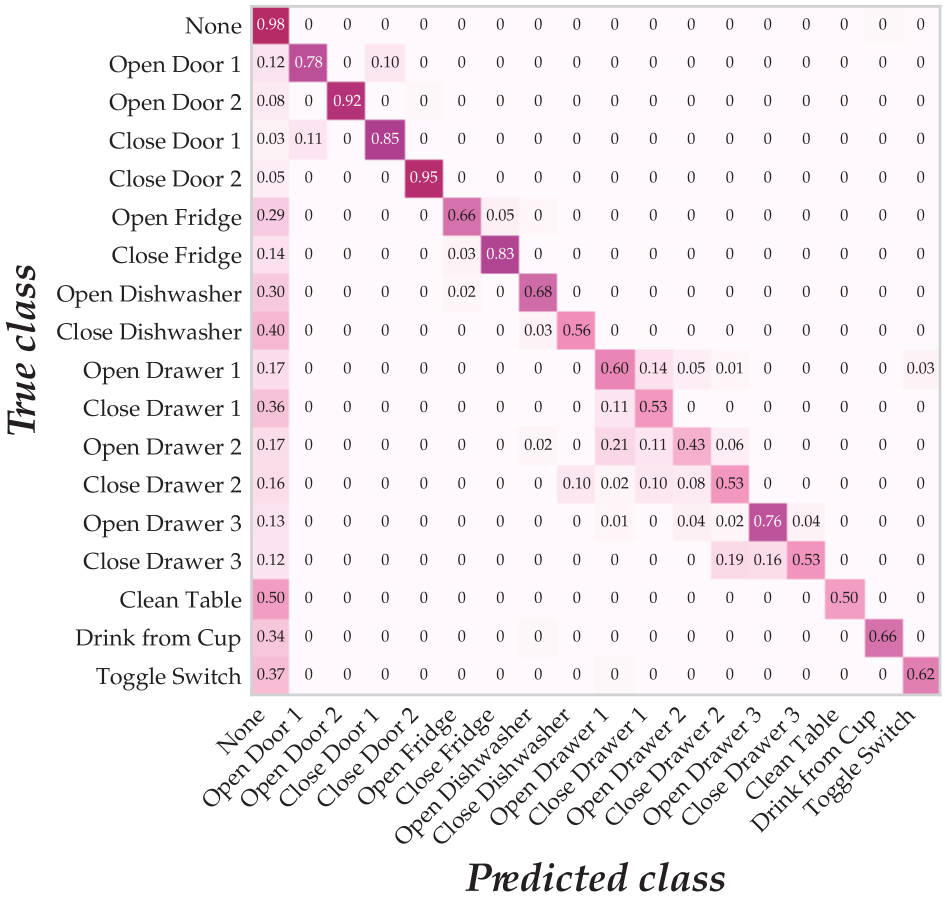


Figure 5.44: *Confusion matrix of the best ensemble using GE individuals with OPPORTUNITY.*

5.5 Summary

In this section, we have carried out experiments in order to validate our working hypotheses. In particular, we have used two different evolutionary computation techniques in order to automatically evolve the topology of convolutional neural networks; namely, genetic algorithms and grammatical evolution. In all cases, the genetic algorithm was used for performing a broader exploration; whereas grammatical evolution allowed us to specialize the search in those areas that were best ranked by the GA, removing uncommon configurations. For both techniques, a niching strategy was included in order to preserve genetic diversity.

These techniques were used against two different datasets: MNIST, which is a extensively used database in the computer vision field comprising handwritten digits, and OPPORTUNITY, which is a human activity dataset. Data augmentation has not been performed in any of the datasets. The fitness function of the evolutionary algorithms was established as a proxy of a quality metric over the test set (accuracy in MNIST and F1 score in OPPORTUNITY), using a reduced version of the training set via random sampling and a very small number of epochs. Results from 10 different runs of each technique were executed and the best individuals were serialized in a hall-of-fame.

Then, the top-20 individuals from the hall-of-fame were subjected to a full training, by re-training their topologies using the whole training set and a much larger number of epochs to enforce convergence. This process is repeated 20 times for each topology, obtaining as a result

20 models, each one with an associated quality metric. Some of the resulting models are highly competitive when compared to the state of the art; for example, the best test error rate for MNIST is 0.37 % and the best F1 score for OPPORTUNITY is 0.9185.

Later, we have combined these models in order to build committees (or ensembles) of CNNs. Our initial hypothesis was that, since the evolutionary algorithms preserved the genetic variability due to the niching scheme, committees would perform better than individual models. The idea behind committees is that while some models may fail at recognizing some particular examples, other models will succeed. Thus, it is interesting to have a diverse set of models.

In order to build the committees, we have sorted the best model found for each topology in descending order of performance. Then, models were added one-by-one to the committee. The resulting performance was measured and the best committee was serialized. The results are very promising for the two datasets used in this thesis: the best test error rate for MNIST is 0.28 %, and the best F1 score for OPPORTUNITY is 0.9275. These results are indeed highly competitive, and would rank among the top positions in the ranking of state-of-the-art works. In fact, both results would head the ranking if results reported in papers published in pre-print repositories (and therefore not submitted for peer review) are not included in the ranking.

Finally, we have tested an approach to transfer learning by reusing the topologies learned for the MNIST dataset with a new database, namely EMNIST, recently introduced in 2017. EMNIST has two datasets: Digits and Letters, and both of them share structure with the MNIST dataset. The former is the exact same domain than in MNIST, yet obtained for a different data source; whereas the second is a different domain, containing handwritten letters instead of digits. The results obtained after the whole process, which again involves full training of the models and the construction of committees of CNNs, show that the learned topologies can be reused successfully. The best accuracy found for EMNIST Digits is 99.7725 % and for EMNIST Letters is 95.35 %. To the best of our knowledge, the result for EMNIST Digits is the best found so far in the state of the art, whereas the result of EMNIST Letters is only slightly outperformed by that reported by Peng and Yin (95.44 %). In any case, it is worth noting that this dataset has not been extensively used as a benchmark as of the time of writing.

In summary, the whole process shows that the topologies can be optimized, leading to automatically-designed architectures that, upon an exhaustive training process, are able to produce very competitive models. Additionally, these models can be combined to form a committee, which can improve the performance even more due to the natural diversity found in these models. Finally, topologies can be reused from one problem to another with an equivalent representation, even when both problems involve different domains or data sources.

Chapter 6

Conclusions and Future Work

Almost three decades ago, neuroevolution arose in order to use evolutionary algorithms to optimize some aspects of neural networks. In some cases, weights were evolved because classical backpropagation was insufficient (e.g., in the case of reinforcement learning, when a loss function could not be computed due to the non-availability of labeled data). However, as soon as in the late 1980s some works started to use neuroevolution for determining optimal topologies of a neural network. This was an interesting field of research, given that most rules-of-thumb suggesting how to determine good topologies for artificial neural networks were useless.

Nowadays, times have changed significantly. While most of the classical works on neuroevolution focused on architectures with one hidden layer, nowadays new discoveries in machine learning and the advancements on hardware architectures (specially in GPUs) have led to the rise of deep learning, which often involves massive use of convolutional neural networks to solve a variety of problems. Today, this kind of networks have become an industry standard for solving many machine learning applications.

However, convolutional neural networks involve much more complex topologies than traditional feed-forward networks. They can contain up to dozens of convolutional layers which are responsible for extracting significant features from raw data. These convolutional layers will have a certain number of filters (or kernels) of a given size, and with a given activation function. Also, pooling can be used to reduce the dimensionality as data passes through convolutional layers. Eventually, the output of the last convolutional layer will be flattened and introduced to a dense network, involving several feed-forward or recurrent layers, with specific setups regarding the number of neurons, their activation function or their regularization configuration.

Because convolutional neural networks are extensively used today and have been successfully applied to very diverse domains, and since their topology can be very complex and is hard and expensive to be designed by hand; neuroevolution shows as a promising field to perform automatic and evolutionary design of these topologies.

Unfortunately, because of the important cost associated to the computation of the fitness function, which involves training a convolutional neural network and testing it against a validation set, this field remains mostly unexplored. Most scientific production in this area has taken place in 2017 and 2018, and these works often impose many constraints on the topologies in order to reduce the size of the search space and accelerate evolution.

To the best of our knowledge, this is the most complete work up to date in neuroevolution of convolutional neural networks. In this thesis, we have proposed the use of two evolutionary computation techniques, namely genetic algorithms and grammatical evolution, in order to evolve most aspects of the network topology. In order to accelerate the evolutionary process, we have

developed a simplified fitness function which provides an approximation of the network performance, without requiring a full training process. Additionally, we have used a niching scheme to preserve diversity in the population, which has the additional advantage of enabling the building of committees (ensembles) of convolutional neural networks.

Our proposal has been validated against several datasets. The first of them is MNIST, a well-known dataset for handwritten digit recognition, attaining highly competitive results. In fact, we have achieved a test error rate that is only outperformed by two related works. Also, ensembles have been proved to outperform individual models.

Later, we have reused the best topologies found in EMNIST, a newly introduced extension of MNIST which includes more samples and a new datasets of letters. These topologies have been proved successful in the new dataset, obtaining results better than any other work in the state of the art, and pointing out that topologies can be reused among domains of similar characteristics. To better understand the models mistakes, we have plotted the misclassified samples, so that the reader can check how they involve hardly-recognizable digits or characters.

Finally, we have also tested the neuroevolutionary process against OPPORTUNITY, a dataset with labelled human activity information which uses high dimensional data gathered from a variety of sensors and whose objective is to recognize the gestures performed by the user. The F1 score attained using grammatical evolution to evolve an ensemble of CNNs have outperformed many of the best results reported so far in related works, placing our approach by the top of the ranking.

Upon these discoveries, we consider the working hypotheses of this thesis validated, and the proposed objectives successfully achieved. In summary, evolutionary computation has proven to be successful when optimizing the topology of convolutional neural networks, leading to results better than those reported on the state of the art with hand-made architectures. In the remainder of this chapter, we summarize the validated hypothesis and objectives, and discuss future work to further extend this research line.

6.1 Validation of Working Hypotheses

Upon the completion of this thesis, we consider the following working hypotesis validated:

> *Manual design of convolutional neural networks is a time-consuming task and leads to topologies which are far from the optimal.*

When surveying the state of the art, we have found that many authors agree on saying that the design of neural network topologies is mostly a matter of trial-and-error. Therefore, it is inherently a time-consuming task, since the time required to train and evaluate a neural network is not negligible. The lack of rules of thumb to determine optimal (or even good) topologies suggests that researchers have to spend time and effort in trying several designs before finding one that meets their needs. And once such a topology is found, there are no guarantees that it conforms an optimal solution, but rather one that is good enough.

> *Topology design of CNNs can be approached as an optimization problem.*

In this thesis, we have validated this hypothesis by building a solution that tackles the design of CNN architectures as an optimization problem. In this problem, a quality metric over a validation set must be maximized (or minimized if the metric is a measurement of the error). Therefore, different architectures can be compared against the others by means of this quality metric.

Metaheuristics, and in particular evolutionary computation techniques, can be used for automatic optimization of CNN topologies in an affordable amount of time.

We have shown how evolutionary computation is suitable for evolving the design of CNNs. The time has been reduced significantly by using a fitness function that avoids a full training of the network, but is still capable of comparing diferent topologies to evolve the population.

Evolved topologies will in average outperform manually-designed topologies, or, in the worst case, lead to similar results requiring less effort.

We have seen how the CNNs evolved in this thesis are very competitive with the state of the art in datasets belonging to different domains, heading the ranking in some cases. This confirms the fact that the evolutionary design of neural network is able to achieve architectures that compete with the best manually-designed ones, outperforming them in many cases.

Committees of convolutional neural networks will often perform better than a single neural network, and thus if several good solutions are found they can be used to build a committee.

In this thesis we have proved that committees of CNNs unanimously perform better than individual models as long as the ensemble is comprised of three or more models. It is worth noting that due to the introduction of niches in the evolutionary algorithms, the individuals conforming the ensembles are built with diversity in mind.

A system can be designed to automatically optimize the topology of CNNs given a supervised learning problem in an effective and efficient manner.

This hypothesis can be considered validated given that the aforementioned system has been indeed implemented and evaluated in this dissertation.

6.2 Validation of Objectives

After completing this thesis, we consider that all the proposed objetives have been achieved:

(1) *Proposal of one or more domain-agnostic encodings that model the topology of a CNN in a way that can be evolved using specific evolutionary computation techniques.*

A general framework for the evolution of CNN topologies was described in chapter 4, whereas encodings were described in chapter 5. However, these encodings are not 100 % domain-agnostic. Still, the only aspect of optimization that is domain-dependent is the sampling setup, which is only applicable to those domains where samples are not segmented beforehand (e.g., signals).

(2) *Proposal of a fitness function, which may be domain-dependent, to be able to compare the performance of different CNN topologies.*

In chapter 4, we have presented a proxy fitness function that is able to accelerate the evolutionary process. The specific fitness functions are domain-dependent, since we have chosen to optimize the quality metric that is usually reported in the state of the art of each dataset.

(3) *Design and development of a system able to optimize the topology of a convolutional neural network, given the fulfillment of objectives (1) and (2).*

The system design was described in chapter 4, and its implementation has been completed.

(4) *Improvement of the system designed in objective (3) in order to reduce the computational cost in terms of time, by enabling parallel fitness computations.*

The previous system has been designed so that computation ca be effectively distributed among different GPUs. In particular, each experiment has been parallelized over 2 NVIDIA GTX 1080. This distribution is easy to achieve given a population-based metaheuristic, since the evaluations of different individuals are independent tasks that can be performed in parallel.

(5) *Selection of different domains to evaluate the performance of automatically evolved CNNs. These domains should cover a wide spectrum of applications: handwritten character recognition, signal classification, etc.*

We have selected two domains and a total of four databases. The first domain is handwritten digit recognition, for which we have used the MNIST and EMNIST Digits databases. The second domain is handwritten letters recognition, and we have used the EMNIST Letters dataset. Finally, we have also chosen OPPORTUNITY, a database that uses signals gathered from different opportunistically-found sensors in order to achieve human activity recognition. All these domains have been described in chapter 5.

(6) *Exhaustive review of the state of the art of the domains chosen in objective (5), explicitly discriminating best results attained using convolutional neural networks and other techniques.*

For each of the selected databases (MNIST, EMNIST and OPPORTUNITY), we have performed a careful review of the state of the art, discerning between results reported using CNNs and other techniques. These studies can be found in chapter 5.

(7) *Comparative evaluation of the best CNN obtained for each domain against the state of the art ranking elaborated in objective (6).*

The results of this comparative evaluation where shown in chapter 5, and in all cases the position in the ranking of the best CNN model evolved was discussed.

(8) *Construction of committees / ensembles using more than one of the best found CNN topologies for each domain.*

We have built committees of CNNs using a best-first approach, where new models are added to the committee based on how well they perform. This process was described in chapter 5.

(9) *Evaluation of the performance of the committees built in objective (9), comparing it against the state of the art and the best individual CNN.*

The performance of the committees as new models are added to them has been described in chapter 5 for all the different datasets. It is worth mentioning that the best results reported in this thesis have been always attained by committees of CNNs.

(10) *Analysis of the results and validation, if applicable, of the working hypotheses.*

The quantitative results have been reported in chapter 5. After completing this thesis, we have validated all the working hypotheses, as we described in the previous section.

6.3 Future Work

Neuroevolution of convolutional neural networks is an emerging area of research. However, we expect that current improvements in hardware architectures (e.g. tensor processing units, NVIDIA's Volta, etc) enable fast fitness computation, allowing researchers to focus on the development of new neuroevolution techniques and their application to new problems which remain unsolved. The automatic design of convolutional neural networks is a very promising area, and we believe it must be tackled as of today.

Because this field is quite unexplored, future advancements in this field can be done in many different directions. On one side, architectural improvements (both in hardware and distributed systems) are essential to allow the expansion of this field. For example, fitness could be computed in parallel in a cluster of GPUs specialized for deep learning. This not only requires hardware, but software enabling efficient and intelligent distribution of resources for the different processes.

Also, this thesis has shown that ensembles are quite interesting inasmuch as they have provided the best results attained in this work. However, regarding ensembles, in this thesis we have just scratched the surface, following a rather naive approach for their construction. A more specialized selection of ensembles could be performed, either by analytical processes selecting those models with higher variance, or by using genetic algorithms to determine which models to include in the ensemble. In both cases, performance could be even better than that obtained in this work.

While in this work we have focused on maximizing accuracy, some works may rather want to obtain good models which are also efficient, either in terms of time or energy expenditure. Decreasing time can help to obtain populations whose individuals are faster to train and validate, reducing the cost of fitness computations. Some works have been presented in this direction, such as trying to optimize fitness predictors from reduced training subsets [109]. As for energy expenditure, its reduction can be especially interesting when resulting models are going to be embedded in portable devices, such as smartphones or wearables. In this case, multiobjective evolutionary algorithms are an interesting approach to optimize the different objectives, taking advantage of the benefits of neuroevolution. This line has only been explored so far by Dong et al. [91] in a paper presented in last ICLR, which took place in May 2018. However, future work can settle the foundations for obtaining high performance models which are also energy efficient or which involve minimal topologies.

Finally, another improvement could involve supporting more complex architectures. At the very least, this would mean support for more convolutional or dense layers. However, non-sequential models could be comprised as well, where several convolutional layers are run in parallel and their output is later aggregated via element-wise summation. A general framework supporting any potential setup is a very interesting direction for future research, allowing for the evolution of more flexible deep learning models.

We hope this thesis to be a relevant contribution on the study of the application of neuroevolution to deep learning. However, we know that there is still a lot of work to be done within this area. Only as this field emerges and keeps advancing, new future directions will unveil.

This page has been intentionally left blank.

Appendix A

GPU Architecture

A.1 Introduction

In this work we have used four NVIDIA GeForce GTX 1080 GPUs. While this device is specifically designed for gaming, it offers a good cost-quality tradeoff when used for running deep learning applications, specially when compared to specific deep learning products such as the NVIDIA Tesla line, which is significantly more expensive. Empiric support for choosing the use of this device is provided in the following appendix. Figure A.1 shows the GTX 1080 Founders Edition; however, we have installed ASUS ROG Strix boards, which share the same architecture than the original NVIDIA device, with improved cooling mechanisms and a different look.

According to NVIDIA [271], the continuous advancement of their GPUs is leading to tremendous improvements in GPU-accelerated computing, and enabling "*groundbreaking advances in AI, deep learning, autonomous driving systems, and numerous other compute-intensive applications*".

NVIDIA GeForce GTX 1080 is based on the Pascal GPU architecture, which was first introduced in the datacenter-class GP100 GPU. After the GP100 GPU, NVIDIA presented the GPU-GP104, which was first shipped in the GTX 1080 devices. According to NVIDIA, as of 2016 GP104 was the fastest gaming GPU in the world, and Pascal was the most efficient GPU architecture.

Figure A.1: *NVIDIA GeForce GTX 1080 Founders Edition. Source: NVIDIA [271].*

A.2 Pascal Innovations

The Pascal architecture used in the GTX 1080 devices comprises 7.2 billion transistors and 2,560 single-precision CUDA cores. The NVIDIA GTX 1080 whitepaper [271] remarks the following key highlight of this architecture: 16 nm FinFET manufacturing process, that allows the inclusion of more transistors and improved power efficiency, and GDDRX5 memory, running at a data of 10 Gbps and a 256-bit memory interface, supporting memory compression at an architectural level.

A.3 In-Depth Architecture

Pascal GPUs comprise different configurations of Graphics Processing Clusters (GPCs), Streaming Multiprocessors (SMs), and memory controllers. The architecture of the GP104 GPU, featured in GTX 1080 boards, is shown in figure A.2. It contains four GPCs, each one shipping one raster engine and five TPCs. A TPC is a unit containing a PolyMorph Engine (which provides rendering features) and one SM. As a result, the device ships a total of 20 SMs.

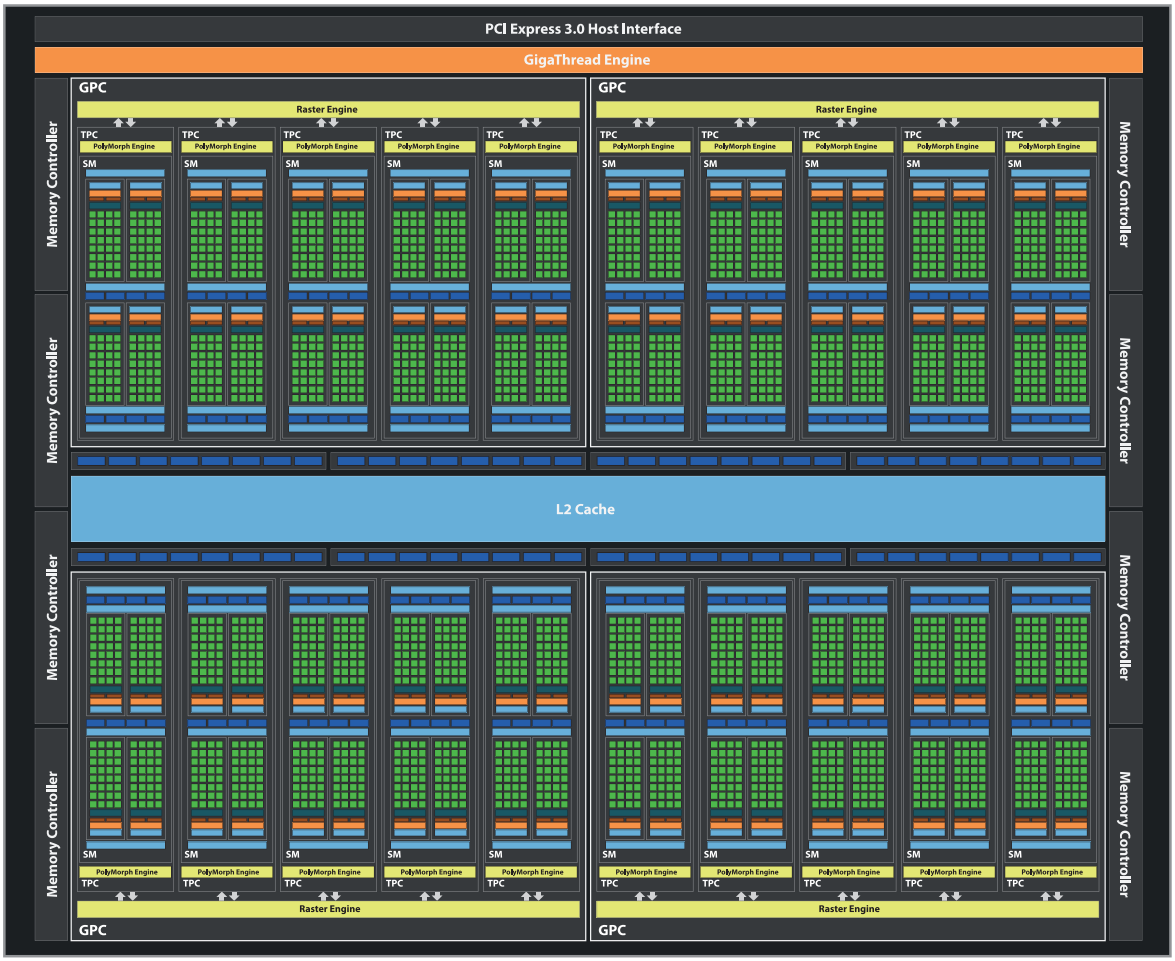


Figure A.2: *Block Diagram of the GP104 GPU. Source: NVIDIA [271].*

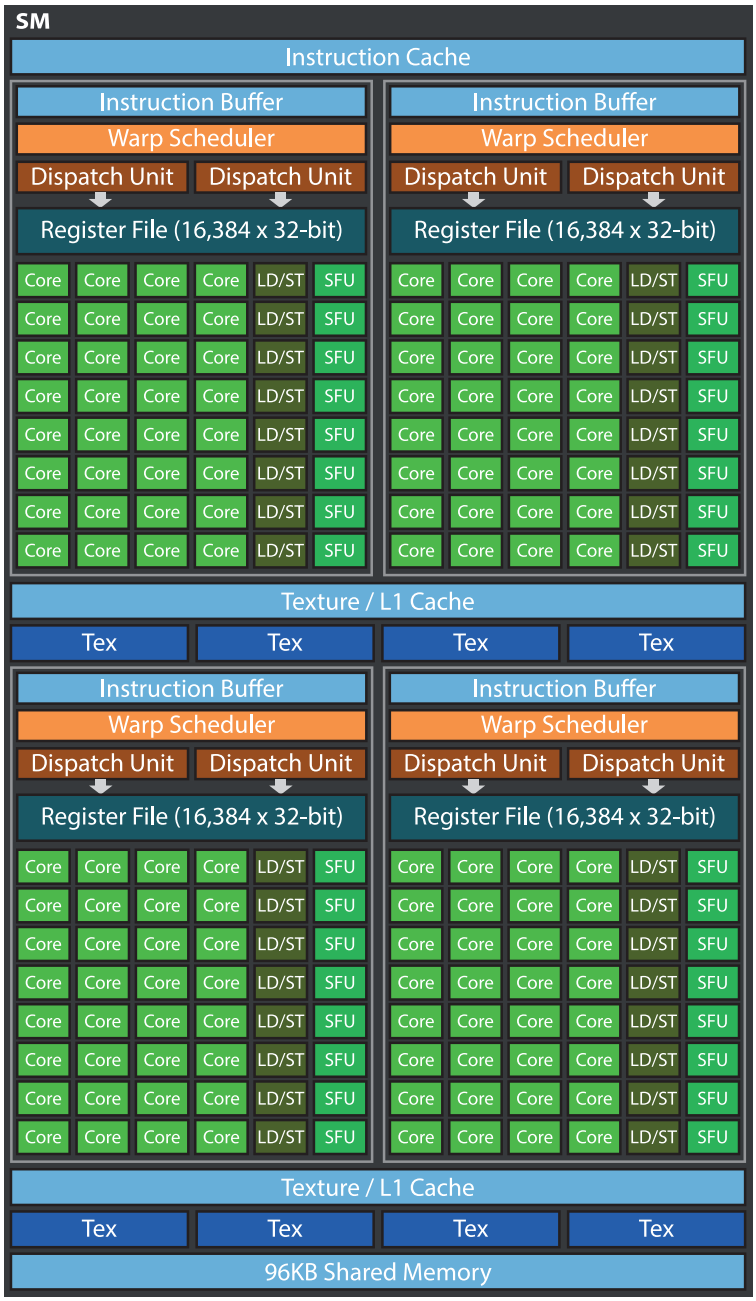


Figure A.3: *GP104 SM Diagram. Source: NVIDIA [271].*

The architecture of each streaming multiprocessor is shown in figure A.3. Each SM contains 128 CUDA cores, 256 KB of register file capacity, a 96 KB shared memory unit, 48 KB of total L1 cache storage, and eight texture units. As a result, it is a highly parallel multiprocessor that, in the words of the NVIDIA whitepaper [271], "*schedules warps (groups of 32 threads) to CUDA cores and other execution units within the SM*".

As a result, SMs are key components of the GPU architecture since all operations flow through a SM at some point. Because the GeForce GTX 1080 includes 20 SMs, it contains a total of 2,560 CUDA cores and 160 texture units.

Feature	**Value**
Streaming Multiprocessors	20
CUDA Cores	2,560
Base Clock Frequency	1,607 MHz
Boost Clock Frequency	1,733 MHz
GFLOPs	8,873
Texture Units	160
Memory capacity	8,192 MB
Memory Clock (Data Rate)	10,000 MHz
Memory Bandwidth	320 GB/sec
L2 Cache Size	2,048 KB
TDP	180 Watts
Transistors	7.2 billion
Die Size	314 mm^2
Manufacturing Process	16 nm

Table A.1: *Technical specifications of the NVIDIA GeForce GTX 1080 GPU.*

Table A.1 summarizes the main technical specifications of the GPU GeForce GTX 1080 used in this thesis. More specifically, these specifications refer to the ASUS ROG Strix version that we have installed in our workstations.

Appendix B

Performance Considerations

Before starting the implementation of the neuroevolutionary software developed in this thesis, we carried out a preliminary study regarding the performance of different devices and deep learning frameworks. Because we did not own any server with modern GPUs suitable for deep learning, at the time of starting the experiments we decided to purchase specific hardware.

In this appendix, we will briefly summarize the main conclusions of this preliminary performance analysis, focusing both in hardware and software aspects.

B.1 Hardware Analysis

At the time of purchasing new hardware with GPUs suitable for deep learning, we reviewed the literature to find which devices presented the best performance-cost ratio. A very complete work was published by Shi et al. [327], comparing different frameworks and hardware architectures, including NVIDIA Tesla K80 with Kepler architecture, GeForce GTX 980 with Maxwell architecture and GeForce GTX 1080 with Pascal architecture. It can be seen that GTX 1080 provides the best performance (figures 9–14 in the paper, ranges in vertical axes differ between images). At the time of purchasing, the cost of GTX 1080 was €699 before VAT. The price of GTX 980 was only slightly inferior; where as Tesla K80 is a much more expensive device, with a price over €4,000 before VAT.

In the end, we decided to purchase servers with two NVIDIA GeForce GTX 1080 each, since it seemed to provide the best performance-cost trade-off. The servers featured an Intel i7-6700 4-core CPU running at 3.4 GHz. The first analysis we performed compared the speed of the CPU against GPUs when running a deep learning model. To achieve this comparison, we installed the GPU-compatible version of TensorFlow 0.12 along with CUDA 8.0 and cuDNN 5.1 (the latest versions available at the time of running these experiments).

The deep learning model trained is described in the "Deep MNIST for Experts" tutorial in the TensorFlow documentation [357], which involves a CNN with two convolutional layers with 32 and 64 filters of size 5 respectively and max-pooling of size 2, one fully connected layer with 1024 neurons and dropout of 50 % and a softmax layer. The training process will run for 20,000 epochs.

B.1.1 Running on Different Devices

In the performance comparison, we will include three different working modes:

Mode	Flags	Time (s)
CPU	`tf.device('/cpu:0')`	1573.05
Single-GPU	`tf.device('/gpu:0')`	90.04
	`tf.device('/gpu:1')`	96.69
Multi-GPU	`tf.device(['/gpu:0', '/gpu:1'])`	98.39

Table B.1: *Performance of different hardware configurations.*

- **CPU**: in this mode, TensorFlow will only use the CPU and ignore existing GPU devices.
- **Single-GPU**: in this configuration, we will use only one of the two GPUs. In order to compare the performance, we will use both GPUs separately, i.e., running two experiments.
- **Multi-GPU**: in this configuration, we will ask TensorFlow to run on the two available GPUs.

The results of the comparison are shown in table B.1. It seems clear that GPUs are much faster than CPUs to run this job, taking 1.5 minutes instead of 25 minutes to train a model. The fact that the second GPU is slightly slower than the first can be due to the fact that its position in the computer chassis makes it to remain at a higher temperature. Finally, using two GPUs in parallel does not increase the speed when training a single CNN model.

Finally, the conclusions of two follow-up studies have been shared publicly with the community. In the first of these studies, we perform a comparison between NVIDIA's GeForce GTX 1080 and Tesla P100 for deep learning [14], whereas in the second we have analyzed the performance of Xeon Phi, an Intel-manufactured CPU with specific optimizations for deep learning [15].

B.1.2 Multithreading

We have seen that trying to parallelize the training of one model between two GPUs is not providing any advantage in speed. However, we still do not know whether the process can be accelerated by training several models in the same GPU board. Therefore, to try to increase the performance gain, we will now use multithreading, performing two different experiments:

- **2-per-GPU**: in this experiment, we will be training four models, two per GPU.
- **4-per-GPU**: in this experiment, we will be training eight models, four per GPU.

The results of the experiments are summarized in table B.2. In these results, we can see how multithreading is useful when training two models concurrently, one on each GPU, since a single model took about 90 seconds and two models take 156, thus resulting in saving about 24 seconds. However, when trying to parallelize more models on the same GPU, there is no advantage at all, since training 4 models takes more than twice the time required to train 2 models.

Mode	Time (s)
2-per-GPU	156.67
4-per-GPU	313.78

Table B.2: *Performance of different multithreading setups.*

B.2 Software Analysis

By the time of starting the experiments, we decided to test the performance of TensorFlow against Theano, which are two of the most-well known deep learning frameworks (although Theano has been discontinued as of November 2017). In order to ease the task of designing the architecture of convolutional neural networks, we decided to use a library for fast prototyping of neural networks, the most relevant at the time being Lasagne and Keras. Lasagne only worked with Theano; whereas Keras was able to run over different backends, including Theano and TensorFlow.

At the time of performing the comparative, we used the following software stack: Theano 0.9.0, Lasagne 0.2.dev1, TensorFlow 0.12 and Keras 1.0. Keras was originally developed for Theano, although several backends were added later, including support for TensorFlow. By the time of starting the experimentation stage of this Ph.D. dissertation, Keras was already available for TensorFlow, but some preliminary testing showed that it ran with an important overhead, taking 2–3x as much as Theano with Lasagne. This can be explained in part because Lasagne had a longer record as an open source project, even though Keras has earned momentum in recent years, probably due to Theano becoming abandoned in November 2017. These results, along with the ease of porting Ordoñez and Roggen's DeepConvLSTM code [277] for evaluating this thesis over the OPPORTUNITY dataset, turned Theano + Lasagne into the stack of choice for this Ph.D.

It is remarkable that many newer versions of TensorFlow, Theano and Keras have been released since these experiments were conducted, and as of today, this section cannot be used for a fair comparison between the current stable releases of these tools. Also, this section was not intended to provide a scientifically sound benchmark of deep learning software, but rather to justify the decision behind choosing particular tools.

This page has been intentionally left blank.

Appendix C

DeepNE Scientific Proposal

This appendix replicates the scientific proposal for DeepNE, a project to be presented to the Spanish Ministry of Economy, Industry and Competitiveness.

The purpose of DeepNE is to extend and develop one of the future works described in chapter 6: the development of a scalable framework that accelerates fitness computations in the evolutionary process. This framework will use the resources provided by a cluster specifically provisioned for this purpose, comprising several GPUs specialized for performing deep learning tasks.

Executive Summary

Very recently, deep learning techniques such as deep and convolutional neural networks (DNNs and CNNs respectively) have become a standard to solve a large variety of problems. However, the performance obtained by using these techniques is highly dependent on their design, which requires very specific expert knowledge.

So far, all well-known deep learning models have been trained using topologies handcrafted by experts. This process is time-consuming and requires a great effort and experience. Even though, the resulting topologies have significant room for improvement.

For the last three decades, neuroevolution (NE) has proven the convenience of applying evolutionary computation techniques in order to determine optimal topologies for neural networks. However, the application of NE to DNNs and CNNs remains unexplored: there are very few works in this area, most of which have appeared on 2017 or 2018.

In DeepNE, we propose the development of a neuroevolutionary system for the automatic design and optimization of DNN and CNN topologies. For this system to be efficient, we have established a three-layers architecture:

1. A layer of hardware resources, namely DeepForce, comprising servers with graphical processing units (GPUs) designed for deep learning, thus providing a very high performance.
2. A system for the coordination and orchestration of the hardware resources, namely DeepRector, whose responsilibity involves allocating processes and tasks to the hardware resources available in DeepForce in order to achieve their successful execution.
3. A software for the optimization of DNN and CNN topologies using neuroevolutionary techniques, namely DeepEvol, which will run in top of DeepRector in order to parallelize fitness computations and the evolution of populations.

This project would allow EVANNAI (Evolutionary Algorithms, Neural Networks and Artificial Intelligence) research group and Universidad Carlos III de Madrid to be at the forefront of innovation in artificial intelligence, by using the developed system to open new possibilities in the research and development of deep learning models, a new emerging field at a global scale.

Moreover, the modular design of the project enables three different possibilities for its commercialization: 1) the renting of hardware resources to third-parties, 2) providing the DeepRector platform to clients in order to run their processes and 3) offering the DeepEvol software as a service for research centers and companies to be able to obtain optimal deep learning topologies and models that suit their needs.

Keywords: Deep Learning, Deep Neural Networks, Convolutional Neural Networks, GPU, Evolutionary Computation, NeuroEvolution

C.1 Background

Artificial intelligence (AI) is a field of study within computer science, which was born in the mid-1950s having as its main aim the development of *intelligent* machines and systems.

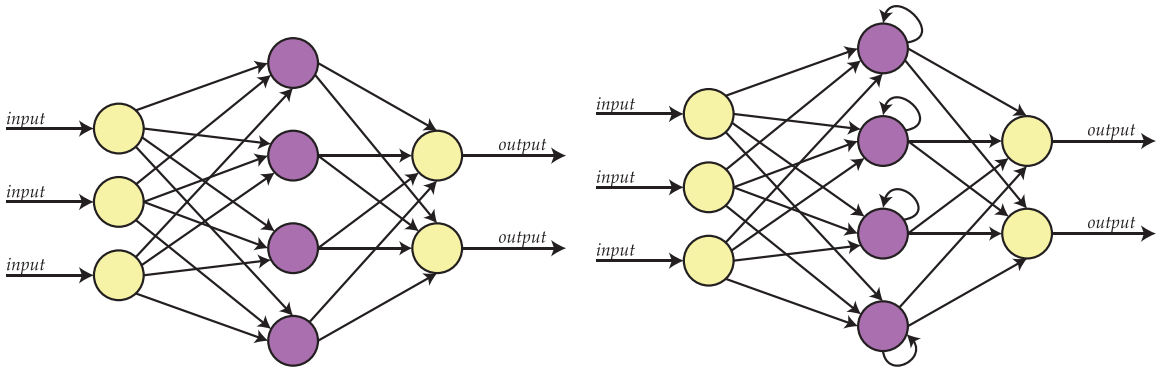
A specific area of AI is *machine learning* (ML), whose aim is the development of systems which are able to learn a task without being explicitly programmed to carry it out. While ML comprises many different techniques, the truth is that most of them can be classified in three broad categories:

- Supervised learning: starting with labelled data (previously classified), it tries to learn a model which is able to automatically infer the class for unlabelled data.
- Unsupervised learning: starting with unlabelled data, it looks for clusters of items showing certain common patterns.
- Reinforcement learning: given an environment and a system able to perform certain actions over it, searches the optimal policy for achieving certain objectives over that environment.

One of the AI techniques allowing the resolution of tasks belonging to any of the three previous categories are artificial neural networks (ANNs). Despite ANNs appearing in the 1950s as an approach to AI that tried to resemble and mimic the behavior of the human brain, the systems' limitations and a poor development of the mathematical background behind their functioning prevented ANNs from being widely used until the 1980s.

Starting that decade, and specially since 1986 when the backpropagation algorithm was discovered, ANNs gained a significant relevance in the development of AI applications, such as autonomous driving [285], intelligent non-playable characters [358] or handwriting recognition [204]. Moreover, new advancements and developments in the architectures of ANN, such as recurrent neural networks, enabled modelling and learning temporal behaviors.

An important issue regarding ANN is that its performance, both in terms of effectiveness and efficiency, is highly dependent on its topology. So far, there are no established analytic rules able to determine the optimal ANN topology for solving a specific problem. In consequence, the most common approach of determining valid topologies is through trial-and-error: testing different architectures and evaluating their performance, then manually adjusting each hyperparameter until a satisfactory solution for the problem is found.



(a) *Feed-forward* **(b)** *Recurrent*

Figure C.1: *Different connectivity patterns in artificial neural networks.*

In general terms, when talking about the architecture (or topology) of an ANN we refer to the number of hidden neurons and their disposition between several layers, as well as the connectivity pattern between such neurons. Figure C.1 shows two different types of neural networks: a feed-forward network in the left size (more specifically, one known as *multilayer perceptron*) and a recurrent neural network in the right side. In the first case, connections always exist from neurons in one layer to neurons in the following layer; whereas in the second, connections can occur within the same layer or to previous layers. It is worth noting that the number of input and output neurons is determined by the problem, and do not constitute an optimizable hyperparameter.

Already since 1989, the scientific community has tried to develop systems which are able to automatically find optimal ANN architectures to solve specific problems. As soon as this need arose, several authors considered that given the nature of this problem and its characteristics, a suitable approach for the development of this system involves the use of evolutionary computation [245,400].

These early works gave birth to a new field of study which would be object of a large number of scientific contributions and developments over the next three decades: neuroevolution (NE), also known in some contexts as *evolutionary artificial neural networks* (EANNs).

The Encyclopedia of Machine Learning and Data Mining [313] defines NE as a method for evolving the weights or the topology of a neural network in order to learn a given task, using evolutionary computation techniques that search for the hyperparameters maximizing a performance metric of the network over such task.

In the 1990s and 2000s, several remarkable developments were presented in the field of NE, such as GNARL in 1994 [6], EPNet in 1999 [401], NEAT in 2002 [345] or EANT in 2005 [175]. Furthermore, there are thousands of scientific contributions during this period evaluating the convenience of applying NE to specific problems, being some common applications videogames [303], robotics [40,269] or even biomedicine [181,215].

Despite NE being used almost for three decades, the truth is that most of the time it was focused on networks with only one single hidden layer. For this reason, many works in NE aimed at optimizing the topology of ANN addressed very small search spaces.

Nevertheless, due to the advancement of research and the improvement of current computing capabilities, in recent years (especially since 2009) new concepts have been developed within the world of ANNs. On one side, it is common to find deep neural networks, which are those with more than one hidden layer. Besides, it is also common the use of convolutional neural networks, which are those introducing layers that run a convolution operator over the input.

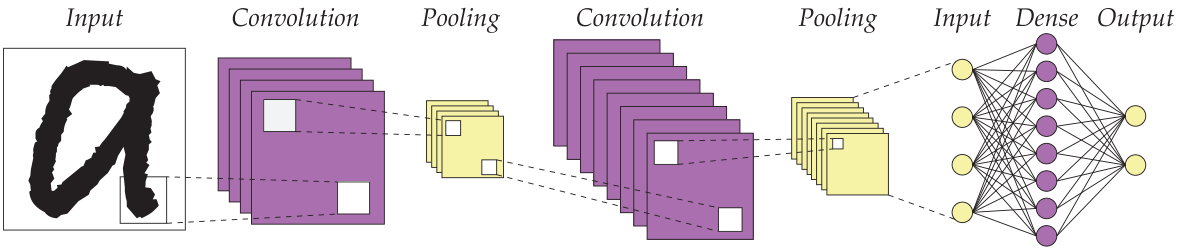


Figure C.2: *CNN comprising two convolutional layers with pooling and one dense layer.*

An example of a CNN is shown in figure C.2. In this figure, it can be seen how the network comprises several convolutional layers. Each of these layers contains several filters (also known as *kernels*) of a given size. These filters will be applied over the input data (in the case of the first convolutional layer) or over the output of the previous layer (in subsequent convolutional layers). Moreover, between convolutional layers we can find a so-called *pooling* layer which can reduce the data dimensionality. Finally, after all the convolutional layers, the figure shows an architecture with feed-forward layers (also called *dense* layers) which will carry out the classification task given the features extracted by the convolutional layers.

As it can be inferred from figure C.2, DNNs and CNNs comprise more complex architectures. Some of the optimizable hyperparameters are: the number of convolutional layers, the number of filters in each layer, the size of these filters, the existence or not of pooling layers, the pooling size, the number of dense layers, the number of hidden neurons in each of these, the neurons' activation functions, the connectivity pattern (feed-forward or recurrent), etc. Moreover, some hyperparameters of the learning process can also be optimized, such as the learning rule or the learning rate.

In this case, the search space is extremely large and the automatic design of DNNs and CNNs is a problem with a high research potential, given the extensive use of this kind of techniques. However, so far all of the main architectures and models for solving diverse problems have been obtained via a manual design procedure.

Nowadays, there is roughly a dozen academic works in this new research line, remaining as a mostly unexplored field. Most of the bibliography have been published since March 2017 in preprint repositories like arXiv. The most remarkable works exploring the evolution of convolutional neural networks are four: MetaQNN [10], GeNet [393], CoDeepNEAT [241], and EXACT [88].

From these four works, only the last three would be framed in the area of NE, since MetaQNN does not rely on evolutionary computation techniques. As for the other three, an in-depth analysis of their features enables the finding of some room for improvement:

- GeNet and EXACT do not evolve the part of the network involving dense or recurrent layers, nor the neurons' activation functions or the learning hyperparameters. Moreover, they do not support the development of committees (ensembles) of CNNs, a solution that enables mixing several networks in order to obtain a more robust result with a better performance.
- CoDeepNEAT provides a very complete proposal, except for the lack of support to the development of CNN committees.

The current proposal relies on some early exploratory works performed in the scope of a Ph.D. dissertation, entitled "Evolutionary Design of Deep Neural Networks", partially funded by the FPU scholarships programme of the Spanish Ministry of Education, Culture and Sports under grant no. FPU13/03917. The thesis is being completed within EVANNAI, a group with a solid teaching and

research record in the fields of evolutionary computation and artificial neural networks, and to which most of the project team is affiliated.

The preliminary results obtained so far with highly constrained resources seem to point out that these techniques can turn out to be very convenient to solve this task, and have served as a motivation to request this project.

The current proposal has as its main aim to advance the research that tackles certain obstacles that are currently limiting the possibilities for developing neuroevolutionary systems for the automatic design and optimization of DNN and CNN topologies. To do so, we are proposing a line of work organized in several axes that enable the deployment of a distributed and scalable system to ease the training and exploitation of DNN and CNN models.

C.2 Working Hypotheses and General Goals

The current proposal is made after several working hypotheses related to the obstacles that are hardening the development of neuroevolution techniques for DNNs and CNNs, some of which have been addressed before. Specifically, these hypotheses are the following:

- Manual design of DNN and CNN architectures is a task requiring a significant effort and leading to topologies which are far from the optimal.
- The design of DNN and CNN architectures can be modelled as an optimization problem, which can be solved in an effective and efficient manner using evolutionary computation.
- The topologies obtained through evolutionary optimization will have a higher performance when compared to those manually designed ones. In the worst case, they will show a similar performance, yet requiring less design effort.
- The creation of committees of DNNs and CNNs will enable obtaining a higher performance that any of the individual models they comprise. For this reason, if during the evolutionary process several successful solutions are found, these can be used to build a committee.
- A system can be developed so that, using evolutionary computation, it is able to determine the optimal topology for a problem at hand receiving as input only a training set and a quality metric over a validation set.

On the other hand, a system of such characteristics implies very strict computing infrastructure requirements, since the evolutionary process must train and evaluate a large number of DNN and CNN models, resulting in a very time-consuming task.

It is well known that computations on this kind of models are approximately one order of magnitude faster when GPUs are used instead of general purpose processors (CPU). This is because current generation GPUs comprise thousands of cores able to run computations in parallel. For instance, NVIDIA recently released the GeForce TITAN Xp, featuring 3,840 CUDA cores working at a frequency of 1.5 GHz. Additionally, in recent years specific primitives for deep learning have been released, such as cuDNN, enabling the execution of certain common operations when training and exploiting neural networks over the GPU in a more efficient manner.

Moreover, we have explored the possibility of using cloud computing with GPU instances, an option already available in the most important cloud vendors, such as Amazon Web Services, Google Cloud or Microsoft Azure. However, we have determined that the cost of using this service is higher in the mid- and long-terms, offering less performance than dedicated hardware.

Based on this fact, research enabling the development of scalable architectures supporting the neuroevolutionary system would have a multiplier effect. Its results would affect neuroevolutionary techniques *per-se*, as well as simplify the creation and exploitation of DNN and CNN models by potential users non-familiar with the complexities of deep learning.

The requesting group considers that the only road to advance the research in deep learning requires to contribute to the development of scalable systems for specific GPU computing. To do so, we would work in two different lines: 1) research on how to tackle this problem, and 2) development of a fully-functional solution. This idea arises from two additional hypotheses:

- If a pool if GPU resources is available, then a hardware-software platform can be developed for managing such resources, therefore enabling to run processes over them.
- An architecture of this kind is a requirement for developing scalable systems enabling the training and exploitation of DNNs and CNNs in a parallel fashion.

Once the working hypotheses have been settled, we can establish the proposal's general goals:

GG.1 The development of a neuroevolutionary system for the design of deep and convolutional neural network topologies, able to obtain a result outperforming other machine learning techniques and topologies handcrafted by experts and currently reporting state-of-the-art results.

GG.2 The design and development of a scalable hardware-software platform for deep learning using GPUs, over which the hypotheses will be validated and the GG.1 will be tackled.

C.3 Specific Goals

In order to achieve the previous general goals, we have specified specific goals, being each of them verifiable in an independent manner. These goals are the following:

SG.1 Provision of a distributed system for GPU computing specifically for deep learning. This system will comprise several nodes, each of them with one or more GPU. Without this hardware setup, the rest of the objectives are not achievable.

SG.2 Development of an orchestration system for the coordination of these GPU resources. This system must remain aware at all times of the available resources and their status (if they are idle, running processes, their temperature, technical specifications, etc).

SG.3 Development of a processing framework, which accepts clients requests with a self-contained processing bundle including data, code and running instructions; and uses the orchestration system for running this processing bundle, returning the result to the client.

SG.4 Proposal of one or more domain-agnostic encodings that enable the representation of DNN and CNN topologies, so that these can be evolved using evolutionary computation techniques.

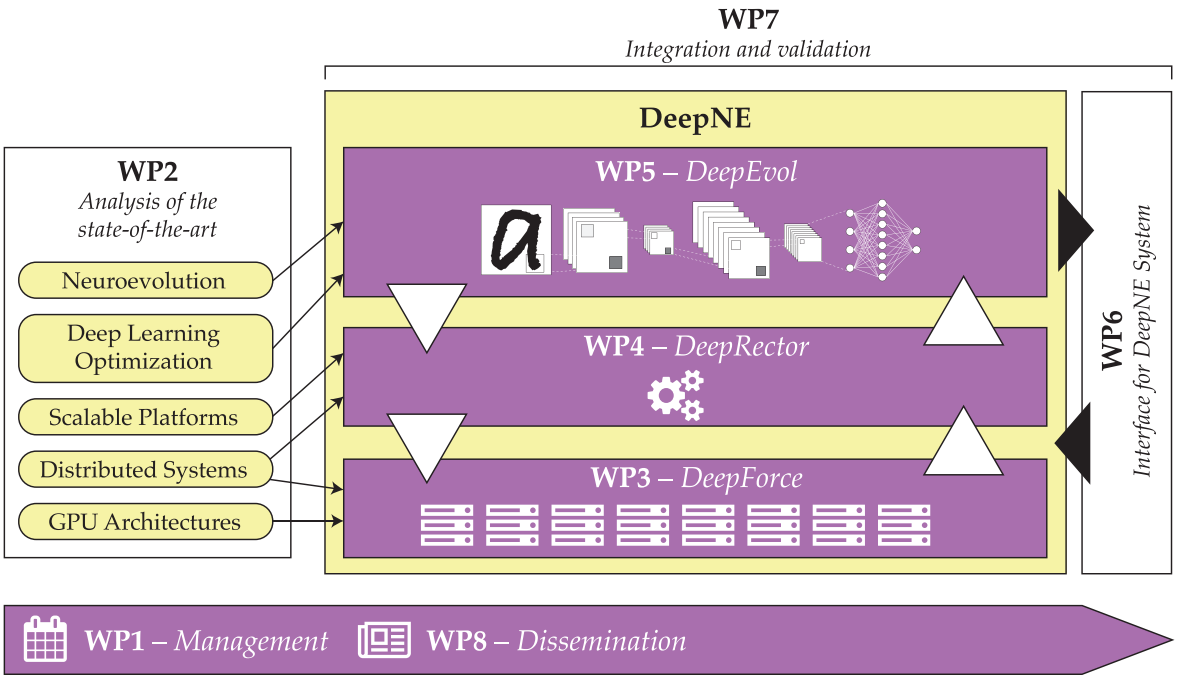


Figure C.3: *Schematic architecture of the DeepNE proposal, along with its working packages.*

SG.5 Design and development of an evolutionary system able to optimize DNN and CNN topologies using the encodings proposed in the specific goal SG.4. This system must be implemented over the processing framework developed in the specific goal SG.3, thus enabling scalable training and evaluation of the neural networks comprising the populations of the evolutionary algorithm.

SG.6 Development of an interface that simplifies the access to DeepNE and its use to users, even those without expertise in deep learning.

SG.7 Selection of different domains to carry out a comparative and objective evaluation of the neuroevolutionary system. These domains will ideally belong to very diverse areas (handwriting texts, images, waveforms, etc.) so that the system can be validated in different application areas. The results obtained in this evaluation will be compared against the state of the art, in order to ellaborate conclusions regarding the system's performance.

The general system architecture is shown in figure C.3, placing the different working packages, which will be described in the following section, into context. Next, each of the elements involved in this architecture will be described in further detail.

DeepForce

DeepForce is a pool of computational resources mostly composed of GPUs. Each of these graphic cards will contain several thousands of processing cores, thus enabling the training and exploitation of DNNs and CNNs approximately one order of magnitude faster than using CPU.

The main advantage of DeepForce is that computational resources can be of several types, i.e., they can involve different GPUs models, with different setups or even belonging to different

generations. The platform layer, DeepRector, will be in charge of coordinating the power and capabilities offered by the DeepForce resources.

DeepRector

DeepRector is a platform whose responsibility is to coordinate (or orchestrate) the hardware resources available in DeepForce. Its main task is to receive works via clients requests and allocate resources to those works in order to successfully achieve their completion.

To do so, DeepRector must be aware of the resources availability. To enable this, an approach common in many distributed systems will be followed: nodes in DeepForce will run a little background service (*daemon*) that will periodically notify DeepRector of the services available in each of them. In particular, this daemon will share with the orchestrator the number of available GPU devices, and for each of these GPUs, it will inform of its status (idle or busy), its model, its core temperature, its workload and its free memory.

When a client makes a request to DeepRector, it must send a self-contained bundle including data, code to be run and a specification containing running instructions. From that moment, DeepRector should provision the hardware resources needed to successfully complete the request. Upon completion, DeepRector will provide the client with the output generated by the program.

The resources allocation stage can be intelligent enough as to avoid unnecessary network traffic, preventing data and code to be sent to a node if the node already has them. Moreover, DeepRector must have means for re-running the execution of the program if a node failed or stopped working.

DeepEvol

DeepEvol is an evolutionary computation system for the optimization of DNN and CNN topologies. Its purpose consists on finding an optimal DNN or CNN architecture given a training dataset, a validation dataset, and a quality metric to evaluate a model over the validation set. This optimal architecture must maximize the quality value.

Since evolutionary computation techniques are based on populations, they can be easily parallelized. This means that DeepEvol can scale out its efficiency in a linear fashion, thus doubling the execution speed if the available computing resources were doubled as well.

For this reason, DeepEvol will request the fitness computations, which require the training and validation of a given DNN or CNN topology to check its quality, to DeepRector. This will enable distributing the load of the evolutionary process among all available hardware resources.

C.4 Methodology

The project is organized in eight working packages (WPs), as shown in table C.1.

All the proposed tasks are feasible and we do not foresee any difficulty that could jeopardize the achievement of the project goals, besides those common to every research task, especially in those new research lines that will be tackled in the proposal.

For the application and validation of the neuroevolution techniques for deep learning, we will mostly be using the Python programming language. In order to ease the implementation and

WP #	Title	Start Month	End Month
WP.1	Project management	M01	M36
WP.2	Research of the state-of-the-art	M01	M06
WP.3	Provisioning and deployment of a GPU computing hardware environment specific for deep learning	M02	M08
WP.4	Research, analysis, design and development of a system for coordinating GPU resources	M06	M12
WP.5	Research and development of a neuroevolutionary system	M06	M28
WP.6	Development of an interface for accessing DeepNE	M20	M26
WP.7	Integration and validation of the developed system	M24	M36
WP.8	Project dissemination	M01	M36

Table C.1: *Working packages and duration.*

optimize the execution of the developed software, we will use the latest stable versions of specific libraries for numerical calculus (NumPy), scientific processing (SciPy), machine learning (Scikit-Learn), and data analysis (Pandas). Moreover, we will evaluate the best alternatives regarding deep learning libraries, comparing TensorFlow, Theano and PyTorch. Moreover, we will also study the use of libraries for abstracting the design of DNN and CNN models, such as Keras or Lasagne.

Each of these working packages implies the completion of a set of tasks and subtasks, which are thoroughly described in this section. The schedule for each task is detailed in section C.6.

WP1: Project management

This working package will cover the whole project duration. Its main objectives are: 1) ensuring fluid communication between team members, 2) checking out the achievement of all tasks in due time and 3) working on the partial and final project reports.

The two principal investigators will be in charge of this working package. They will be responsible for coordinating the tasks and evaluating the progress of the process. The working methodology proposed for this project has been successfully used in previous projects within the EVANNAI research group. For achieving these objectives, we propose the following tasks:

Task T1.1 *Establishing the organization and methodology for collaborative work.* This task will serve for settling the basis of collaborative workflow for this project. We will create a mailing list for all the members in the project team and a repository for code, data and documents relative to the project. The proposal observes the use of the following tools:

- Google Docs as the tool for collaborative work and document edition.
- Dropbox as the general tool for file sharing, version control and backups.
- ShareLatex as the tool for writing academic papers.
- Skype and/or Gotomeeting for video-conferences when on-site meetings are not possible.

Task T1.2 *Project control and follow-up.* We will perform a strict control of the status of all tasks based on the objectives and milestones described in this proposal, using Atlassian JIRA as the integrated software for task management. By using this system we will plan and supervise the

tasks in a methodological way. Moreover, we can generate reports and easily detect issues during the project execution. We will adhere to the agile methodology SCRUM for software development, in which system requirements are described in a summarized manner and they are tackled in an iterative and incremental way.

Task T1.3 *Project coordination.* We will have periodic follow-up and coordination meetings no longer than 30 minutes. We will also hold a closing meeting upon completion of each work package.

Task T1.4 *Project reports.* The principal investigators will be responsible for delivering the final technical and economical reports to the Ministry of Economy, Industry and Competitiveness.

Deliverable D1.1 *Project partial report (first year)*

Deliverable D1.2 *Project partial report (second year)*

Deliverable D1.3 *Project partial report (third year) and final report*

Risks A possible risk involves the need to ensure the compatibility between the adequate project management and coordination with other teaching tasks or other ongoing research projects.

Contingency plan We have decided to propose two principal investigators in order to be able to balance the management workload, thus tackling this potential risk.

WP2: Research of the state-of-the-art

This working package is essential in order to be able to achieve specific goals from SG.1 to SG.6. It involves fundamental research which will be carried out by all the project members. This working package is divided in the following tasks:

Task T2.1 *Exhaustive analysis on the techniques for automatic design of deep neural networks.* In this task we will perform a research work that allows us to summarize all the techniques for automatic design of deep neural networks existing up to date. We will include all these techniques, describing in further detail the most remarkable ones and the results obtained by these.

Task T2.2 *Exhaustive analysis on the software technologies for deep learning.* This task proposes a research work that enables finding those existing software technologies for running deep-learning based systems. We will include all of those distributed under open-source licenses, describing their strengths and weaknesses in order to choose the most appropriate technologies.

Task T2.3 *Comparative analysis on hardware for deep learning.* The last task of this working packages proposes to perform a comparative analysis of all existing hardware platforms for running deep learning systems. This task will support the decision taking stage in task T3.1.

Deliverable D2.1 *Project-related state of the art.* As the result of this working package we will elaborate a document that gathers the research conclusions of the three tasks. This document will serve as the basis for dissemination of this project in international conferences and journals.

Risks In deep learning, the algorithms and versions of the studied systems can change rapidly, so it is possible that a survey of the state-of-the-art in the first year be obsolete in the third year.

Contingency plan We propose updating the survey of the state-of-the-art in an annual basis, using the latest stable versions when possible and using a continuous integration system that enables us to adapt during the development of the project when new versions or technologies are available.

WP3: Provisioning and deployment of a GPU computing hardware environment specific for deep learning

This working package directly addresses the specific goal SG.1: acquiring, installing and configuring the hardware environment (DeepForce) for the execution of the project. This equipment is very specialized and expensive, and its installation must comply with very specific cooling and electric supply requirements.

In this project, we will install the hardware in Universidad Carlos III de Madrid's computing center, which are facilities administered by qualified personnel, whose temperature and electric supply guarantees the proper functioning of the GPU-based computing system involved in DeepForce.

Task T3.1 *Definition of technical specifications.* Based on the cost and technical advances on specific hardware for deep learning that take place between this proposal and the time when this task will be carried out, we will decide on the specific technical capabilities that must be purchased to constitute the best investment for the budget allocated for hardware acquisition.

Task T3.2 *Public procedure for contracting of suppliers.* Given the cost of the equipment required for this project, we will complete a procedure for contracting of suppliers via UC3M's Financial Office, as specified in the Law of Contracts for the Public Sector.

Task T3.3 *Physical deployment.* Once the public procedure is resolved and the equiment is received, we will proceed to its installation in UC3M's computing center. To do so, we need to hire the hosting and electric supply required by the equipment, proceed with the physical installation and configure the networking setup of the hosts.

Task T3.4 *Software configuration.* This task will install and setup the software components required for this project. We will install Ubuntu Linux 18.04 LTS as the operating system in all the nodes and develop a script for the automatic deployment of all the required software for this project.

Figure C.4: *Approximate illustration of the equipment that will be used for dissemination purposes.*

Deliverable D3.1 *Dissemination of DeepNE system.* At this point, and in order to give visibility to the project, we will take pictures of the hardware items along with DeepNE and MINECO's branding (see figure C.4), and will work on a press release in the project website, which will be shared with UC3M's Office for Scientific Information in order to publicly announce the acquired equipment and its possibilities.

Milestone M3.1 *Hardware available and working.* This is an essential milestone supporting the rest of the project, since WP4 and WP5 rely on this hardware.

Risks This working package is critical due to its dependencies with third-parties. In particular, it depends on UC3M's Financial Office to carry out the contracting process and on the contracted supplier to deliver the equipment in due time. Moreover, besides a possible delay in the delivery, there could also be technical issues with some of the acquired hardware components. An additional source for potential delays is the appearance of NVIDIA's Volta architecture, which is faster than Pascal and optimized for tensor computing. It should be studied whether delaying the purchase 2 or 3 months is a suitable option for acquiring Volta hardware.

Contingency plan If this task is delayed a few months, it would not have a great impact as we could start working in WP4 and WP5 with 2 nodes with 2 GPU each that are already available in the EVANNAI research group. Nevertheless, this delay could never be longer than 12 months, since the limited capabilities of these 4 GPU devices would impose a significant delay in the overall project that would jeopardize its completion.

WP4: Research, analysis, design and development of a system for coordinating GPU resources

This working package tackles general goal GG.2 and specific goals SG.2 and SG.3. The successful achievement of this working package is critical for working package WP5, which relies on the GPU resources coordination system, namely DeepRector, to run the neuroevolutionary system.

In order to work in this package, we have identified the following tasks:

Task T4.1 *Requirements specification.* In the first place, we will specify the requirements that the DeepRector system for coordinating GPU resources must fulfill. This specification will clearly indicate which information must be sent by the worker nodes (those with available hardware resources for running processes) to the coordination system for proper functioning.

Task T4.2 *Design of the coordination system.* It is critical that DeepRector has a carefully-thought design so that it can fulfill the needs of clients requesting the execution of deep learning processes. This stage is critical, since it is required to specify the criteria followed by DeepRector to choose the most convenient resources to run a given process. Moreover, we need to design the interface that will enable customers to communicate with DeepRector, and how this system will proceed when potential failures arise in DeepForce (lack of resources availability, hangouts, etc.) in order to not negatively affect the process execution.

Task T4.3 *Development of the coordination system.* Based on the requirements defined in task T4.1 and the design specified in task T4.2, we will proceed to the development of DeepRector.

Task T4.4 *Integration and testing.* Following the scheme for continuous integration we will incorporate the development to the production system and will perform thorough testing in order to guarantee the successful fulfillment of the requirements.

Deliverable D4.1 *Testing and results report*

Milestone M4.1 *DeepRector available and successfully working.* This milestone is crucial for the development of working package WP5.

Risks We have not identified remarkable risks or difficulties in this working package.

WP5: Research and development of a neuroevolutionary system

This working package tackles the general goal GG.1 and specific goals SG.4 and SG.5. The neuroevolutionary system DeepEvol is the core of the whole project, constituting the innovative piece of software that can make the difference with other commercial proposals. In this package we will try to validate the working hypothesis studying the possibility of carrying out the automatic design of DNN and CNN topologies using evolutionary computation to improve their performance.

In order to work in this package, we have identified the following tasks:

Task T5.1 *Formal definition of the encoding.* The first step involves determining the most appropriate evolutionary computation technique for evolving the topologies and the encoding required to represent DNNs and CNNs in a way that can be processed by DeepEvol. This encoding must be domain-independent, though it can incorporate some domain-based heuristics in order to improve the evolutionary process. Moreover, it must be flexible when observing the largest possible variety of DNN and CNN architectures, attaining a tradeoff that reduces redundancy and keeps the search space within boundaries.

Task T5.2 *Requirements specification.* DeepEvol's requirements must be specified, explicitly describing the system functionality as well as non-functional validation criteria (efficiency, etc).

Task T5.3 *Development of the neuroevolutionary system.* Based on the requirements in task T5.2 and the encoding defined in task T5.1, we will develop the DeepEvol neuroevolutionary system.

Task T5.4 *Integration and testing.* Following the scheme for continuous integration we will incorporate the development to the production system and will perform thorough testing in order to guarantee the successful fulfillment of the requirements.

Deliverable D5.1 *Testing and results report*

Milestone M5.1 *DeepEvol available and successfully working.* This milestone is essential since DeepEvol is the core of the current project proposal.

Risks We have not identified remarkable risks or difficulties in this working package.

WP6: Development of an interface for accessing DeepNE

In order to increase the usability of the proposed system it is required to develop an interface that simplifies the access and basic setup of DeepNE to users not familiarized with deep learning, tackling specific goal SG.6. To do so, we will tackle the following tasks:

Task T6.1 *Requirements specification.* In the first place, we will define the requirements for the interface. The objective is that such interface simplifies the whole machine learning process.

Task T6.2 *Development of the access interface.* Based on the requirements of task T6.1, we will develop the interface, which must be simple to use.

Task T6.3 *Integration and testing.* Following the scheme for continuous integration we will incorporate the development to the production system and will perform thorough testing in order to guarantee the successful fulfillment of the requirements.

Risks We have not identified remarkable risks or difficulties in this working package.

WP7: Integration and validation of the developed system

This working package is in charge of the final integration of all the software and hardware components and the validation of the working hypotheses that supports this research project. Once the different modules are integrated, we will proceed to select datasets from real-world problems

that are representative of different application areas. These problems will allow us to thoroughly evaluate the deep learning system and compare the obtained results with the ones reported in the scientific literature. If the working hypotheses hold, we expect to be able to improve the best results found so far in diverse areas.

Task T7.1 *Integration of the developed modules.* This task is actually done all over the project execution and is finally consolidated at this stage. Given that the system comprises three modules and an access interface it is important to test its behavior when these modules work as a whole. As many tests as required will be performed to ensure the proper behavior of the system.

Task T7.2 *Selection of representative validation domains.* For the validation of the proposed system it is needed to test its behavior on representative domains. To this extent, based on the research done on WP2, task T2.1, we will identify the most relevant datasets with a larger number of bibliographic references in order to tackle them with DeepNE, thus allowing a comparison of the results obtained with our system with those reported in the state of the art. An additional criterion when choosing the validation domains will be its area of application, prioritizing the diversity of areas: images, biomedical signals, physical activity, mixed data, etc.

Task T7.3 *Execution and empirical validation of the selected domains.* Once the domains are selected, we can systematically test DeepNE over those domains. Once the evaluation is completed, we will compare those with the ones in the state of the art. In case our results are competitive with the state of the art, we can state that the working hypotheses are proved.

Deliverable D7.1 *Comparative analysis of DeepNE's performance against other techniques*

Milestone M7.1 *Proved working hypotheses.* This is the most relevant milestone in the whole project, as it will enable to confirm whether all the work done to implement the DeepNE system was worthy. Based on the previous knowledge of the research team and the exploratory approaches done so far, the preliminary results are encouraging. Nevertheless, a hardware-software infrastructure is required to confirm this point.

Risks Despite having a very powerful hardware infrastructure, the automatic optimization of DNN and CNN designs will require large amounts of time when tested against large datasets. It is essential to carefully pick up the validation sets to avoid very large running times and not extending this working package beyond the schedule.

Contingency plan In case of identifying delays in some DeepNE instances we are considering two options: 1) accelerating the computations by using sampling, but incurring on a loss of precision in the results, and 2) requesting an extension of 2-3 months to the Ministry of Economy, Industry and Competitiveness.

WP8: Project dissemination

This working package, which is extended throughout the whole project duration, is considered extremely important. Initially, a website will be developed including the project description and

the contact information. Also, accounts in the main specialized social networks (such as LinkedIn) will be created in order to announce the main project milestones. Moreover, we will give visibility to the project by means of UC3M's Scientific Information Office. Also, we will communicate the project results to national and international forums specialized in artificial intelligence and machine learning. We plan to publish partial results and advances throughout the project execution, and establish as final task the presentation and publication of all the validation results.

Task T8.1 *Design and publication of a website.* We will use attractive templates to promote the project and publish news regarding its level of execution.

Task T8.2 *Dissemination in social networks.* We will provide project information in LinkedIn and create a Twitter account to report general information about Deep Neuroevolution.

Task T8.3 *Dissemination in a press release.* We will promote the project via UC3M's Scientific Information Office by means of an informative video and a press release in several languages.

Task T8.4 *Dissemination in scientific conferences.* The publication and presentation of the work in scientific conferences and congresses constitute an enriching discussion forum and helps to disseminate information about the project.

Task T8.5 *Publication of results in scientific journals.* This task is considered essential as it would imply the high interest of the scientific community in DeepNE project. We expect to publish related papers in journals with high impact factor (JCR Q1 and Q2) with open-access policies for achieving a larger impact and dissemination.

C.5 Materials, Infrastructure and Equipment

At this moment, EVANNAI research group owns laptops and PCs, and a rack of servers installed in UC3M's computing centre. In particular, this rack involves a domain server, a dedicated cluster for creating a pool of virtual machines, and two disk servers with RAID redundancy for data storage. These servers can only perform CPU processing and store the project website, but they are not designed for deep learning, given that the computational resources required for training deep and convolutional neural networks are very high.

In the last year, we have acquired two nodes with two GPU devices each, which are specifically configured for performing deep learning tasks. When using these two nodes, a deep neural networks with a very small dataset (e.g., MNIST), the network requires several hours to train. When very large datasets are involved, the process could take up to months. It is for this reason that in order to perform this research it is strictly required to have suitable hardware at our disposition. The minimum requirements, which will be included in the budget allocation, are the following:

- Rack for deep learning computing servers, an example of this rack would be: Rack 19" 42U 600 x 1200 CPD ImServ[1] (*€1,194.72*).

[1] https://www.rackonline.es/armario-rack-imserv/rack-19-42u-600-x-1200-cpd-imserv.html

- Servers configured for GPU computing, such as the following: 4 x PXXT10-2260V4-4GPU Tesla P100 16 GB HBM2[2] (*4 x €23,713.02*). The price is subject to changes, since the manufacturer can make discounts for academic institutions. In that case, the discount would be use to acquire equipment with better specifications for the same budget.
- Software licenses of the project management tool Atlassian JIRA (*36 x €10*) and the collaborative tool ShareLatex (*2 x €168*). All the remaining software tools will be available for free, and published under opensource licenses when possible.

C.6 Schedule

The schedule for the DeepNE project is detailed in table C.2.

DeepNE Schedule	Month																	
	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36
WP1: Management	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
T1.1 Establishing the methodology	•																	
T1.2 Project control and follow-up	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
T1.3 Project coordination	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
T1.4 Project reports						•						•						•
WP2: Research of the state-of-the-art	•	•	•															
T2.1 Techniques for automatic design of DNNs	•	•	•															
T2.2 Software technologies for deep learning	•	•	•															
T2.3 Hardware for deep learning	•	•	•															
WP3: DeepForce		•	•	•														
T3.1 Definition of technical specifications		•																
T3.2 Procedure for contracting of suppliers		•	•															
T3.3 Physical deployment				•														
T3.4 Software configuration				•														
WP4: DeepRector				•	•	•												
T4.1 Requirements specification				•														
T4.2 Design				•														
T4.3 Development					•	•												
T4.4 Integration and testing						•												
WP5: DeepEvol				•	•	•	•	•	•	•	•	•	•	•				
T5.1 Formal encoding definition				•	•	•												
T5.2 Requirements specification						•	•	•										
T5.3 Development								•	•	•	•	•	•					
T5.4 Integration and testing													•	•				
WP6: DeepNE access interface											•	•	•					
T6.1 Requirements specification											•							
T6.2 Development												•	•					
T6.3 Integration and testing													•					
WP7: Integration and validation													•	•	•	•	•	•
T7.1 Integration													•	•	•			
T7.2 Domains selection														•	•			
T7.3 Evaluation and validation															•	•	•	•
WP8: Dissemination	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
T8.1 Website	•					•						•						•
T8.2 Social networks	•			•		•						•		•				•
T8.3 Press release				•													•	•
T8.4 Scientific conferences	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
T8.5 Publication in academic journals	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

Table C.2: *DeepNE project schedule.*

[2] http://www.thinkmate.com/system/gpx-xt10-2260v4-4gpu

C.7 Human Resources

This project requires hiring an experienced computer engineer (preferably with a M.Sc. degree) in order to carry out, to a large extent, the software development of the different parts of DeepNE and its validation. The research team members will mostly be focused in those areas requiring specific deep learning expertise. They will first focus on the research of techniques and related technologies (WP2). Later, the equipment will be acquired, installed and deployed. By this time, a person will be already hired will full-time dedication to the project. It is estimated that this person will be working for the project during two years and a half, with a weekly dedication of 35 hours.

C.8 Scientific, Social and Economic Impact

Since TensorFlow code was liberated by Google, artificial intelligence is experiencing a revolution like never before, with new techniques and applications based on deep learning. News are coming from Asia and USA on the new advancements on these topics, and DeepNE project aims at bringing that innovation to Spain. DeepNE project and the future activities that can be derived from its use will have an unprecedented impact, given that so far there are not (1) baremetal-as-a-service suppliers for deep learning, (2) a system for integrating and balancing deep learning tasks with a pool of available GPU resources or (3) software tools able to design complex DNN and CNN topologies, optimizing them to dynamically obtain the best performance for a problem at hand.

The DeepNE project will enable EVANNAI research group and Universidad Carlos III de Madrid to be at the forefront of innovation in a key technology that, supported by specific hardware, will be made available to the Spanish universities network, research centres and technological companies; entities that have already shown their interest in using a tool of these characteristics. The future commercialization of this system in a pay-per-use mode will be key to guarantee its continuity, progress and growth.

Moreover, the system modularity enables its commercial exploitation at three different levels:

- The on-demand renting of hardware resources (DeepForce), offering virtualized instances with direct access to GPU devices for their use. The price would be established based on the rented resources and the renting time. This approach is similar to that offered by cloud infrastructure-as-a-service (IaaS) providers, yet current solutions have a cost which is right now excessive for long projects and a performance with significant room for improvement when considering deep learning tasks.
- DeepRector platform can be offered as a service, enabling researchers and developers to directly address this platform with their requests, which will be run over the hardware resources available at DeepForce. In this case, the platform users do not need to deal with low-level hardware; instead, DeepRector provides an abstraction and returns the result of their request. To our best knowledge, there are not similar proposals in the market.
- DeepEvol software can be offered as a service, enabling researchers and developers not familiarized with deep learning to just input their data and the quality metric to be optimized, and obtaining as a result the optimal topology for solving their problem. This software is clearly innovative, as there are no similar proposals at the moment.

As we have outlined before, the completion of this project will end up with a full hardware-software environment that will allow to work on a large number of research applications in very

diverse areas where the success of deep learning techniques has been already proven: text and sound recognition, biomedical signals processing, images, etc. An environment such as DeepNE would enable significant research advances on many diverse fields due to the great computing capabilities it offers and the flexibility of the neuroevolutionary system proposed (DeepEvol).

Also, this system is a pioneer in proposing an experimental architecture to allocate and coordinate processes between the different GPU devices available in DeepForce. This architecture will support the further development of applications that, just like DeepEvol, are sustained over DeepRector offering new research potentials.

As a consequence of all this, DeepNE project will offer the research team the potential to become an international reference, by significantly accelerating deep learning based research.

C.9 Dissemination Plan

The relevance of this project requires a careful dissemination plan that makes academia and the general society aware of the DeepNE features and capabilities.

In order to define a careful dissemination and internationalization plan, we have proposed a specific working package. The tasks and milestones involved in this plan are detailed in the description of working package WP8.

C.10 Results Transfer

The pay-per-use commercialization of DeepNE (either as the on-demand renting of DeepForce hardware resources or the offer of DeepRector or DeepEvol as services) can be of a great interest for both the public and private sectores, by allowing them to advance on their research lines and specific developments where they require GPU resources with high computing capabilities or a specific platform for running their deep learning applications. In due time, we will plan the commercialization of DeepNE system through UC3M's Office for Transfer of Research Results.

Finally, after an initial analysis we have observed that the Spanish Network of Supercomputing Centers do not have specific hardware resources for deep learning. While this proposal suggests the creation of a small-scaled computing centre for deep learning, the acquired know-how and expertise during its deployment could be transferred in the future to those entities that have means to establish a supercomputing center for deep learning, or to extend our center in future project proposals to increase its computing power.

This page has been intentionally left blank.

Appendix D

Accountability

The work in this Ph.D. thesis is partially supported by public funding from the Spanish Ministry of Education, Culture and Sports. Every year, the Ph.D. candidate has been required to inform in due time both the Spanish Government and *Universidad Carlos III de Madrid* about the advances in this thesis and the highlights in the author's resume.

Nevertheless, we have considered an important duty to be accountable about the work performed during these years, beyond the research presented in this document and the document itself. As a result, this appendix describes the candidate's scientific production over these years, the teaching tasks performed by him and other relevant merits. Finally, this appendix also describes the legal background of the public funding scheme behind this thesis and describes the documents delivered to the different public institutions when required to comply with the candidate's obligations as the recipient of this funding.

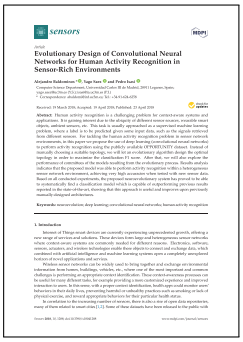
D.1 Publications

First, we will enumerate all the articles published from Nov. 2014 (the data in which the candidate enrolled in the Ph.D. programme) to the current date, including papers in journals, conferences, book chapters, etc. They will be listed in descending chronological order for each of these sections, along with their abstracts. This list is exhaustive inasmuch as papers contained in it might not necessarily be related with this thesis, but rather be products of other research activities or teaching (e.g., papers published after a B.Sc. thesis).

At the time of depositing this Ph.D. thesis, the candidate have achieved the following metrics in his academic record:

- **Citations**: 73 (Source: Google Scholar [125])
- **h-index**: 5 (Source: Google Scholar [125])
- **i10-index**: 2 (Source: Google Scholar [125])
- **Citations in 2016–2017**: 48 (Source: Google Scholar [125])
- **Citations in 2018**: 13 (Source: Google Scholar [125])
- **RG Score**: 10.60 (Source: ResearchGate [301])

D.1.1 Journals Indexed in the Journal Citation Report



A. Baldominos, Y. Sáez and P. Isasi

Evolutionary Design of Convolutional Neural Networks for Human Activity Recognition in Sensor-Rich Environments

Apr 2018 · ***Sensors*** 18:4, pp. 1288

I.F. 2017: **2.475** | **Q2** Instruments and Instrumentation

Human activity recognition is a challenging problem for context-aware systems and applications. It is gaining interest due to the ubiquity of different sensor sources, wearable smart objects, ambient sensors, etc. This task is usually approached as a supervised machine learning problem, where a label is to be predicted given some input data, such as the signals retrieved from different sensors.

For tackling the human activity recognition problem in sensor network environments, in this paper we propose the use of deep learning (convolutional neural networks) to perform activity recognition using the publicly available OPPORTUNITY dataset. Instead of manually choosing a suitable topology, we will let an evolutionary algorithm design the optimal topology in order to maximize the classification F1 score. After that, we will also explore the performance of committees of the models resulting from the evolutionary process.

Results analysis indicates that the proposed model was able to perform activity recognition within a heterogeneous sensor network environment, achieving very high accuracies when tested with new sensor data. Based on all conducted experiments, the proposed neuroevolutionary system has proved to be able to systematically find a classification model which is capable of outperforming previous results reported in the state-of-the-art, showing that this approach is useful and improves upon previously manually-designed architectures.

A. Baldominos, Y. Sáez and P. Isasi

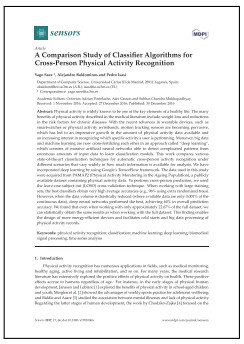
Evolutionary convolutional neural networks: An application to handwriting recognition

Mar 2018 · ***Neurocomputing*** 283, pp. 38–52

I.F. 2017: **3.241** | **Q1** Computer Science, Artificial Intelligence

Convolutional neural networks (CNNs) have been used over the past years to solve many different artificial intelligence (AI) problems, providing significant advances in some domains and leading to state-of-the-art results. However, the topologies of CNNs involve many different parameters, and in most cases, their design remains a manual process that involves effort and a significant amount of trial and error.

In this work, we have explored the application of neuroevolution to the automatic design of CNN topologies, introducing a common framework for this task and developing two novel solutions based on genetic algorithms and grammatical evolution. We have evaluated our proposal using the MNIST dataset for handwritten digit recognition, achieving a result that is highly competitive with the state-of-the-art without any kind of data augmentation or preprocessing. When misclassified samples are carefully observed, it is found that most of them involve handwritten digits that are difficult to recognize even by a human.



Y. Sáez, A. Baldominos and P. Isasi

A Comparison Study of Classifier Algorithms for Cross-Person Physical Activity Recognition

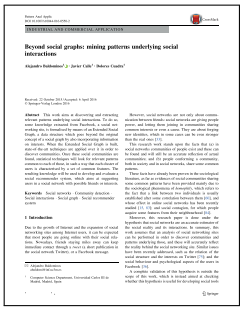
Dec 2016 · ***Sensors*** 17:1, pp. 66

I.F. 2016: **2.677** | **Q1** Instruments and Instrumentation

Physical activity is widely known to be one of the key elements of a healthy life. The many benefits of physical activity described in the medical literature include weight loss and reductions in the risk factors for chronic diseases. With the recent advances in wearable devices, such as smartwatches or physical activity wristbands, motion tracking sensors are becoming pervasive, which has led to an impressive growth in the amount of physical activity data available and an increasing interest in recognizing which specific activity a user is performing. Moreover, big data and machine learning are now cross-fertilizing each other in an approach called "deep learning", which consists of massive artificial neural networks able to detect complicated patterns from enormous amounts of input data to learn classification models.

This work compares various state-of-the-art classification techniques for automatic cross-person activity recognition under different scenarios that vary widely in how much information is available for analysis. We have incorporated deep learning by using Google's TensorFlow framework. The data used in this study were acquired from PAMAP2 (Physical Activity Monitoring in the Ageing Population), a publicly available dataset containing physical activity data. To perform cross-person prediction, we used the leave-one-subject-out (LOSO) cross-validation technique.

When working with large training sets, the best classifiers obtain very high average accuracies (e.g., 96 % using extra randomized trees). However, when the data volume is drastically reduced (where available data are only 0.001 % of the continuous data), deep neural networks performed the best, achieving 60 % in overall prediction accuracy.We found that even when working with only approximately 22.67 % of the full dataset, we can statistically obtain the same results as when working with the full dataset. This finding enables the design of more energy-efficient devices and facilitates cold starts and big data processing of physical activity records.



A. Baldominos, F.J. Calle and D. Cuadra

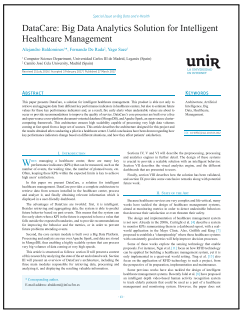
Beyond Social Graphs: Mining Patterns Underlying Social Interactions

Apr 2016 · ***Pattern Anal Applic*** 20:1, pp. 269–285

I.F. 2016: **1.352** | **Q3** Computer Science, Artificial Intelligence

This work aims at discovering and extracting relevant patterns underlying social interactions. To do so, some knowledge extracted from Facebook, a social networking site, is formalised by means of an Extended Social Graph, a data structure which goes beyond the original concept of a social graph by also incorporating information on interests. When the Extended Social Graph is built, state-of-the-art techniques are applied over it in order to discover communities. Once these social communities are found, statistical techniques will look for relevant patterns common to each of those, in such a way that each cluster of users is characterised by a set of common features. The resulting knowledge will be used to develop and evaluate a social recommender system, which aims at suggesting users in a social network with possible friends or interests.

D.1.2 Other Journals


DataCare: Big Data Analytics Solution for Intelligent Healthcare Management

A. Baldominos, F. Rada and Y. Sáez

DataCare: Big Data Analytics Solution for Intelligent Healthcare Management

Mar 2018 · ***IJIMAI*** 4:7, pp. 13–20

This paper presents DataCare, a solution for intelligent healthcare management. This product is able not only to retrieve and aggregate data from different key performance indicators in healthcare centers, but also to estimate future values for these key performance indicators and, as a result, fire early alerts when undesirable values are about to occur or provide recommendations to improve the quality of service. DataCare's core processes are built over a free and open-source cross-platform document-oriented database (MongoDB), and Apache Spark, an open-source cluster-computing framework. This architecture ensures high scalability capable of processing very high data volumes coming at fast speed from a large set of sources. This article describes the architecture designed for this project and the results obtained after conducting a pilot in a healthcare center. Useful conclusions have been drawn regarding how key performance indicators change based on different situations, and how they affect patients' satisfaction.


Real-Time Prediction of Gamers Behavior Using Variable Order Markov and Big Data Technology: A Case of Study

A. Baldominos, A. Albacete, I. Marrero and Y. Sáez

Real-Time Prediction of Gamers Behavior Using Variable Order Markov and Big Data Technology: A Case of Study

Mar 2016 · ***IJIMAI*** 3:6, pp. 44–51

This paper presents the results and conclusions found when predicting the behavior of gamers in commercial videogames datasets. In particular, it uses Variable-Order Markov (VOM) to build a probabilistic model that is able to use the historic behavior of gamers and to infer what will be their next actions. Being able to predict with accuracy the next user's actions can be of special interest to learn from the behavior of gamers, to make them more engaged and to reduce churn rate. In order to support a big volume and velocity of data, the system is built on top of the Hadoop ecosystem, using HBase for real-time processing; and the prediction tool is provided as a service (SaaS) and accessible through a RESTful API. The prediction system is evaluated using a case of study with two commercial videogames, attaining promising results with high prediction accuracies.


An approach to physical rehabilitation using state-of-the-art virtual reality and motion tracking technologies

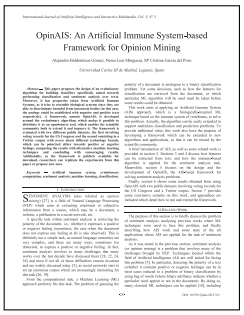
A. Baldominos, Y. Sáez and C. García del Pozo

An Approach to Physical Rehabilitation Using State-of-the Art Virtual Reality and Motion Tracking Technologies

Oct 2015 · ***Procedia Computer Science*** 64, pp. 10–16

This paper explores an approach to physical rehabilitation using state-of-the-art technologies in virtual reality and motion tracking; in particular, Oculus Rift DK2 (released in July, 2014) and Intel RealSense (released in November, 2014) are used. A game is developed which requires from the patient to perform an established set of abduction and adduction arm movements to achieve rotator cuff rehabilitation after injury. While conduct of clinical trials is outside the scope of this work, experts

in physical rehabilitation working in the medical field have carried out a preliminary evaluation, showing encouraging results.



A. Baldominos, N. Luis and C. García del Pozo

OpinAIS: An Artificial Immune System-based Framework for Opinion Mining

Jun 2015 · ***IJIMAI*** 3:3, pp. 25–34

This paper proposes the design of an evolutionary algorithm for building classifiers specifically aimed towards performing classification and sentiment analysis over texts. Moreover, it has properties taken from Artificial Immune Systems, as it tries to resemble biological systems since they are able to discriminate harmful from innocuous bodies (in this case, the analogy could be established with negative and positive texts respectively). A framework, namely OpinAIS, is developed around the evolutionary algorithm, which makes it possible to distribute it as an open-source tool, which enables the scientific community both to extend it and improve it. The framework is evaluated with two different public datasets, the first involving voting records for the US Congress and the second consisting in a Twitter corpus with tweets about different technology brands, which can be polarized either towards positive or negative feelings; comparing the results with alternative machine learning techniques and concluding with encouraging results. Additionally, as the framework is publicly available for download, researchers can replicate the experiments from this paper or propose new ones.

D.1.3 Conference Papers

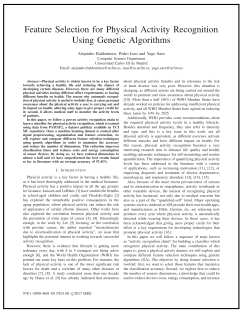
A. Baldominos, M. González-Evstrópova, E. Martín and Y. Sáez

Planet Wars: an Approach Using Ant Colony Optimization

Oct 2018 · ***META'18***, *In press* · Marrakech, Morocco

This paper describes an application of Ant Colony Optimization (ACO), a wellknown biologically-inspired technique for graph search, for solving a complex multi-objective task posed in the form of a video game, namely PlanetWars, which is itself based on a popular video game known as Galcon and was presented in the Google AI Challenge 2010.

Throughout the paper we introduce the game mechanics and rules and describe how we haveapplied ACO to solve it, by working on different strategies (expansion, defense and troops rebalancing) and different heuristics which are later submitted to evaluation and whose results are discussed by the end of the paper. These results show that the best performing heuristic is able to beat at least half of the times the Google baseline bots, while in some cases they are defeated in all games and maps.


Feature Selection for Physical Activity Recognition Using Genetic Algorithms

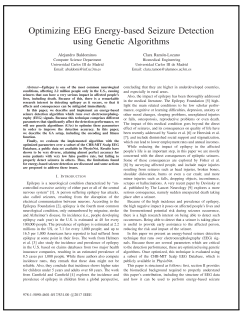
A. Baldominos, P. Isasi and Y. Sáez

Feature Selection for Physical Activity Recognition Using Genetic Algorithms

Jun 2017 · ***IEEE CEC'17***, pp. 2185–2192 · Donostia, Spain

Physical activity is widely known to be a key factor towards achieving a healthy life and reducing the chance of developing certain diseases. However, there are many different physical activities having different effort requirements or having different benefits on health. The reason why automatic recognition of physical activity is useful is twofold: first, it raises personal awareness about the physical activity a user is carrying out and its impact on health, allowing some apps to give proper credit for it; second, it allows medical staff to monitor the activity levels of patients.

In this paper, we follow a proven activity recognition chain to learn a classifier for physical activity recognition, which is trained using data from PAMAP2, a dataset publicly available in UCI ML repository. Once a machine learning dataset is created after signal preprocessing, segmentation and feature extraction, we will explore and compare different feature selection techniques using genetic algorithms in order to maximize the accuracy and reduce the number of dimensions. This reduction improves classification times and reduces costs and energy consumption of sensor devices. By doing so, we have reduced dimensions to almost a half and we have outperformed the best results found so far in literature with an average accuracy of 97.45 %.


Optimizing EEG Energy-based Seizure Detection using Genetic Algorithms

A. Baldominos and C. Ramón-Lozano

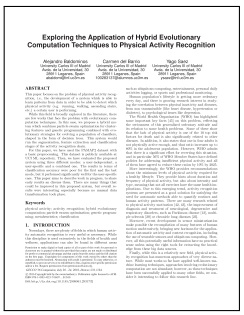
Optimizing EEG Energy-based Seizure Detection using Genetic Algorithms

Jun 2017 · ***IEEE CEC'17***, pp. 2338–2345 · Donostia, Spain

Epilepsy is one of the most common neurological conditions, affecting 2.2 million people only in the U.S., causing seizures that can have a very serious impact in affected people's lives, including death. Because of this, there is a remarkable research interest in detecting epilepsy as it occurs, so that it effects and consequences can be mitigated immediately.

In this paper, we describe and implement an energy-based seizure detection algorithm which runs over electroencephalography (EEG) signals. Because this technique comprises different parameters that significantly affect the detection performance, we will use genetic algorithms (GAs) to optimize these parameters in order to improve the detection accuracy.

In this paper, we describe the GA setup, including the encoding and fitness function. Finally, we evaluate the implemented algorithm with the optimized parameters over a subset of the CHB-MIT Scalp EEG Database, a public data set available in PhysioNet. Results have shown to be very diverse, attaining almost perfect accuracy for some patients with very low false positive rate, but failing to properly detect seizures in others. Thus, the limitations found for energy-based seizure detection are discussed and some actions are proposed to address these issues.


Exploring the Application of Hybrid Evolutionary Computation Techniques to Physical Activity Recognition

A. Baldominos, C. del Barrio and Y. Sáez

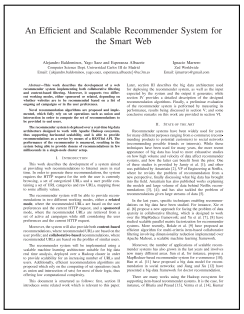
Exploring the Application of Hybrid Evolutionary Computation Techniques to Physical Activity Recognition

Jul 2016 · ***ACM GECCO'16***, pp. 1377–1384 · Denver (CO), USA

This paper focuses on the problem of physical activity recognition, i.e., the development of a system which is able to learn patterns from data in order to be able to detect which physical activity (e.g. running, walking, ascending stairs, etc.) a certain user is performing.

While this field is broadly explored in the literature, there are few works that face the problem with evolutionary computation techniques. In this case, we propose a hybrid system which combines particle swarm optimization for clustering features and genetic programming combined with evolutionary strategies for evolving a population of classifiers, shaped in the form of decision trees. This system would run the segmentation, feature extraction and classification stages of the activity recognition chain.

For this paper, we have used the PAMAP2 dataset with a basic preprocessing. This dataset is publicly available at UCI ML repository. Then, we have evaluated the proposed system using three different modes: a user-independent, a user-specific and a combined one. The results in terms of classification accuracy were poor for the first and the last mode, but it performed significantly well for the user-specific case. This paper aims to describe work in progress, to share early results an discuss them. There are many things that could be improved in this proposed system, but overall results were interesting especially because no manual data transformation took place.


An Efficient and Scalable Recommender System for the Smart Web

A. Baldominos, Y. Sáez, E. Albacete and I. Marrero

An Efficient and Scalable Recommender System for the Smart Web

Nov 2015 · ***IIT'15***, pp. 296–301 · Dubai, UAE

This work describes the development of a web recommender system implementing both collaborative filtering and content-based filtering. Moreover, it supports two different working modes, either sponsored or related, depending on whether websites are to be recommended based on a list of ongoing ad campaigns or in the user preferences. Novel recommendation algorithms are proposed and implemented, which fully rely on set operations such as union and intersection in order to compute the set of recommendations to be provided to end users. The recommender system is deployed over a real-time big data architecture designed to work with Apache Hadoop ecosystem, thus supporting horizontal scalability, and is able to provide recommendations as a service by means of a RESTful API. The performance of the recommender is measured, resulting in the system being able to provide dozens of recommendations in few milliseconds in a single-node cluster setup.

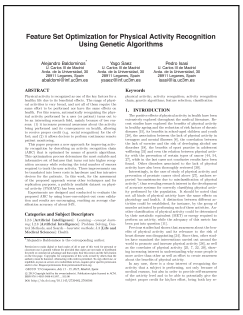
A. Baldominos, P. Isasi, Y. Sáez and B. Manderick

Monte Carlo Schemata Searching for Physical Activity Recognition

Sep 2015 · ***INCoS'15***, pp. 176–183 · Taipei, Taiwan

Medical literature have recognized physical activity as a key factor for a healthy life due to its remarkable benefits. However, there is a great variety of physical activities and not all of them have the same effects on health nor require the same effort. As a result, and due to the ubiquity of commodity devices able to track users' motion, there is an increasing interest on performing activity recognition in order to detect the type of activity carried out by the subjects and being able to credit them for their effort, which has been detected as a key requirement to promote physical activity.

This paper proposes a novel approach for performing activity recognition using Monte Carlo Schemata Search (MCSS) for feature selection and random forests for classification. To validate this approach we have carried out an evaluation over PAMAP2, a public dataset on physical activity available in UCI Machine Learning repository, enabling replication and assessment. The experiments are conducted using leave-one-subject-out cross validation and attain classification accuracies of over 93 % by using roughly one third of the total set of features. Results are promising, as they outperform those obtained in other works on the same dataset and significantly reduce the set of features used, which could translate in a decrease of the number of sensors required to perform activity recognition and, as a result, a reduction of costs.



A. Baldominos, Y Sáez and P. Isasi

Feature Set Optimization for Physical Activity Recognition Using Genetic Algorithms

Jul 2015 · ***ACM GECCO'15***, pp. 1311–1318 · Madrid, Spain

Physical activity is recognized as one of the key factors for a healthy life due to its beneficial effects. The range of physical activities is very broad, and not all of them require the same effort to be performed nor have the same effects on health. For this reason, automatically recognizing the physical activity performed by a user (or patient) turns out to be an interesting research field, mainly because of two reasons: (1) it increases personal awareness about the activity being performed and its consequences on health, allowing to receive proper credit (e.g. social recognition) for the effort; and (2) it allows doctors to perform continuous remote patient monitoring.

This paper proposes a new approach for improving activity recognition by describing an activity recognition chain (ARC) that is optimized by means of genetic algorithm. This optimization process determines the most suitable and informative set of features that turns out into higher recognition accuracy while reducing the total number of sensors required to track the user activity. These improvements can be translated into lower costs in hardware and less intrusive devices for the patients. In this work, for the assessment of the proposed approach versus other techniques and for replication purposes, a publicly available dataset on physical activity (PAMAP2) has been used.

Experiments are designed and conducted to evaluate the proposed ARC by using leave-one-subject-out cross validation and results are encouraging, reaching an average classification accuracy of about 94 %.

A. Baldominos, E. Albacete, Y Sáez and P. Isasi

A Scalable Machine Learning Online Service for Big Data Real-Time Analysis

Dec 2014 · ***IEEE SSCI'14***, pp. 1–8 · Orlando (FL), USA

This work describes a proposal for developing and testing a scalable machine learning architecture able to provide real-time predictions or analytics as a service over domain-independent big data, working on top of the Hadoop ecosystem and providing real-time analytics as a service through a RESTful API. Systems implementing this architecture could provide companies with on-demand tools facilitating the tasks of storing, analyzing, understanding and reacting to their data, either in batch or stream fashion; and could turn into a valuable asset for improving the business performance and be a key market differentiator in this fast pace environment.

In order to validate the proposed architecture, two systems are developed, each one providing classical machine-learning services in different domains: the first one involves a recommender system for web advertising, while the second consists in a prediction system which learns from gamers' behavior and tries to predict future events such as purchases or churning. An evaluation is carried out on these systems, and results show how both services are able to provide fast responses even when a number of concurrent requests are made, and in the particular case of the second system, results clearly prove that computed predictions significantly outperform those obtained if random guess was used.

D.1.4 Book Chapters

A. Baldominos, Y. Sáez, G. Recio and F. Calle

Learning Levels of Mario AI Using Genetic Algorithms

Nov 2015 · ***Lecture Notes in Computer Science*** 9422 pp. 267–277 · Springer

This paper introduces an approach based on Genetic Algorithms to learn levels from the Mario AI simulator, based on the Infinite Mario Bros. game (which is, at the same time, based on the Super Mario World game from Nintendo). In this approach, an autonomous agent playing Mario is able to learn a sequence of actions in order to maximize the score, not looking at the current state of the game at each time.

Different parameters for the Genetic Algorithm are explored, and two different stages are executed: in the first, domain independent genetic operators are used; while in the second knowledge about the domain is incorporated to these operators in order to improve the results.

Results are encouraging, as Mario is able to complete very difficult levels full of enemies, resembling the behavior of an expert human player.

D.1.5 Books

A. Baldominos

Procesamiento y Análisis Inteligente de Big Data

Jul 2017 · García-Maroto Editores

Este libro es una guía completa y esencial para realizar una primera aproximación al procesamiento y análisis inteligente de Big Data. Los conceptos se presentan con un enfoque introductorio, adecuado para aquellos lectores que deseen adentrarse en este fascinante mundo, y se complementan con casos prácticos para que el lector pueda ponerse "manos a la obra" lo antes posible. Si se trabajan los temas señalados, los lectores serán capaces de:

- Conocer los paradigmas de procesamiento de datos por lotes y en tiempo real.
- Entender el paradigma MapReduce y plantear soluciones a problemas de procesamiento de datos.
- Familiarizarse con el análisis predictivo y el análisis de patrones, pudiendo proponer soluciones de inteligencia artificial y "machine learning" a problemas de negocio.
- Emplear de un modo básico el ecosistema Spark para el procesamiento y análisis de Big Data.
- Conocer las infinitas posibilidades de los servicios de procesamiento y análisis de Big Data en la nube.

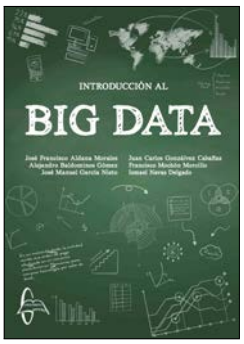
A. Baldominos

Almacenamiento de Big Data

Jun 2017 · García-Maroto Editores

Este libro resulta una guía fundamental para adentrarse en el mundo del almacenamiento de Big Data. Su enfoque introductorio resulta adecuado para aquellos lectores con conocimientos básicos en el mundo de las TIC, si bien aquellos lectores con más experiencia podrán continuar su formación con este libro. Si se trabajan de forma suficiente los temas señalados, los lectores serán capaces de:

- Entender conceptos básicos de organización de ficheros.
- Comprender el funcionamiento de sistemas de ficheros distribuidos.
- Realizar un despliegue básico de Hadoop para poder almacenar ficheros en un entorno Big Data.
- Identificar las principales tecnologías de bases de datos NoSQL, pudiendo determinar la elección más acertada para resolver un problema.
- Conocer el amplio abanico de servicios de almacenamiento en la nube.



J.F. Aldana, A. Baldominos, J.M. García, J.C. Gonzálvez, F. Mochón and I. Navas

Introducción al Big Data

Mar 2016 · García-Maroto Editores

Big Data permite aprovechar la inmensa cantidad de datos que se generan cada día, especialmente a raíz de la eclosión de las redes sociales online, del crecimiento exponencial de dispositivos y de las redes de sensores. Estos datos debidamente utilizados hacen posible que el proceso de toma de decisiones sea más objetivo, y se base menos en la intuición. Con Gig Data se pueden detectar tendencias, realizar predicciones de sucesos futuros, o extraer patroines del comportamiento de los usuarios, para adaptar mejor los servicios a sus necesidades.

El contenido del libro se ha estructurado de forma que se ofrezca una visión global de todos los temas que forman parte de un análisis de Big Data. El libro se ha concebido con un carácter eminentemente aplicado y en el último capítulo se presenta un caso de estudio completo. Por todo ello, el libro puede utilizarse como manual de referencia para realizar una primera toma de contacto con el análisis de Big Data.

El libro puede ser de interés tanto para lectores que no tengan una formación técnica, como para aquellos con formación o amplia experiencia en el mundo de las TIC. Los temas se presentan vía ejemplos de forma que fácilmente se pueden visualizar las posibilidades que ofrece Big Data. Pero los principios teóricos esbozados y el uso de las herramientas propuestas en el libro, constituyen una rigurosa introducción al manejo de los datos.

A. Baldominos

Herramientas Tecnológicas para la Empresa Digital

Jun 2015 · García-Maroto Editores

Vivimos en un mundo lleno de emprendedores, donde cada día se crean nuevas empresas que han detectado un nicho de negocio donde ejercer su actividad. Algunas tendrán éxito y otras fracasarán. ¿Podemos saber de primeras cuál será el futuro de nuestra empresa?

Dar respuesta a esta pregunta es difícil, porque el éxito depende de muchos factores. Sin embargo, hay un aspecto clave en el correcto funcionamiento de una empresa: el buen uso que haga de la tecnología.

¿Cómo puedo desarrollar mi primer sitio web corporativo? ¿Existen factores de diseño que puedan mejorar la experiencia de mis clientes? ¿Qué es la nube y cómo puede mi empresa aprovecharse de ella? ¿Puedo conocer el comportamiento de los usuarios de mi sitio y sacar provecho de ello? ¿Hay herramientas que permitan mejorar los procesos internos de mi empresa?

Este libro trata de dar respuesta a todas estas preguntas, presentando de una forma fundamentalmente práctica y mediante guías paso a paso, el uso de herramientas que permiten desarrollar el entorno digital de una compañía.

D.2 International Research Stays

The Ph.D. candidate visited the Massachusetts Institute of Technology (MIT) for a 6-months short stay starting in March 23, 2016 and ending in September 22, 2016, under a funding scheme regulated by the FPU grants for short stays with identifier EST15/00260.

The candidate was admitted to the ALFA (Anyscale Learning For All) Group of the Computer Science and Artificial Intelligence Laboratory (CSAIL), under the supervision of Prof. Una-May O'Reilly. The working place was located within Stata Center, a building designed by architect Frank Gehry in Cambridge, Massachusetts (see figure D.1). During the stay, the candidate participated in two research projects: GigaBEATS and Iron Deficiency Prediction.

D.2.1 GigaBEATS

The purpose of this project is to perform data science and machine learning with medical data.

In particular, the candidate was largely involved in the development of BeatDB [3], which is a tool that allows researchers to build predictive models through physiological waveform mining and analysis. One of its strengths is its flexibility, allowing users to define features to track, conditions to detect, and filters to apply with short user-defined scripts.

BeatDB is integrated within Amazon Web Services (AWS), allowing for users to run computations in parallel in the cloud. Because of these features, BeatDB allows researchers and scientists to

Figure D.1: *MIT's Stata Center. Source: original picture, taken on July 4, 2016.*

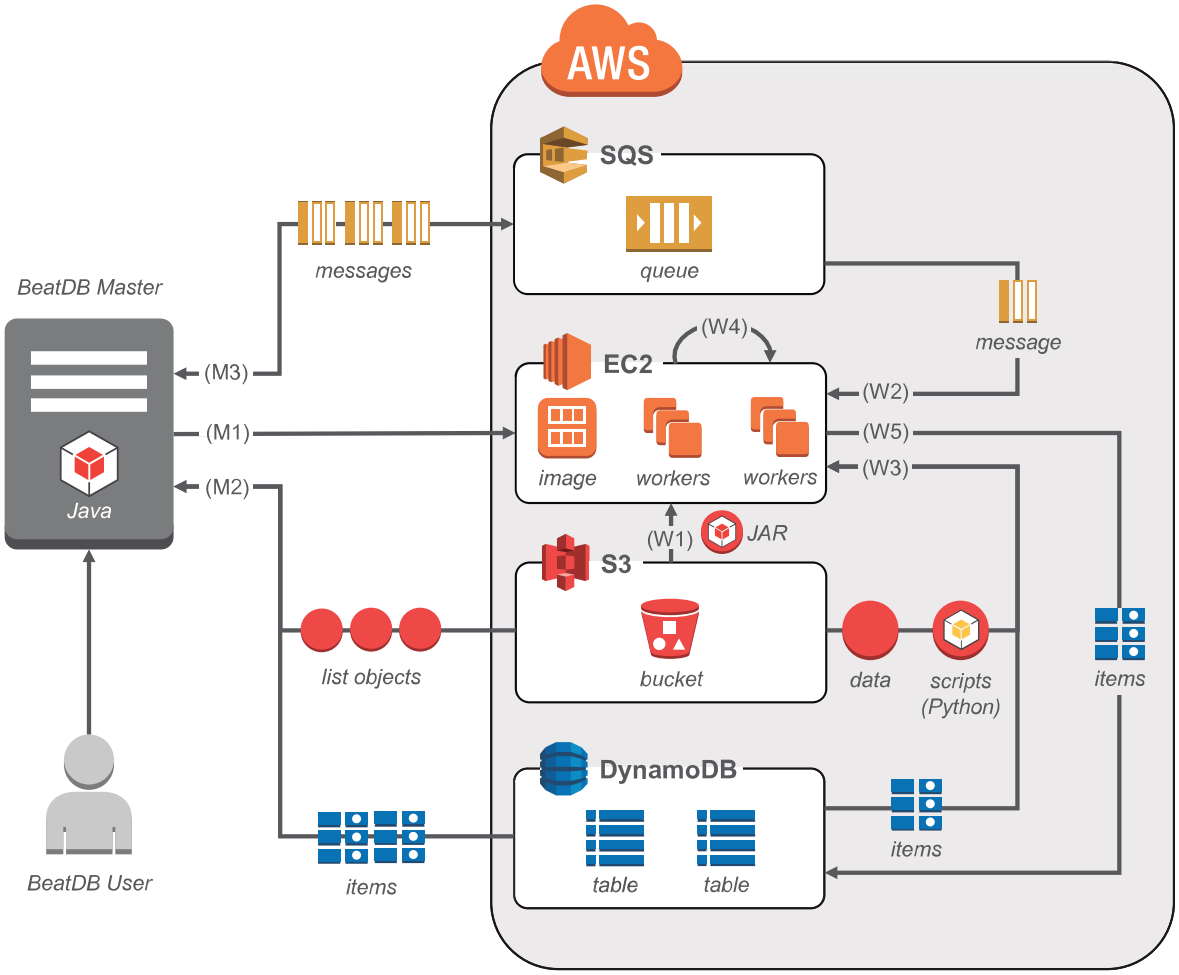


Figure D.2: *Cloud architecture of the BeatDB project.*

cut down on time needed for prediction studies and data processing without sacrificing any of the parameterization and specificity to the data possible with custom (and often single-use) scripts. In particular, the project combines four products of the AWS ecosystem: EC2 for computation, S3 for storage of raw files, DynamoDB for storage of processed features and SQS for orchestration. The global architecture of the BeatDB project is shown in figure D.2.

The project has been tested with the MIMIC-II dataset, using ECG (electrocardiography) and ABP (arterial blood pressure) for detecting acute hypotensive episodes (AHE) and hemodynamic instability (HDI). This project is supported by Philips Research North America.

D.2.2 Iron Deficiency Prediction

This project started as a collaboration with of a team medical researchers affiliated with the Harvard Medical School and the Center for Systems Biology of the Massachusetts General Hospital, under the supervision of John M. Higgins.

The purpose of this project is to apply machine learning techniques in order to predict iron deficiency anemia, given a corpus of patients whose complete blood count (CBC) were periodically

measured. In a first approach of the project, a basic analysis and data summary of the population was carried out, and several supervised learning techniques were applied to diagnose anemia at the moment of the CBC. Further work, which is currently ongoing, involves the prediction of anemia in the future based on historical CBC records.

D.3 Teaching

The FPU scholarship that establishes the current Ph.D. funding scheme requires candidates to actively participate in teaching activities, in order to improve their skills. In this period, the candidate has imparted lessons in both undergraduate studies and master studies. In this section, all the teaching activities in which the candidate was involved are described.

D.3.1 Teaching in Bachelor Degrees

All the teaching activities in bachelor degrees have been carried out in Universidad Carlos III de Madrid, which is the candidate's affiliation. The candidate has participated in the following courses:

Genetic and Evolutionary Algorithms This course is elective in the fourth year of the Bachelor Degree in Computer Science and Engineering, for students in the Computer Science specialization.

The candidate was in charge of ellaborating, imparting and evaluating the laboratory sessions. This course comprises two laboratory activities. In the first activity, students have to work individually to implement a genetic algorithm and apply it to a certain domain, which varies from year to year (e.g.: stock market, air quality measurement, N-queens problem, etc). In the second activity, students join groups of 4–5 people to work in a larger development for using different evolutionary computation techniques for an engineering or scientific problem which also changes yearly (e.g.: video games agents, symbolic regression, etc).

The candidate has imparted a total of **55 hours** in the following dates:

- October 7, 2015 – January 15, 2016
- September 12, 2016 – January 13, 2017
- September 18, 2017 – January 22, 2018

In the 2016 and 2017 editions of this course, the candidate received an extraordinary positive assessment in the satisfaction surveys to students because of his teaching activity.

Artificial Neural Networks This course is required for all students of the fourth year of the Bachelor Degree in Computer Science and Engineering in the Computer Science specialization.

The candidate was in charge of co-imparting the laboratory sessions, solving the student's questions. The practice involved the application of a multilayer perceptron to solve a classification problem, as well as evaluating the performance of Kohonen self-organizing maps and learning vector quantization. Additionally, the candidate offered a 2-hours seminar on an introduction to deep learning, showing how to develop a cat recognizer using TensorFlow and Keras.

The candidate has imparted **7 hours** in the following dates: November 10 – December 14, 2017

Artificial Intelligence This course is required for all students of the second year of the Bachelor Degree in Computer Science and Engineering.

The candidate was in charge of co-imparting the laboratory sessions, solving the students questions and evaluating the submissions. The practice involved an application of reinforcement learning to a simple Pacman scenario. Additionally, the candidate offered a 1-hour seminar on an introduction to deep learning.

The candidate has imparted **10 hours** in the following dates: April 21 – May 5, 2017

Foundations of Programming This course is elective in the third year of the Bachelor Degree in Library and Information.

The candidate was in charge of designing the laboratory sessions, solving the student's questions and evaluating the submissions. Because the course was offered in a semi-presential fashion, the candidate also recorded audiovisual materials for helping students with the practice work.

The candidate has imparted a total of **29 hours** in the following dates:

- September 12, 2015 – January 15, 2016
- September 12, 2016 – January 13, 2017

D.3.2 Teaching in Master Degrees

The candidate has participated as a collaborator professor in Universidad International de la Rioja, where he has imparted the 3 ECTS course "Engineering for Massive Data Processing" in the Master in Visual Analytics and Big Data. During this course, students have learned the Hadoop ecosystem, and particularly how to apply HDFS, MapReduce and Hadoop tools (such as Hive or Pig) to solve real problems involving big data.

The candidate has imparted a total of **285 hours** in the following dates:

- June 30 – July 31, 2014
- April 6 – May 15, 2015
- October 19 – November 27, 2015
- April 4 – May 13, 2016
- October 17 – November 25, 2016

D.3.3 Direction of Bachelor Theses

Between October 2014 and the date of deposit of this document, the candidate has also supervised bachelor theses in Universidad Carlos III de Madrid. A complete list of thesis can be found in tables D.1 and D.2, which contain the thesis directed and co-directed by the candidate respectively. In thesis directed by the candidate, he was the main advisor of the undergraduate student, whereas in thesis co-directed by him, he was a secondary advisor.

2015–16	*"Realidad Virtual Aplicada a la Rehabilitación Física"* **Author:** Carlos Andrés Aguado Fidalgo (Computer Sci. and Eng.) **Advisors:** A. Baldominos & Y. Sáez
2016–17	*"Estudio del Stack Tecnológico para el Desarrollo de un Framework Escalable para APIs REST"* **Author:** Giancarlo Alfredo Muñoz Reinoso (Computer Eng. for Management) **Advisor:** A. Baldominos
	"Aprendizaje Automático Escalable para la Predicción del Rendimiento de Campañas de Marketing" **Author:** Jorge Sánchez García (Computer Eng. for Management) **Advisor:** A. Baldominos
	"Diseño y Desarrollo de un Sistema Inteligente para la Gestión de Productos" **Author:** Felipe Sordo Ruiz (Computer Eng. for Management) **Advisor:** A. Baldominos
2017–18	*"Development of an OpenStack Cloud Deployment Tool"* **Author:** Sergio Pérez Fernández (Computer Sci. and Eng.) **Advisor:** A. Baldominos
	"EIRA Project: Advancing in the Management of Personal Healthcare" **Author:** VIrene Sanz Vizcaíno (Biomedical Eng.) **Advisors:** A. Baldominos & A. Casado-Rivas
	"Sistema de Cerradura Electrónica Activada con Llave Virtual" **Author:** Yago Pérez Sáiz (Telematics Eng.) **Advisor:** A. Baldominos
	"Proyecto Jupiter: Framework para el Uso de Modelos de Machine Learning" **Author:** Miguel Fonseca Martínez (Telecommunication Eng.) **Advisor:** A. Baldominos
	"Comparativa de Tecnologías de Deep Learning" **Author:** Alberto Lozano Benjumea (Computer Sci. and Eng.) **Advisors:** A. Baldominos & I. Navarro
	"Vida Artificial en Watership: El Gen Egoísta" **Author:** Adrián Borja Pimentel (Computer Science and Eng.) **Advisor:** A. Baldominos
	"An Intelligent Diagnosis System for Primary Care" **Author:** Carlos Martín Cabarcos (Biomedical Eng.) **Advisor:** A. Baldominos
	"Diseño y Desarrollo de Estrategia BI para Optimización de Recursos de Empresa Simulada" **Author:** Ignacio Orihuela Espiga (Communication Systems Eng.) **Advisor:** A. Baldominos

Table D.1: *Bachelor theses directed by the candidate in Universidad Carlos III de Madrid.*

D.3.4 Direction of Master Theses

The candidate has also directed Master theses during the first two editions of the Master in Visual Analytics and Big Data of Universidad Internacional de la Rioja, which took place in 2014 and 2015. The candidate accumulated a total of **156 hours** of teaching because of this concept. The whole list of master theses directed by the candidate is shown in table D.3.

2015–16	*"Nuevas Formas de Interacción Gráfica con Videojuegos: Utilización de Leap Motion y Oculus Rift"* **Author:** Jennifer García de la Calle (Computer Sci. and Eng.) **Advisors:** G. Recio & A. Baldominos
2015–16	*"Modelado 3D del Campus de Leganés para su Integración en un Prototipo de Videojuego FPS"* **Author:** Miguel Bohada Arranz (Computer Sci. and Eng.) **Advisors:** G. Recio & A. Baldominos
2016–17	*"Desarrollo Prueba Concepto Videojuego Trivial en HTML5"* **Author:** Violeta Martín Fraile (Audiovisual Systems Eng.) **Advisors:** Y. Sáez & A. Baldominos
2016–17	*"Sistema de Simulación Inmersiva con Oculus Rift y WiiFit"* **Author:** Ángel Lahera García (Computer Sci. and Eng.) **Advisors:** Y. Sáez & A. Baldominos
2016–17	*"Computación Evolutiva Aplicada a la Clasificación a partir de Monitores de Actividad Física"* **Author:** María del Carmen del Barrio Cerro (Computer Science and Eng.) **Advisors:** Y. Sáez & A. Baldominos

Table D.2: *Bachelor theses co-directed by the candidate in Universidad Carlos III de Madrid.*

2014	*"Extracción y Explotación de la Información Relevante del Curso Clínico Digital"* **Author:** Ángel Lavado Cuevas (Visual Analytics and Big Data)
2014	*"Metodología para Segmentación de Dispositivos Móviles en Función de su Uso"* **Author:** Carmen Martínez Olmo (Visual Analytics and Big Data)
2015	*"Implementando una Plataforma Big Data en un Sistema Bancario Tradicional"* **Author:** Aída Martínez Almarza (Visual Analytics and Big Data)
2015	*"Big Data y Trazabilidad en Centrales de Esterilización"* **Author:** Alberto Quirós Narváez (Visual Analytics and Big Data)
2015	*"Procesamiento y Modelado de Datos para un Sistema Business Intelligence con Tecnología Big Data"* **Author:** Miriam Fresno Arranz (Visual Analytics and Big Data)

Table D.3: *Master theses directed by the candidate in Universidad Internacional de la Rioja.*

D.3.5 Other Teaching Activities

In the year 2018, the candidate has been imparting tutorial sessions to two groups of international students for "Files and Databases", which is a required course in the second year of the Bachelor in Computer Science and Engineering. This activity is part of the Engineering Programme for International Students, which has been introduced by the International School of Universidad Carlos III de Madrid to enhance the learning experience of incoming students from universities in the United States of America. A total of **30 hours** were spent by the candidate between the two groups.

Also, the candidate has enrolled in the program of informative and scientific classes aimed at secondary schools, promoted by Universidad Carlos III de Madrid. In this program, the candidate has presented the class "Have you already met Artificial Intelligence?" to high-school students between 15 and 18 years old. The purpose of this class is to introduce artificial intelligence, always from a critical perspective, showing and ellaborating on some of the state-of-the-art applications. The candidate has presented this class in the following sessions:

- IES Gaspar Sanz on January 1, 2018 (40 students).
- IES Ana María Matute on January 19, 2018 (80 students).
- IES María Zambrano on January 23, 2018 (2 sessions, 80 students).
- IES La Serna on February 1, 2018 (50 students).
- IES La Serna on February 13, 2018 (25 students).
- IES Calderón de la Barca on March 14, 2018 (180 students).
- IES María Guerrero on March 15, 2018 (70 students).

Finally, the candidate has also participated in the following non-formal teaching:

- MBA for Digital Businesses in CES Cardenal Cisneros (2015).
- Master in Business Intelligence and Big Data in CES Cardenal Cisneros (2015).
- Master in Big Data for Managers in CES Cardenal Cisneros (2016).
- Course on Introduction to Big Data in IESDE School of Management in Mexico (2016).
- Expert in Data Science Practitioner in Universidad Internacional de la Rioja (2017).

D.4 Training

During the Ph.D. funding period, the candidate has completed courses in order to get specialized training and improve his knowledge and areas of expertise. This section describes these courses.

D.4.1 Online Courses

The candidate has completed several verified massive open online courses (MOOCs), which are enumerated in this section.

Deep Learning Specialization (Deeplearning.ai)

coursera.org/verify/specialization/PPBDV3TMSSM2

The Deep Learning Specialization is designed to prepare learners to participate in the development of cutting-edge AI technology, and to understand the capability, the challenges, and the consequences of the rise of deep learning. Through five interconnected courses, learners develop a profound knowledge of the hottest AI algorithms, mastering deep learning from its foundations (neural networks) to its industry applications (Computer Vision, Natural Language Processing, Speech Recognition, etc.)

Neural Networks and Deep Learning (Deeplearning.ai)

coursera.org/verify/C6Y4CS57K8K9

This course helps to break into cutting-edge AI. Deep learning engineers are highly sought after, and mastering deep learning will allow numerous new career opportunities. Deep learning is also a new "superpower" that will let you build AI systems that just weren't possible a few years ago. This course teaches the foundations of deep learning and enables to: (1) Understand the major technology trends driving Deep Learning, (2) Be able to build, train and apply fully connected deep neural networks, (3) Know how to implement efficient (vectorized) neural networks, and (4) Understand the key parameters in a neural network's architecture. This course also teaches how Deep Learning actually works, rather than presenting only a cursory or surface-level description.

Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization (Deeplearning.ai)

coursera.org/verify/VAAV8FCDMCXY

This course teaches the "magic" of getting deep learning to work well. Rather than the deep learning process being a black box, the course allows to understand what drives performance, and to be able to more systematically get good results. This course enables to: (1) Understand industry best-practices for building deep learning applications, (2) Be able to effectively use the common neural network "tricks", including initialization, L2 and dropout regularization, Batch normalization, gradient checking, (3) Be able to implement and apply a variety of optimization algorithms, such as mini-batch gradient descent, Momentum, RMSprop and Adam, and check for their convergence, (4) Understand new best-practices for the deep learning era of how to set up train/dev/test sets and analyze bias/variance, and (5) Be able to implement a neural network in TensorFlow.

Structuring Machine Learning Projects (Deeplearning.ai)

coursera.org/verify/G89UZCDWSUJD

This course teaches how to build a successful machine learning project. Much of this content has never been taught elsewhere, and is drawn from the professor's experience building and shipping many deep learning products. This course also has two "flight simulators" that allows practicing decision-making as a machine learning project leader. This provides "industry experience" that might otherwise only be obtained after years of ML work experience. This course enables to: (1) Understand how to diagnose errors in a machine learning system, (2) Be able to prioritize the most promising directions for reducing error, (3) Understand complex ML settings, such as mismatched training/test sets, and comparing to and/or surpassing human-level performance, and (4) Know how to apply end-to-end learning, transfer learning, and multi-task learning.

Convolutional Neural Networks (Deeplearning.ai)

coursera.org/verify/TQT5S53K2TEY

This course teaches how to build convolutional neural networks and apply it to image data. Thanks to deep learning, computer vision is working far better than just two years ago, and this is enabling numerous exciting applications ranging from safe autonomous driving, to accurate face recognition, to automatic reading of radiology images. This course enables to: (1) Understand how to build a convolutional neural network, including recent variations such as residual networks, (2) Know how to apply convolutional to visual detection and recognition tasks, (3) Know to use neural style transfer to generate art, and (4) Be able to apply these algorithms to a variety of image, video, and other 2D or 3D data.

Sequence Models (Deeplearning.ai)

coursera.org/verify/QR5ZYW3CBA5V

This course teaches how to build models for natural language, audio, and other sequence data. Thanks to deep learning, sequence algorithms are working far better than just two years ago, and this is enabling numerous exciting applications in speech recognition, music synthesis, chatbots, machine translation, natural language understanding, and many others. This courses enables to: (1) Understand how to build and train Recurrent Neural Networks (RNNs), and commonly-used variants such as GRUs and LSTMs, (2) Be able to apply sequence models to natural language problems, including text synthesis, and (3) Be able to apply sequence models to audio applications, including speech recognition and music synthesis.

Big Data XSeries (University of California, Berkeley)

verify.edx.org/cert/ac294210a94c4e1286bbc22e676ec61e

This specialization, created in partnership with Databricks, teaches how to perform data science and data engineering at scale using Spark, a cluster computing system well-suited for large-scale machine learning tasks. It also presents an integrated view of data processing by highlighting the various components of data analysis pipelines, including exploratory data analysis, feature extraction, supervised learning, and model evaluation. Students will gain hands-on experience building and debugging Spark applications. Internal details of Spark and distributed machine learning algorithms are covered, which provide an intuition about working with big data and developing code for a distributed environment. The specialization teaches: (1) How to use Spark and its libraries to solve big data problems, (2) How to approach large scale data science and engineering problems, (3) Spark's APIs, architecture, and many internal details, (4) The trade-offs between communication and computation in a distributed environment, and (5) Use cases for Spark.

Introduction to Big Data with Apache Spark (University of California, Berkeley)

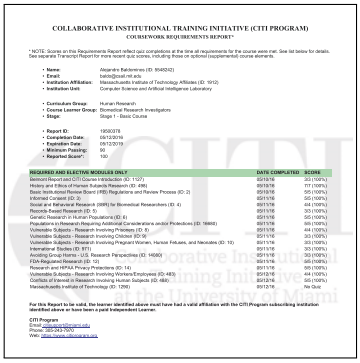
verify.edx.org/cert/4d7f1ea49988488394dd8be5f55c6ab8

Spark is rapidly becoming the compute engine of choice for big data. Spark programs are more concise and often run 10-100 times faster than Hadoop MapReduce jobs. As companies realize this, Spark developers are becoming increasingly valued. This course teaches the basics of working with Spark and provides with the necessary foundation for diving deeper into Spark. It covers Spark's architecture and programming model, including commonly used APIs. After completing this course, students will able to write and debug basic Spark applications. This course also explains how to use Spark's web user interface (UI), how to recognize common coding errors, and how to proactively prevent errors. The focus of this course is Spark Core and Spark SQL. This course teaches: (1) Basic Spark architecture, (2) Common operations, (3) How to avoid coding mistakes, and (4) How to debug Spark program.

Scalable Machine Learning (University of California, Berkeley)

verify.edx.org/cert/b87bbe5b46024d9b80ade6725e809e0b

Machine learning aims to extract knowledge from data, relying on fundamental concepts in computer science, statistics, probability and optimization. Learning algorithms enable a wide range of applications, from everyday tasks such as product recommendations and spam filtering to bleeding edge applications like self-driving cars and personalized medicine. In the age of "big data" with datasets rapidly growing in size and complexity and cloud computing becoming more pervasive, machine learning techniques are fast becoming a core component of large-scale data processing pipelines. This course introduces the underlying statistical and algorithmic principles required to develop scalable real-world machine learning pipelines. Students will gain hands-on experience applying these principles using Spark, a cluster computing system well-suited for large-scale machine learning tasks. This course teaches: (1) The underlying statistical and algorithmic principles required to develop scalable real-world machine learning pipelines, (2) Exploratory data analysis, feature extraction, supervised learning, and model evaluation, (3) Application of these principles using Spark, and (4) How to implement distributed algorithms for fundamental statistical models.


COLLABORATIVE INSTITUTIONAL TRAINING INITIATIVE (CITI PROGRAM)
COURSEWORK REQUIREMENTS REPORT*

Human Research for Biomedical Research Investigators (Miami CITI Program)

This course provides an introduction to the protection of human subjects in biomedical research. It offers historic and current information on regulatory and ethical issues important to the conduct of research involving human subjects. Case studies are used within the modules to present key concepts.

Federal regulations require that all personnel involved in any NIH sponsored research take and pass a training course on human subjects research before embarking on such research. MIT policy extends this requirement to all MIT personnel involved in any human subjects research. This requirement extends to all personnel who play a role in research involving human subjects including principal investigators, associate investigators, student investigators, study coordinators, visiting

scientists, consultants, laboratory technicians and assistants. The requirements encompasses all types of interactions with human subjects including, direct contact, indirect involvement, analysis of data and analysis of blood/tissue samples.

D.4.2 Research Skills Training Program

The Ph.D. Program in Computer Science and Technology at UC3M, regulated under the RD 99/2011 Rules and Regulations for Doctoral Studies in Spain, require the completion of 6 ECTS of research skills training during the doctoral training period.

In order to achieve the completion of this credits in research skills, the candidate has attended the following events or completed the following activities:

- Doctoral Workshop: "Winning Horizon2020 with Open Science" organized by Universidad Carlos III de Madrid.
- Third International Congress in Emerging Technologies and Society, organized by Universidad Internacional de la Rioja.
- Big Data Spain 2014.
- Big Data Spain 2015.
- Training Course for External Auditors in ANECA AUDIT Program, for the evaluation of Internal Systems of Quality Assurance.
- Organization of T3chFest 2017.

D.5 Other Merits

Between October 2014 and the date of deposit of this document, the candidate has received the following awards:

- National Excellence Award due to an outstanding academic performance during undergraduate studies, as published in BOE (*Boletín Oficial del Estado*) on November 21, 2015 [35].
- Excellence Award from UC3M Social Council in the Alumni category [368].
- Best Big Data Architecture Solution Special Mention in the Data Science Awards 2017.

D.6 Legal Background of Funding Scheme

The Ph.D. candidate has completed his dissertation under a funding scheme regulated by the FPU (*Formación del Profesorado Universitario*, University Faculty Teaching) scholarships of the Spanish Ministry of Education, Culture and Sports. The following legal documents describe the background and specifics of this scholarships programme:

- Documents announcing, modifying and resolving the FPU scholarships:
  - The resolution in which the 2013 FPU scholarships are announced is published in BOE (*Boletín Oficial del Estado*) on November 21, 2013 [32].
  - The resolution in which the 2013 FPU scholarships are resolved is published in BOE on September 4, 2014 [33]. As stated in Annex I of this document, the Ph.D. candidate was the recipient of one of these scholarships under grant number FPU13/03917.
  - A resolution modifying the 2013 resolution of FPU scholarships is published in BOE on December 9, 2015 [34].
  - A resolution modifying the 2013 resolution of FPU scholarships is published in BOE on February 23, 2018 [39].
- Documents resolving the grants for covering tuition costs for doctoral studies under the FPU scholarships framework:
  - The resolution in which the FPU grants for covering tuition costs for the course 2014–2015 is published in BOE on March 31, 2015 [36].
  - The resolution in which the FPU grants for covering tuition costs for the course 2015–2016 is published in BOE on December 4, 2015 [37].
  - The resolution in which the FPU grants for covering tuition costs for the course 2017–2018 is published by the Spanish Ministry of Education, Culture and Sports on December 19, 2017 [248].
- Documents announcing and resolving the travel grants for international short stays under the FPU scholarships framework:
  - The resolution in which the FPU research stays grants for the year 2016 are announced is published in BOE on August 5, 2015 [38].
  - The resolution in which the FPU research stays grants for the year 2016 are resolved is published by the Spanish Ministry of Education, Culture and Sports on March 4, 2016 [247]. As stated in Annex I of this document, the candidate was the recipient of one of these grants for a short stay in the United States of America with a evaluation score of 8.75 under identifier EST15/00260.

D.7 List of Documents for Accountability

In order to comply with the obligations and accountability imposed by the formerly described legal framework, the candidate has delivered the following documentation:

- 2015 Yearly follow-up delivered to Universidad Carlos III de Madrid in June 2015, comprising the initial research plan and the advisors' assessment.
- 2015 Yearly follow-up delivered to the Spanish Ministry of Education, Culture and Sports in September 2015, comprising the candidate's resume, the doctoral report for the first year, the teaching plan for the upcoming year, the advisors' assessment and the Academic Committee's assessment.
- International short stay plan delivered to the Spanish Ministry of Education, Culture and Sports in October 2015, comprising the initial research plan for the stay, the authorization of the adscription centre and the acceptance in the host institution.

- Academic visit authorization delivered to Universidad Carlos III de Madrid in February 2016, comprising the research plan, and the authorization from the Head of the Computer Science Department and the Human Resources Department.
- 2016 Yearly follow-up delivered to Universidad Carlos III de Madrid in June 2016, comprising the evaluation of completion of the research plan and the advisors' assessment.
- International short stay report delivered to the Spanish Ministry of Education, Culture and Sports in July 2016, comprising the summary report of research activities during the stay and the certificate of attendance signed by the host institution.
- Academic visit report for the "International Doctor" distinction, delivered to Universidad Carlos III de Madrid in September 2016, comprising the summary report of research activities and the certificate of attendance signed by the host institution.
- 2016 Yearly follow-up delivered to the Spanish Ministry of Education, Culture and Sports in September 2016, comprising the candidate's resume, the yearly doctoral report, the summary of teaching tasks completed by the candidate, the teaching plan for the upcoming year, the advisors' assessment and the Academic Committee's assessment.
- 2017 Yearly follow-up delivered to Universidad Carlos III de Madrid in June 2017, comprising the evaluation of completion of the research plan and the advisors' assessment.
- Request for a 1-year extension delivered to Universidad Carlos III de Madrid in July 2017.
- 2017 Yearly follow-up delivered to the Spanish Ministry of Education, Culture and Sports in September 2016, comprising the candidate's resume, the yearly doctoral report, the summary of teaching tasks completed by the candidate, the teaching plan for the upcoming year, the advisors' assessment and the Academic Committee's assessment.
- 2018 Yearly follow-up delivered to Universidad Carlos III de Madrid in May 2017, comprising the evaluation of completion of the research plan and the advisors' assessment.

All the previous documents have been validated and approved by the entity in charge, being this a requirement to complete the doctoral studies. The previous list is not exhaustive inasmuch as it does not include documentation delivered before the enrollment in the doctoral studies in 2014 (e.g., the research plan report delivered for requesting the FPU scholarship) or after the deposit of the current document (e.g., completion reports).

Appendix E

Beyond the Machines

As someone who was born and grew in the 1990s, I have always made myself many questions about the life of earlier generations of people. To be fair, these questions are not always as profound as they probably should be. In fact, a question that always intrigued me was how our parents lived and felt the moment in which Darth Vader reveals to Luke Skywalker about their relationship.

I remember the moment when I watched The Empire Strikes back for the first time. It was not in a theatre, of course, since the movie had been released 10 years before I was born. Instead, my father put the old videotape to play at home. I had never watched it before and, somehow, I knew what was about to happen. I was only waiting for the moment of it happening: the legendary "Luke, I am your father"[1] quote. For some reason that quote had already *viralized*[2], even before the era of Internet. Somehow, it was part of the pop culture, and we were familiar with that quote before watching the film. We were only kids, but the society had already spoiled us the movie.

Our spoilers to future generations; however, may go far beyond these insignificancies.

It is true that during most of my adolescence I have had Internet, a mobile phone, MP3s and DVDs. But I still could enjoy part of the previous generation. I have travelled in a car without air conditioner. I remember the birthday in which I was given a Discman for the first time. Even more, I remember inserting audiotapes in the car radio. I have made calls using rotary dial telephones and stored data in floppy disks, even the old 5.25" ones[3]. It is mostly a matter of few years that kids will look at one of these floppy disks and will just see a 3D-printed version of the save icon[4].

Technology is advancing at a fast pace. And so does so-called "artificial intelligence" (AI). In chapter 2, we already discussed the origins of the term, and we agreed on that this thesis would remain agnostic regarding what is to be considered an intelligent behavior. However, I personally think that our consideration of what is an intelligent feature in a computer is correlated with the ubiquity of these features. While working at Xerox PARC, Mark Weiser [385] once said:

[1] However, this quote is indeed a misquotation, probably the most widespread in the history of film along with "Play it again, Sam" from Casablanca. Interestingly, these misquotations are still present even in different dubs, and these two examples of misquotations are also common in Spanish. Some communities try to explain these particular misconceptions as a result of a Mandela effect; resisting to accept the fact that they were unable to remember the original movie script and were cheated by the popular culture.

[2] Or *still viralized*, given that this happened more than one decade after the movie was released.

[3] To be fair, this was mostly for fun.

[4] According to The Verge [99], so far this has only been a running gag.

> *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."*

Two decades ago, IBM ViaVoice was born. It was a software that performed speech recognition, translating the user voice into text. This form of interaction was; however, a little awkward: First, the user had to train a model by reading aloud certain book fragments, and secondly, the system often failed at recognizing some words; and still, it was able to learn from errors. Today, we can communicate with personal assistants such as Alexa, Siri or Cortana without any previous training and with significant accuracy. What is more, YouTube is able to provide automatic captioning to videos involving not only speech recognition but also machine translation. Advancements in AI in the last decade have been significant, and while most of these advances amaze me, they are starting to become ubiquitous to newer generations.

I think that our vision of intelligence is highly anthropocentric. That may be the reason why many science-fiction novels and movies have typically depicted robots as humanoids, even when different designs could result in a simpler and more useful interaction with the environment [281]. I already suggested in chapter 2 how I think this anthropocentric perspective impacted on the definition of AI provided by the Turing test. It is quite likely that early users of computers thought that they were "smart" because of their amazing capability of performing fast calculations. Today, this feature is given for granted, and we hardly consider a pocket calculator "smart" in any way[5].

Therefore, the requirements for what is to be considered an intelligent behavior seems to reside in the eyes of those who look, and our consideration will likely change over time as new, more capable devices and algorithms are devised and become ubiquitous. What seems to be certain; however, is the fact that artificial intelligence will impact the way we interact with the world.

Six years ago, as a part of my bachelor thesis [13], I wrote [6]:

> *"Research in artificial intelligence (AI) often generates expectation, as it reveals new discoveries and inventions which show us realities which we were not able to imagine a few years ago. However, research in these areas is not controversy-free. On one side, advancements in AI systems help to build a more modern and comfortable society. On the other, these intelligent systems are perceived as machines which will eventually replace humans at work. Regardless of whether these fears are unfounded or not, it is true that interest in AI is accompanied by some distrust."*

In fact, the issue of how AI will impact our future life has been covered extensively in latest years, and top-tier journal Nature recently included a monograph on "the future of work" within the October 19 issue of 2017. Within this monograph, some authors raise and address concerns regarding the possibility that machine learning and robots could replace human workers in fields that for many years have seemed impossible to automate. This dystopia, by the way, is not necessarily something negative: "*if automated systems start making routine medical diagnoses, it could free doctors to spend more time interacting with patients and working on complex cases*", claims Emily Anthes [7].

But I find the comment from Yuval Noah Harari [140] remarkably interesting: he discusses some applications of artificial intelligence, such as self-driving vehicles, where he suggests that such vehicles would reduce accident rates, and derives an interesting conclusion:

[5]However, for some reason, we are starting to add the prefix "smart–" to many consumer products just for the sake of advertising. Not only does this happen with *smartphones* or *smartwatches* (which may not include any intelligent behavior at all), but I am highly amused by how OralB define their toothbrush travel case as "smart" just because it features a USB port.

[6]From the more than 150 pages that conform my B.Sc. thesis, I find this paragraph to be the most up-to-date statement. And to some extent, that holds both a comforting and a creepy side.

> *"Self-driving vehicles illustrate two important points. First, that in some fields it might make sense to replace all humans with robots and computers, even if individually some humans do a better job. Second, that when change comes to some realms, it might do so suddenly, not step-wise."*

Yet, he agrees on that new socioeconomic models should be developed, even mentioning the universal basic income, since it would "*cushion the poor against job loss and economic dislocation, and protect the rich from populist rage*". And he concludes:

> *"In the nineteenth century, the Industrial Revolution created new conditions and problems that none of the existing social, economic and political models could cope with. Consequently, humankind had to develop completely new models — liberal democracies, communist dictatorships and fascist regimes. It took more than a century of terrible wars and revolutions to experiment with these, separate the wheat from the chaff and implement the best solutions.*
>
> *The challenges posed in the twenty-first century by the merger of infotech and biotech are arguably bigger than those thrown up by steam engines, railways, electricity and fossil fuels. Given the immense destructive power of our modern civilization, we cannot afford more failed models, world wars and bloody revolutions. We have to do better this time."*

While the future remains unknown, we must understand that at some point in time we will be sharing the world with computer programs that are able to think. This co-living setting can materialize in a variety of ways, but it could happen that in some scenarios, humans and robots be indistinguishable, therefore the latter adhering to the definition of intelligence described in the Turing test. We have seen some examples of this recently in the last Google I/O conference in May 2018, when Google Duplex was presented, an automated voice assistance that resembled a human when calling on the phone to handle reservations. After its presentation, the issue of whether the machine should be forced to identify as such has been widely debated [143].

Not even a month later, in early June 2018, Google's CEO Sundar Pichai have announced some ethical principles that would rule Google research and development in AI [283]:

> *"We recognize that such powerful technology [artificial intelligence] raises equally powerful questions about its use. How AI is developed and used will have a significant impact on society for many years to come."*

A broader ethical framework for artificial intelligence has been proposed under the name of "friendly AI", which was first presented by Yudkowsky in 2001 [405] and refined throughout the years. According to this framework, if AI agents are developed who have their own goals, then these goals should be human-friendly, never going against the human nature.

We might be spoiling life to future generations, or maybe we are just simplifying it. What seems clear is that they might not be required to deal with most of the problems that we have found in our lives. But at the same time, we should take care of people living today, which might not adapt fast to the novelties of technology. As Harari stated, what we do, we must do it the right way.

This page has been intentionally left blank.

Glossary

activation function

function that is applied over the output of one neuron, before passing it as input to neurons in the following layer. In many cases, this function is a non-linearity, although technically any differentiable function can be used. The result of computing this function is called the *activation value* 4, 9, 18–21, 24–27, 29, 30, 35, 36, 38–43, 52, 54–56, 67, 68, 70, 72, 74, 76, 79, 85, 86, 108, 112, 116, 137, 140, 144, 147, 153, 170

architecture

see *topology* 4, 5, 9, 14, 20, 28–30, 34, 41, 52–55, 57, 58, 60–62, 65–70, 72, 74–78, 81–86, 88–90, 103, 104, 109–113, 116, 117, 120, 125, 129, 137, 141, 143, 148, 152–155, 157, 165, 168–171, 174, 179, 188, 205

artificial intelligence

area of computer science that aims at developing software or hardware that displays certain intelligent behavior, understood as those often present in human intelligence 1–3, 7–11, 13–15, 20, 28, 47, 56, 78, 83, 86, 159, 168, 182, 184, 191, 203–205, 211–213

artificial neural network

software or hardware implementation of an artificial intelligence program through an structure that tries to resemble the human brain, although in a simplified form and significantly smaller scale. A neural network is composed by a set of units, called neurons, which are connected among them through links each with a certain associated weight. Each neuron will receive several input values and produce one output value, as a result of applying certain activation function over the input 2–5, 7, 9, 11–19, 22, 23, 25, 28–31, 38–41, 43, 44, 46–48, 52–70, 75, 77, 79, 84, 87, 90, 102–105, 124, 139, 142, 153–155, 165, 167–169, 171, 173, 189, 204, 205

attribute

see *feature* 12, 137

backpropagation

process that takes place during the training of a neural network, using gradient descent to update the weights so that a certain loss function is reduced 4, 14, 15, 18, 27, 29, 30, 34, 35, 39, 41–43, 46, 48, 52, 56–58, 60–62, 68, 71, 73, 77, 80, 86, 153, 168

batch

subset of the training set that is introduced at once to the gradient descent optimizer for one backpropagation step 43, 44, 54, 68, 70, 75, 76, 79, 81, 85, 86, 90, 107, 139, 140, 142, 148, 205

bias (parameter)

special kind of weight that is introduced to neurons but is not connected to any other neuron (or otherwise seen, it is connected to a neuron with a fixed unitary activation value) 4, 18, 19, 21–24, 26, 27, 29, 30, 34–36, 43, 44

chromosome

see *genotype* 50, 52, 59, 68, 72, 75, 86, 91, 92, 94–96, 107–109, 139, 140

committee

group of machine learning models whose outputs are combined following a certain policy to generate a single output, for example, by computing the average value or the class returned by the majority of models 4–7, 54, 65, 70, 72, 75, 79, 83, 89–91, 97, 103, 105, 107, 113–115, 119, 120, 124, 131, 132, 142, 145, 150, 152, 154–156, 170, 171, 188

convolutional neural network

type of neural network frequently found in deep learning applications that includes layers which are in charge of using convolution for automatically extracting relevant features from raw data. In this document, we also use the term "convolutional layers" to refer to the specific layers of the network that are in charge of performing representation learning, and "convolutional neural network" for the whole network, including the feed-forward or recurrent layers it might have 2–9, 14, 15, 28, 29, 31–36, 38, 39, 41, 43, 47, 48, 52–56, 67–92, 97, 99–110, 112, 113, 116, 119–121, 123, 124, 127, 128, 131, 132, 137, 138, 140–142, 144, 145, 147, 148, 150–157, 163–165, 167, 169–175, 179, 181, 182, 184, 188, 206

cost function

average of the loss function over all the instances in a batch, used to compute the derivatives for the gradient descent process 22–24, 27, 41–44

cross entropy

loss function commonly used in multi-class classification problems, to compute the error of the softmax output 22, 27, 40, 137

deep learning

subfield within machine learning that aims at automatically learning relevant features from data in order to obtain complex functions that map some input to an output. In other words, deep learning techniques allow avoiding a manual feature engineering stage by automatizing this process. The term "deep" refers to the number of levels of composition of non-linear operations in the function learned, which is often multiple in this kind of techniques 1, 2, 4, 9, 11, 15, 28, 29, 31, 34, 41, 43, 46–48, 52, 55, 67, 68, 72, 78, 81, 101, 121, 137, 153, 157, 159, 163–165, 167, 168, 171–177, 179–185, 188, 189, 200, 201, 204–206

deep neural network

specific implementation of a deep learning technique using neural networks. In some cases, a deep neural network can be a convolutional neural network, although according to some definitions, a deep neural network can be a neural network with many hidden layers, even when convolution is not performed 2, 9, 29, 31, 32, 39, 41, 47, 48, 54, 68, 71, 72, 74, 77, 78, 101, 106, 138, 139, 167, 169–176, 179, 181–184, 189, 205

dense (layer)

in many deep learning frameworks, this term is used to refer to feed-forward fully connected layers 2, 9, 36, 38, 74, 76, 85, 90, 105, 106, 108–110, 112, 116, 137, 140–142, 144, 153, 157, 170

dropout

regularization technique, widely used to avoid overfitting in deep neural networks, where some neurons are randomly disabled in each epoch 43, 68, 72, 74, 86, 103, 105, 106, 108, 112, 116, 140, 144, 147, 163, 205

EMNIST

extended version of the MNIST database, containing different handwritten digits as well as letters, yet sharing the same format and structure than MNIST. It was released in 2017 to provide a more challenging benchmark than MNIST, which is esentially solved using convolutional neural networks 99, 120, 121, 123–133, 149, 152, 154, 156

ensemble

see *committee* 3, 4, 6, 13, 14, 54, 56, 65, 74, 79, 83, 84, 97, 105, 106, 110, 113–115, 119, 120, 127–132, 137, 138, 145, 146, 150–152, 154–157, 170

epoch

step of the gradient descent optimizer where all the neural network weights are updated after one pass over the whole training set 43, 75, 76, 88–90, 109, 110, 112, 113, 118, 125, 142, 145, 148, 149, 151, 163

evolutionary algorithm

particular implementation of a evolutionary computation technique. Most common examples of evolutionary algorithms are genetic algorithms, evolutionary strategies, genetic programming or evolutionary programming 48–53, 61, 62, 67, 69, 74, 76, 79, 84, 85, 87–89, 92, 94, 113, 137, 151–153, 155, 157, 173, 188, 191

evolutionary computation

set of metaheuristic optimization techniques that are biologically inspired, aiming at applying concepts from Darwin's theory of natural selection and evolution in order to improve the fitness of a population of individuals, which are candidate solutions to a problem 3, 5–7, 9, 48–50, 52, 56–58, 60, 61, 65, 74, 79, 80, 84–87, 89, 91, 97, 99, 110, 125, 151, 153–155, 167, 169–172, 174, 179, 193, 200

feature

value (either numeric or categorical) extracted from data that, together with other features, can be used for training a machine learning model 2, 4, 7, 9, 12, 18, 21, 25, 28, 30–34, 36, 43, 52, 68, 81, 82, 103–106, 112, 116, 139, 144, 153, 170, 189, 192–194, 198, 199, 206, 207

feature map

dimensional set of features extracted automatically by a convolutional layer, either from raw data or from lower level features from a previous layer 32–36, 77, 78

feed-forward

in a neural network, sequence of layers where values are propagated to the following layers, not displaying a recurrent behavior 4, 32, 35, 36, 48, 53, 56, 57, 59, 60, 63, 66, 67, 71, 72, 74, 75, 77, 79, 80, 86, 108, 116, 140, 153, 169, 170

filter

see *kernel* 33, 68, 71–76, 78, 80, 85, 104, 153, 163, 170

fitness

quality metric of an individual to optimize in an evolutionary algorithm 5, 7, 8, 49, 50, 52, 55–57, 60, 63–65, 72, 74–76, 86, 88–92, 94, 95, 97, 109–113, 116, 117, 141–144, 147, 148, 151, 153–157, 167, 174, 192

fully connected

sequence of layers of an artificial neural networks where all neurons from one layer are connected to all the neurons of the following layer 25, 32, 34–36, 38, 41, 43, 53, 57, 68–70, 74–77, 79, 86, 104, 112, 124, 137, 147, 163, 205

gated recurrent unit

implementation of a recurrent neuron that implements a cell state and update and output gates, helping to reduce the learning problems that frequently occur in classical recurrent neural networks. It is similar to LSTM, yet simpler 37, 86, 108, 140, 144, 147, 206

generation

step of the evolutionary algorithm, where the evolutionary operators are applied over the current population to generate a new one 49, 50, 63, 73–75, 86, 87, 89, 90, 92, 93, 95, 96, 110, 116, 142, 147

genetic algorithm

evolutionary computation technique in which individuals are encoded into a genotype (most likely a binary string) and genetic operators are applied over a population of genotypes, such as selection, recombination or mutation, expecting the quality of individuals to improve as generations happen 3, 9, 49, 50, 52, 55, 57–60, 62, 65, 68, 70, 71, 73, 75, 80, 89, 91–97, 101, 107–116, 118–120, 125–128, 130, 133, 139, 141–148, 150, 151, 153, 157, 188, 192, 194, 200

genotype

computational representation of a candidate solution in an evolutionary algorithm, according to a certain encoding, over which the evolutionary operators are applied 50–52, 57–59, 62–64, 86, 87, 91, 94, 95, 107, 139

gradient

vector of derivative values for a function of several variables at a certain point, thus pointing in the direction of the greatest rate of increase (or decrease) of the function at that point. This vector can be used by a gradient descent algorithm in order to find the minimum of a function 22, 29, 30, 35, 36, 39, 41–46, 78, 104

gradient descent

iterative optimization algorithm for finding the minimum of a function by taking steps proportional to the negative of the gradient at the current point 4, 22–24, 42–45, 76, 77, 104, 108, 140, 205

grammatical evolution

evolutionary computation technique in which individuals are encoded as a string belonging to a language that can be generated by a specified grammar, so that the algorithm will apply genetic operators such as selection, recombination or mutation, evolving these strings and expecting the quality of individuals to improve as generations happen 3, 9, 50–52, 64, 80, 89, 91, 94–97, 101, 107–110, 116–120, 125, 129–133, 139, 141, 142, 147–151, 153, 154, 188

graphical processing unit

hardware specifically designed for performing graphics computing, in recent years are also becoming widely used for massively parallelizing scientific applications, including deep learning neural networks 7, 8, 15, 43, 46–48, 52, 74, 76, 77, 99–101, 148, 153, 156, 157, 159–164, 167, 171–175, 177–179, 182–185

hidden layer

layer of a neural network that is neither the input nor the output layer 2, 4, 25, 29, 30, 53, 54, 56, 57, 67, 84, 103, 153, 169

hyperparameters

settings that control the execution of a machine learning algorithm which must be defined before the algorithm starts, and whose tuning can affect the performance of such algorithm. Examples are the topology or optimizer in a neural network, or the number of generations and population size in an evolutionary algorithm 4, 9, 19, 23, 32, 33, 42, 43, 45, 46, 53, 54, 56, 57, 59, 61, 62, 67–82, 85–90, 93, 94, 96, 104, 107–109, 114, 119, 139–141, 168–170, 195

instance

one item of a data set, where the set features are assigned (instantiated with) values 12, 13, 18, 20–27, 30, 31, 35, 39–43, 82, 99, 102, 121, 122, 129, 133, 135–138, 147

kernel

in a convolutional layer, a kernel is a multidimensional grid, whose cells contain real numbers (weights) which serve for convolving the input in order to generate one feature map 4, 33–35, 67, 72, 74–76, 81, 82, 85, 86, 88, 104, 108, 109, 116, 140, 144, 147, 153, 170

learning rate

rate at which the weights are updated using gradient descent. Larger values will update weights faster, but can overshoot the optimal value and diverge; whereas smaller values can take more time to reach the optimal value 23, 44–46, 57, 59–61, 68, 70, 72, 74, 76, 79, 81, 85, 86, 90, 108, 109, 112, 116, 140, 141, 145, 148, 170

learning rule

see *optimizer* 43, 46, 58, 59, 62, 65, 66, 79, 84, 86, 141, 145, 170

long short-term memory

implementation of a recurrent neuron that implements a cell state and input, forget and output gates, helping to reduce the learning problems that frequently occur in classical recurrent neural networks 36, 37, 55, 72, 76, 79, 86, 104, 108, 137, 138, 140, 144, 147, 206, 218

loss function

function that computes the classification error of an instance when compared to the real value 9, 12, 18, 22–24, 27, 40, 41, 44, 56, 103, 137, 153

machine learning

field of artificial intelligence that aims at developing software or hardwar that is able to automatically learn from some source: data, experience, etc. The concept of *learning* in this definition is very broad, and can encompass things such as inferring rules, extracting patters, learning some mapping function, etc. 1–3, 7, 9, 11–15, 28, 52, 77, 78, 82, 101, 102, 107, 124, 136, 137, 153, 168, 172, 175, 180, 182, 188, 189, 191–193, 195, 196, 198, 199, 205–207

metaheuristic

optimization technique that is designed to work efficiently over large search spaces, without using specific knowledge about the domain, but rather using some quality metric of candidate solutions 2, 3, 5, 49, 52, 84, 85, 97, 155, 156

MNIST

database of handwritten digits released by NIST (National Institute of Standards and Technology of the US) and widely used as a benchmark for evaluating machine learning algorithms. Convolutional neural networks have basically solved the test set, attaining error rates comparable to those of a human expert 14, 68–70, 74–77, 99, 101–122, 125, 126, 131, 141, 142, 144, 149, 151, 152, 154, 156, 163, 182, 188

multilayer perceptron

type of feedforward artificial neural network with at least one hidden layer 4, 25, 27, 28, 30, 34, 38, 53, 54, 103, 124, 169, 200

neuroevolution

research field born in the late 1980s which applies evolutionary computation techniques in order to optimize certain aspects of neural networks: weights, topologies, learning rules, etc 53, 56–58, 62–68, 74, 76, 78, 79, 84, 85, 87, 88, 91, 97, 107, 153, 157, 163, 167, 169, 171, 174

neuron

in artificial neural networks, each processing unit, that takes several values from neurons in the previous layer and process them to generate a single activation value. This concept is inspired by biological neurons, which are also interconnected and their firing (activation) is based on their input 4, 9, 13, 14, 16–21, 25, 26, 30, 36, 38, 40, 42, 43, 53, 54, 56, 59, 60, 63, 67, 70, 72, 74–76, 85, 86, 90, 108, 116, 140, 144, 153, 163, 169, 170

niching

technique, inspired by the biological concept of ecological niches, used for preserving diversity in an evolutionary algorithm 63, 76, 90, 93, 95, 97, 112, 114, 142, 151, 152, 154, 155

OPPORTUNITY

human activity data set where sensors are located "opportunistically", meaning that it is intended to resemble an environment where sensors can be found in many places without a very specific placement 99, 133–139, 141–152, 154, 156, 165, 188

optimizer

algorithm that applies gradient descent during the backpropagation phase to train the weights of the neural network in order to minimize the loss function 9, 30, 43, 68, 78, 80, 90, 108, 116, 140, 145, 148

parameter

see *weight* 4, 9, 18, 19, 21–24, 27, 29, 30, 34–36, 39, 41, 43, 44, 52, 55, 56, 59, 66, 67, 70, 76, 79, 80, 114

patch

see *kernel* 4, 33, 85, 144

phenotype

candidate solution in an evolutionary algorithm, after being decoded from the genotype 50–52, 62–64, 71, 86, 87, 91, 94, 95, 107–110, 116, 119, 139, 141, 142

pooling

in convolutional neural networks, pooling layers are in charge of performing downsampling of the input feature maps by computing one statistical value for a whole subset of each feature map. Most common statistical values are the maximum (in max-pooling) and the mean (in average-pooling) 4, 34–36, 67–71, 74, 75, 81, 85, 86, 90, 103–106, 108, 116, 140, 144, 147, 153, 163, 170

rectified linear unit

simple non-linear activation function very used in deep learning, computed as $y = max(0, x)$ 39, 40, 71, 85, 86, 105, 106, 108, 112, 116, 137, 140, 144, 147

recurrent neural network

type of artificial neural network where some neurons are connected to themselves or to previous neurons, creating loops during the forward propagation step. This kind of neurons have been commonly used for learning time series, although in recent years they are being replaced by LSTM or GRU implementations, which offer some advantages during the learning process. In this work, we use the term "recurrent layers" to refer to some layers with a recurrent component (even those with LSTM or GRU cells), which can be part of a classical or a convolutional neural network 2, 4, 7, 14, 36, 38, 47, 48, 52, 55–57, 59, 60, 63, 67, 68, 70–72, 74–77, 79, 80, 86, 104, 105, 108, 112, 116, 137, 140, 144, 147, 153, 168–170

regularization

technique for avoiding overfitting by adding a penalty proportional to the size of the weights, or by removing connections, like in the case of dropout 9, 41–43, 52, 86, 104, 106, 108, 112, 116, 140, 144, 147, 153, 205

sample

see *instance* 43, 85, 88, 101, 102, 115, 120–122, 124, 129, 133, 136, 139, 142, 148, 154, 155, 188

softmax

function commonly applied over the output values of a neural network in order to normalize them so that all the values sum up to one, so they can be construed as probability 40, 41, 69, 70, 103, 105, 137, 163

supervised learning

set of machine learning techniques that aim at learning some mapping function between an input and an output given some labelled historic experience. Supervised learning problems include classification and regression, where the output is a discrete label or a continuous label respectively 1–6, 11, 12, 14, 18, 21, 40, 53, 57, 64, 66, 155, 168, 200, 206, 207

tensor

multidimensional array, used in deep learning to represent data taking advantage of the underlying dimensional structure 9, 31–36, 46, 47, 71, 82, 138, 139, 157, 178

topology

structure of an artificial neural network, defined by its number of layers, number of neurons in each layer (or filters in convolutional layers), how these neurons are connected, etc 4–7, 25, 32, 33, 43, 52–70, 72–74, 76, 77, 79–90, 92, 97, 99, 101, 104, 112, 113, 116, 118–120, 124–126, 130, 133, 142, 144, 145, 147, 149, 151–157, 167–169, 171–174, 179, 184, 188

unit

see *neuron* 18, 37, 40, 42, 43, 53, 54, 56–58, 60, 67, 68, 80, 104–106, 109, 147

update rule

see *optimizer* 9, 27

weight

parameter of the neural network consisting of a real value assigned to convolutional kernels and connections between neurons that is learned by the optimization algorithm 4, 16, 18–27, 29, 30, 34–36, 41–44, 46, 48, 54, 56–65, 68–73, 75–77, 79–82, 84, 86, 105, 108, 109, 120, 140, 153, 169

This page has been intentionally left blank.

Acronyms

AI

artificial intelligence 1–3, 7–11, 13–15, 20, 28, 47, 56, 78, 83, 86, 159, 168, 182, 184, 188, 191, 203–205, 211–213

ANN

artificial neural network 2–5, 7, 9, 11–19, 22, 23, 25, 28–31, 38–41, 43, 44, 46–48, 52–70, 75, 77, 79, 84, 87, 90, 102–105, 124, 139, 142, 153–155, 165, 167–169, 171, 173, 189, 204, 205

CNN

convolutional neural network 2–9, 14, 15, 28, 29, 31–36, 38, 39, 41, 43, 47, 48, 52–56, 67–92, 97, 99–110, 112, 113, 116, 119–121, 123, 124, 127, 128, 131, 132, 137, 138, 140–142, 144, 145, 147, 148, 150–157, 163–165, 167, 169–175, 179, 181, 182, 184, 188, 206

DNN

deep neural network 2, 9, 29, 31, 32, 39, 41, 47, 48, 54, 68, 71, 72, 74, 77, 78, 101, 106, 138, 139, 167, 169–176, 179, 181–184, 189, 205

GA

genetic algorithm 3, 9, 49, 50, 52, 55, 57–60, 62, 65, 68, 70, 71, 73, 75, 80, 89, 91–97, 101, 107–116, 118–120, 125–128, 130, 133, 139, 141–148, 150, 151, 153, 157, 188, 192, 194, 200

GE

grammatical evolution 3, 9, 50–52, 64, 80, 89, 91, 94–97, 101, 107–110, 116–120, 125, 129–133, 139, 141, 142, 147–151, 153, 154, 188

GPU

graphical processing unit 7, 8, 15, 43, 46–48, 52, 74, 76, 77, 99–101, 148, 153, 156, 157, 159–164, 167, 171–175, 177–179, 182–185

GRU

gated recurrent unit 37, 86, 108, 140, 144, 147, 206

LSTM

long short-term memory 14, 36, 37, 55, 72, 76, 79, 86, 104, 108, 137, 138, 140, 144, 147, 206, 218

ML

machine learning 1–3, 7, 9, 11–15, 28, 52, 77, 78, 82, 101, 102, 107, 124, 136, 137, 153, 168, 172, 175, 180, 182, 188, 189, 191–193, 195, 196, 198, 199, 205–207

MLP
multilayer perceptron 4, 25, 27, 28, 30, 34, 38, 53, 54, 103, 124, 169, 200

ReLU
rectified linear unit 38–40, 71, 85, 86, 105, 106, 108, 112, 116, 137, 140, 144, 147

Bibliography

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation* (Savannah, GA, USA, 2016), pp. 265–283.

[2] Albelwi, S., and Mahmood, A. A framework for designing the architectures of deep convolutional neural network. *Entropy 19*, 6 (2017), 242.

[3] ALFA Group. GIGABEATS, 2018. http://groups.csail.mit.edu/EVO-DesignOpt/gigabeats/projects.html; last visited on 2018-05-18.

[4] Alom, M. Z., Hasan, M., Yakopcic, C., and Taha, T. M. Inception recurrent convolutional neural network for object recognition. *arXiv 1704.07709* (2017).

[5] Alonso, M. J. BigML Release: Automatically Find the Optimal Machine Learning Model with OptiML!, 2018. https://blog.bigml.com/2018/05/07/bigml-release-automatically-find-the-optimal-machine-learning-model-with-optiml/; published on 2018-05-07.

[6] Angeline, P. J., Saunders, G. M., and Pollack, J. B. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks 5*, 1 (1994), 54–65.

[7] Anthes, E. The shape of work to come. *Nature 550*, 7676 (2017), 316–319.

[8] Association for Computing Machinery. The 2012 ACM Computing Classification System, 2017. https://www.acm.org/publications/class-2012; last visited on 2017-06-14.

[9] Azzopardi, G., and Petkov, N. Trainable COSFIRE filters for keypoint detection and pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 2 (2013), 490–503.

[10] Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. In *Proceedings of the 5th International Conference on Learning Representations* (Poughkeepsie, NY, USA, 2017).

[11] Balakrishnan, K., and Honavar, V. Evolutionary design of neural architectures – a preliminary taxonomy and guide to literature. Tech. rep., Iowa State University, 1995. Paper 26.

[12] Baldi, P., and Sadowsky, P. J. Understanding dropout. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26 of *Advances in Neural Information Processing Systems*. Curran Associates, Lake Tahoe, NV, USA, 2013, pp. 2814–2822.

[13] Baldominos, A. Circumstantial knowledge management for human-like interaction, 2012. Bachelor thesis, Universidad Carlos III de Madrid.

[14] Baldominos, A. A Comparison between NVIDIA's GeForce GTX 1080 and Tesla P100 for Deep Learning – Is it worth the dollar?, 2017. https://medium.com/@alexbaldo/a-comparison-between-nvidias-geforce-gtx-1080-and-tesla-p100-for-deep-learning-81a918d5b2c7; published on 2017-10-05.

[15] Baldominos, A. Analyzing the Performance of Intel Xeon Phi for Deep Learning – Does this Intel's brand new processor live up to the expectations?, 2017. https://medium.com/@alexbaldo/analyzing-the-performance-of-intel-xeon-phi-for-deep-learning-89bb79ac3147; published on 2017-12-16.

[16] Balestrassi, P. P., Popova, E., Paiva, A. P., and Marangon-Lima, J. W. Design of experiments on neural network's training for nonlinear time series forecasting. *Neurocomputing 72*, 4–6 (2009), 1160–1178.

[17] Bannach, D., Amft, O., and Lukowicz, P. Rapid prototyping of activity recognition applications. *IEEE Pervasive Computing 7*, 2 (2008), 22–31.

[18] Baum, E. B., and Haussler, D. What size net gives valid generalization? *Neural Computation 1*, 1 (1989), 151–160.

[19] Bayes, T. An essay towards solving a problem in the Doctrine of Chance. *Philosophical Transactions 53* (1763), 370–418.

[20] BBC News. Google achieves AI 'breakthrough' by beating Go champion, 27 January 2016. http://www.bbc.com/news/technology-35420579.

[21] Belew, R. K., McInerney, K., and Schraudolph, N. N. Evolving networks: Using the genetic algorithm with connectionist learning. In *Artificial Life II*, C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, Eds., vol. 10 of *Proceedings of SFI Studies in the Sciences of Complexity*. Addison-Wesley, Redwood City, CA, USA, 1992, pp. 511–547.

[22] Bello, I., Zoph, B., Vasudevan, V., and Le, Q. V. Neural optimizer search with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning* (Sydney, Australia, 2017), vol. 70 of *JMLR Proceedings*, pp. 459–468.

[23] Belongie, S., Malik, J., and Puzicha, J. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 4 (2002), 509–522.

[24] Benenson, R. Classification datasets results. http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html; last visited on 2017-05-21.

[25] Bengio, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning 2*, 1 (2009), 1–127.

[26] Bengio, Y., Courville, A., and Vincent, P. Representation learning: a review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35* (2013), 1798–1828.

[27] Bergstra, J., and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research 13* (2012), 281–305.

[28] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. Theano: A CPU and GPU math compiler in Python. In *Proceedings of the 9th Python in Science Conference* (Austin, TX, USA, 2010).

[29] Bergstra, J., Yamins, D., and Cox, D. Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. *Journal of Machine Learning Research 28*, 1 (2013), 115–123.

[30] Blum, C., and Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys 35*, 3 (2003), 268–308.

[31] Bochinski, E., Senst, T., and Sikora, T. Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In *Proceedings of the 2017 IEEE International Conference on Image Processing* (Beijing, China, 2017), pp. 3924–3928.

[32] Boletín Oficial del Estado. Resolución de 18 de noviembre de 2013, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se convocan diversas ayudas para la formación de profesorado universitario de los subprogramas de Formación y de Movilidad dentro del Programa Estatal de Promoción del Talento y su Empleabilidad, en el marco del Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016 en I+D+i, 2013. BOE Reference: BOE-A-2013-12235, pp. 92894–92940; available at: http://www.boe.es/diario_boe/txt.php?id=BOE-A-2013-12235&lang=en; published on 2013-11-21.

[33] Boletín Oficial del Estado. Resolución de 22 de agosto de 2014, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se conceden ayudas para contratos predoctorales para la Formación de Profesorado Universitario, de los subprogramas de Formación y Movilidad dentro del Programa Estatal de Promoción del Talento y su Empleabilidad, 2014. BOE Reference: BOE-A-2014-9081, pp. 69498–69521; available at: http://www.boe.es/diario_boe/txt.php?id=BOE-A-2014-9081&lang=en; published on 2014-09-04.

[34] Boletín Oficial del Estado. Resolución de 12 de noviembre de 2015, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se modifica la de 18 de noviembre de 2013, por la que se convocan diversas ayudas para formacióó de profesorado universitario, de los subprogramas de Formación y de Movilidad dentro del Programa Estatal de Promoción del Talento y su Empleabilidad, en el marco del Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016 en I+D+i, 2015. BOE Reference: BOE-A-2015-13374, pp. 116387–116390; available at: http://www.boe.es/diario_boe/txt.php?id=BOE-A-2015-13374&lang=en; published on 2015-12-09.

[35] Boletín Oficial del Estado. Resolución de 17 de noviembre de 2015, de la Dirección General de Política Universitaria, por la que se adjudican los Premios Nacionales de Fin de Carrera de Educación Universitaria correspondientes al curso académico 2011-2012, 2015. BOE Reference: BOE-A-2015-12591, pp. 109958–109962; available at: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2015-12591&lang=en; published on 2015-11-21.

[36] Boletín Oficial del Estado. Resolución de 24 de marzo de 2015, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se conceden ayudas por matrícula en programas de doctorado a beneficiarios del subprograma de formación del profesorado universitario, correspondientes al curso 2014-2015, 2015. BOE Reference: BOE-A-2015-3484, pp. 27372–27415; available at: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2015-3484&lang=en; published on 2015-03-31.

[37] Boletín Oficial del Estado. Resolución de 26 de noviembre de 2015, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se conceden ayudas por matrícula en programas de doctorado a beneficiarios del subprograma de formación del profesorado universitario, correspondientes al curso 2015-2016, 2015. BOE Reference: BOE-A-2015-13184, pp. 115272–115329; available at: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2015-13184&lang=en; published on 2015-12-04.

[38] Boletín Oficial del Estado. Resolución de 20 de julio de 2015, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se convocan ayudas complementarias destinadas a beneficiarios del subprograma de formación del profesorado universitario y se establece el plazo para la presentación de las memorias de seguimiento de los beneficiarios que pasan al régimen de contrato en el año 2015, 2015. BOE Reference: BOE-A-2015-8831, pp. 70140–70158; available at: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2015-8831&lang=en; published on 2015-08-05.

[39] Boletín Oficial del Estado. Resolución de 18 de enero de 2018, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se modifica la de 18 de noviembre de 2013, por la que se convocan diversas ayudas para formacióó de profesorado universitario, de los subprogramas de Formación y de Movilidad dentro del Programa Estatal de Promoción del Talento y su Empleabilidad, en el marco del Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016 en I+D+i, 2018. BOE Reference: BOE-A-2018-2592, pp. 21721–21722; available at: http://www.boe.es/diario_boe/txt.php?id=BOE-A-2018-2592&lang=en; published on 2018-02-23.

[40] Bongard, J. Morphological change in machines accelerates the evolution of robust behavior. *Proceedings of the National Academy of Sciences USA 108*, 4 (2011), 1234–1239.

[41] Breiman, L. Bagging predictors. Tech. rep., University of California at Berkeley, 1994.

[42] Breiman, L. Bagging predictors. *Machine Learning 24*, 2 (1996), 123–140.

[43] Breiman, L. Random forests. *Machine Learning 45*, 1 (2001), 5–32.

[44] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[45] Brock, A., Lim, T., Ritchie, J., and Weston, N. SMASH: One-shot model architecture search through hypernetworks. *arXiv 1708.05344* (2017).

[46] Bruna, J., and Mallat, S. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 8 (2013), 1872–1886.

[47] Bryant, B. D., and Miikkulainen, R. Acquiring visibly intelligent behavior with example-guided neuroevolution. In *Proceedings of the 22nd National Conference on Artificial Intelligence* (Vancouver, British Columbia, Canada, 2007), pp. 801–808.

[48] Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. Efficient architecture search by network transformation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence* (New Orleans, LA, USA, 2018).

[49] Calderón, A., Roa-Valle, S., and Victorino, J. Handwritten digit recognition using convolutional neural networks and Gabor filters. In *Proceedings of the 2003 International Congress on Computational Intelligence* (Medellin, Colombia, 2003).

[50] Canziani, A., Paszke, A., and Culurciello, E. An analysis of deep neural network models for practical applications. *arXiv 1605.07678* (2017).

[51] Cao, H., Nguyen, M. N., Phua, C., Krishnaswamy, S., and Li, X.-L. An integrated framework for human activity classification. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (Pittsburgh, PA, USA, 2012), pp. 331–340.

[52] Caruana, R. Generalization vs. net size, 1993. Neural Information Processing Systems, Tutorial. Denver, CO.

[53] Cavalin, P. R., Britto, A. S., Bortolozzi, F., Sabourin, R., and Oliveira, L. E. S. An implicit segmentation-based method for recognition of handwritten strings of characters. In *Proceedings of the 2006 ACM Symposium on Applied Computing* (Dijon, France, 2006), pp. 836–840.

[54] Chalmers, D. J. The evolution of learning: An experiment in genetic connectionism. In *Connectionist Models – Proceedings of the 1990 Summer School*, D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, Eds. Morgan Kaufmann, San Diego, CA, USA, 1990, pp. 81–90.

[55] Chan, T. H., Jia, K., Gao, S., Lu, J., Zeng, Z., and Ma, Y. PCANet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing 24*, 12 (2015), 5017–5032.

[56] Chandra, R. Competition and collaboration in cooperative coevolution of elman recurrent neural networks for time-series prediction. *IEEE Transactions on Neural Networks and Learning Systems 26*, 12 (2015), 3123–3136.

[57] Chang, J. R., and Chen, Y. S. Batch-normalized maxout network in network. *arXiv 1511.02583* (2015).

[58] Chavarriaga, R., Sagha, H., Calatroni, A., Digumarti, S., Tröster, G., del R. Millán, J., and Roggen, D. The Opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recognition Letters 34*, 15 (2013), 2033—2042.

[59] Chellapilla, K., and Fogel, D. B. Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE 87*, 9 (1999), 1471–1496.

[60] Chen, T., Goodfellow, I., and Shlens, J. Net2Net: Accelerating learning via knowledge transfer. In *Proceedings of the 4th International Conference on Learning Representations* (San Juan, Puerto Rico, 2016).

[61] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv 1512.01274* (2015).

[62] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., and Shelhamer, E. cuDNN: Efficient primitives for deep learning. *arXiv 1410.0759* (2014).

[63] Chintala, S. FAIR open sources deep-learning modules for Torch, 2015. https://research.fb.com/fair-open-sources-deep-learning-modules-for-torch/; published on 2015-01-16.

[64] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv 1409.1259* (2014).

[65] Chomsky, N. Three models for the description of language. *IRE Transactions on Information Theory 2* (1956), 113–124.

[66] Chomsky, N., and Schützenberger, M. P. The algebraic theory of context-free languages. In *Computer Programming and Formal Systems*, P. Braffort and D. Hirschberg, Eds., vol. 35 of *Studies in Logic and the Foundations of Mathematics*. Norh Holland, 1963, pp. 118–161.

[67] Choudhary, A., Rishi, R., Dhaka, V. S., and Ahlawat, S. Influence of introducing an additional hidden layer on the character recognition capability of a BP neural network having one hidden layer. *International Journal of Engineering and Technology 2*, 1 (2010), 24–28.

[68] Cireşan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation 22*, 12 (2010), 3207–3220.

[69] Cireşan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. Convolutional neural network committees for handwritten character classification. In *Proceedings of the 2011 International Conference on Document Analysis and Recognition* (Beijing, China, 2011), pp. 1135–1139.

[70] Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence* (Barcelona, Spain, 2011), pp. 1237–1242.

[71] Cireşan, D. C., Meier, U., and Schmidhuber, J. Multi-column deep neural networks for image classification. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition* (Providence, RI, USA, 2012), pp. 3642–3649.

[72] Clune, J., Mouret, J.-B., and Lipson, H. The evolutionary origins of modularity. *Proceedings of the Royal Society of Biological Sciences 280*, 1755 (2013).

[73] Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. EMNIST: an extension of MNIST to handwritten letters. *arXiv 1702.05373* (2017).

[74] Collobert, R., Bengio, S., and Mariéthoz, J. Torch: a modular machine learning software library. Tech. rep., Dalle Molle Institute for Perceptual Artificial Intelligence, IDIAP Research Institute, October 2002.

[75] Collobert, R., Kavukcuoglu, K., and Farabet, C. Torch7: A Matlab-like environment for machine learning. In *Proceedings of Big Learning 2011: NIPS 2011 Workshop on Algorithms, Systems, and Tools for Learning at Scale* (Sierra Nevada, Spain, 2011).

[76] Cortes, C., and Vapnik, V. Support-vector networks. *Machine Learning 20*, 3 (1995), 273–297.

[77] Cramer, N. L. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications* (Pittsburgh, PA, USA, 1985), pp. 183–187.

[78] Cybenko, G. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems 2*, 4 (1989), 303–314.

[79] Darrow, B. Amazon has chosen this framework to guide deep learning strategy, 2016. http://fortune.com/2016/11/22/amazon-deep-learning-mxnet/; published on 2016-11-22.

[80] Darwin, C. R. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray London, 1859.

[81] Dasgupta, D., and McGregor, D. R. Designing application-specific neural networks using the structured genetic algorithm. In *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks* (Baltimore, MD, USA, 1992), pp. 87–96.

[82] David, E. O., and Grental, I. Genetic algorithms for evolving deep neural networks. In *Proceedings of the 2014 ACM Genetic and Evolutionary Computation Conference* (Vancouver, British Columbia, Canada, 2014), pp. 1451–1452.

[83] Davison, J. DEvol: Automated deep neural network design via genetic programming, 2017. https://github.com/joeddav/devol; last visited on 2017-07-01.

[84] de Garis, H. Steerable GenNETS: the genetic programming of steerable behavior in GenNETS. In *Towards a Practice of Autonomous Systems – Proceedings of the First European Conference on Artificial Life*, F. J. Varela and P. Bourgine, Eds. Bradford, Paris, France, 1992, pp. 272–281.

[85] Decoste, D., and Schölkopf, B. Training invariant support vector machines. *Machine Learning 46*, 1 (2002), 161–190.

[86] DeepMind. DeepMind. https://deepmind.com; last visited on 2017-03-21.

[87] Deng, L., and Yu, D. Deep convex net: A scalable architecture for speech pattern classification. In *Proceedings of the 12th Annual Conference of the International Speech Communication Association* (Florence, Italy, 2011), pp. 2285–2288.

[88] Desell, T. Large scale evolution of convolutional neural networks using volunteer computing. *arXiv 1703.05422* (2017).

[89] Desell, T., Clachar, S., Higgins, J., and Wild, B. Evolving deep recurrent neural networks using ant colony optimization. In *Evolutionary Computation in Combinatorial Optimization*, G. Ochoa and F. Chicano, Eds., vol. 9026 of *Lecture Notes in Computer Science*. Springer, Copenhagen, Denmark, 2015, pp. 86–98.

[90] Ding, S., Li, H., Su, C., Yu, J., and Jin, F. Evolutionary artificial neural networks: a review. *Artificial Intelligence Review 39*, 3 (2013), 251–260.

[91] Dong, J.-D., Cheng, A.-C., Juan, D.-C., Wei, W., and Sun, M. PPP-Net: Platform-aware progressive search for pareto-optimal neural architectures. In *Proceedings of the 6th International Conference on Learning Representations* (Vancouver, British Columbia, Canada, 2018).

[92] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research 12* (2011), 2121–2159.

[93] Easley, D., and Kleinberg, J. *Networks, Crowds and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

[94] Edlund, J. A., Chaumont, N., Hintze, A., Koch, C., Tononi, G., and Adami, C. Integrated information increases with fitness in the evolution of animats. *PLOS Computational Biology 7*, 10 (2011).

[95] Elias, J. G. Genetic generation of connection patterns for a dynamic artificial neural network. In *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks* (Baltimore, MD, USA, 1992), pp. 38–54.

[96] Elsken, T., Metzen, J.-H., and Hutter, F. Simple and efficient architecture search for convolutional neural networks. In *Proceedings of the 6th International Conference on Learning Representations* (Vancouver, British Columbia, Canada, 2018).

[97] European Space Agency. How many stars are there in the Universe?, 2017. http://www.esa.int/Our_Activities/Space_Science/Herschel/How_many_stars_are_there_in_the_Universe; last visited on 2017-11-10.

[98] Fahlman, S. E., and Lebiere, C. The cascade-correlation learning architecture. In *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds., vol. 2 of *Advances in Neural Information Processing Systems*. Morgan Kaufmann, Denver, CO, USA, 1990, pp. 524–532.

[99] Farokhmanesh, M. Why is this floppy disk joke still haunting the internet?, 2017. https://www.theverge.com/2017/10/24/16505912/floppy-disk-3d-print-save-joke-meme; published on 2017-10-24.

[100] Fernando, C., Banarse, D., Reynolds, M., Besse, F., Pfau, D., Jaderberg, M., Lanctot, M., and Wierstra, D. Convolution by evolution: Differentiable pattern producing networks. In *Proceedings of the 2016 ACM Genetic and Evolutionary Computation Conference* (Denver, CO, USA, 2016), pp. 109–116.

[101] Ferrero, G. L'inertie mentale et la loi du moindre effort. *Revue Philosophique de la France et de l'Étranger 37* (1894), 169–182.

[102] Flach, P. *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012.

[103] Floreano, D., Dürr, P., and Mattiussi, C. Neuroevolution: from architectures to learning. *Evolutionary Intelligence 1*, 1 (2008), 1–47.

[104] Foley, L. J., Owens, A. J., and Walsh, M. J. *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966.

[105] Forsyth, R. BEAGLE a Darwinian approach to pattern recognition. *Kybernetes 10*, 3 (1981), 159–166.

[106] Frean, M. The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation 2*, 2 (1990), 198–209.

[107] Freud, S. *Entwurf einer Psychologie*. [Manuscript], 1895.

[108] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics 36* (1980), 193–202.

[109] Funika, W., and Koperek, P. Towards co-evolution of fitness predictors and deep neural networks. *arXiv 1801.00119* (2017).

[110] Geurts, P., Ernst, D., and Wehenkel, L. Extremely randomized trees. *Machine Learning 63*, 1 (2006), 3–42.

[111] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feed forward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics* (Chia, Sardinia, Italy, 2010), vol. 9 of *JMLR Proceedings*, pp. 249—-256.

[112] Gluon. Deep learning – the straight dope, 2017. http://gluon.mxnet.io; last visited on 2017-12-08.

[113] Goldberg, D. E., and Richardson, J. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms* (Cambridge, MA, USA, 1987), pp. 148–154.

[114] Gomez, F., and Miikkulainen, R. Solving non-Markovian control tasks with neuroevolution. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (Stockholm, Sweden, 1999), pp. 1356–1361.

[115] Gomez, F., and Miikkulainen, R. Learning robust nonlinear control with neuroevolution. Tech. rep., Department of Computer Sciences, The University of Texas at Austin, 2002.

[116] Gomez, F., Schmidhuber, J., and Miikkulainen, R. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research 9* (2008), 937–965.

[117] Gomez, F. J., and Miikkulainen, R. Active guidance for a finless rocket using neuroevolution. In *Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation – Part II* (Chicago, IL, USA, 2003), pp. 2084–2095.

[118] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016.

[119] Goodfellow, I. J., Mirza, M., Courville, A., and Bengio, Y. Multi-prediction deep Boltzmann machines. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26 of *Advances in Neural Information Processing Systems*. Curran Associates, Lake Tahoe, NV, USA, 2013, pp. 548–556.

[120] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning* (Atlanta, GA, USA, 2013), vol. 28 of *JMLR Proceedings*, pp. 1319–1327.

[121] Google. Research at Google. https://research.google.com/teams/brain/; last visited on 2017-03-21.

[122] Google Cloud. Cloud AutoML – Custom Machine Learning Models, 2018. https://cloud.google.com/automl/; last visited on 2018-06-05.

[123] Google Cloud Platform. Overview of Hyperparameter Tuning, 2017. https://cloud.google.com/ml-engine/docs/concepts/hyperparameter-tuning-overview; last visited on 2017-06-29.

[124] Google Cloud Platform. Predictive Analytics – Google Cloud Machine Learning Engine, 2017. https://cloud.google.com/ml-engine/; last visited on 2017-06-29.

[125] Google Scholar. Alejandro Baldominos – Google Scholar Citations, 2018. https://scholar.google.es/citations?user=uVR71isAAAAJ&hl=en; last visited on 2018-05-17.

[126] Graham, B. Fractional max-pooling. *arXiv 1412.6071* (2015).

[127] Granger, E. G., Henniges, P., Sabourin, R., and Oliveira, L. S. Supervised learning of fuzzy ARTMAP neural networks through particle swarm optimisation. *Journal of Pattern Recognition Research 2*, 1 (2007), 27–60.

[128] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems PP* (2016).

[129] Grisci, B., and Dorn, M. Predicting protein structural features with NeuroEvolution of Augmenting Topologies. In *Proceedings of the 2016 International Joint Conference on Neural Networks* (Vancouver, British Columbia, Canada, 2016), pp. 873–880.

[130] Grisci, B., and Dorn, M. NEAT-FLEX: Predicting the conformational flexibility of amino acids using neuroevolution of augmenting topologies. *Journal of Bioinformatics and Computational Biology 15*, 3 (2017).

[131] Grother, P. J., and Hanaoka, K. K. NIST Special Database 19 Handprinted Forms and Characters Database. Tech. rep., National Institute of Standards and Technology, September 2016.

[132] Gruau, F. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, 1994.

[133] Grzeszick, R., Lenk, J. M., Moya Rueda, F., Fink, G. A., Feldhorst, S., and ten Hompel, M. Deep neural network based human activity recognition for the order picking process. In *Proceedings of the 4th international Workshop on Sensor-based Activity Recognition and Interaction* (Rostock, Germany, 2017).

[134] Guan, Y., and Plötz, T. Ensembles of deep lstm learners for activity recognition using wearables. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 1*, 2 (2017), 11.

[135] Guyon, I., Bennett, K., Cawley, G., Escalante, H. J., Escalera, S., Ho, T. K., Macia, N., Ray, B., Saeed, M., Statnikov, A., et al. Design of the 2015 ChaLearn AutoML Challenge. In *Proceedings of the 2015 International Joint Conference on Neural Networks* (Killarney, Ireland, 2015).

[136] Ha, D., Dai, A., and Le, Q. V. HyperNetworks. In *Proceedings of the 5th International Conference on Learning Representations* (Toulon, France, 2017).

[137] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The weka data mining software: An update. *SIGKDD Explorations 11*, 1 (2009), 10–18.

[138] Hammerla, N. Y., Halloran, S., and Plötz, T. Deep, convolutional, and recurrent models for human activity recognition using wearables. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence* (New York City, NY, USA, 2016), pp. 1533–1540.

[139] Hancock, P. J. B. Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. In *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks* (Baltimore, MD, USA, 1992), pp. 108–122.

[140] Harari, Y. N. Reboot for the AI revolution. *Nature 550*, 7676 (2017), 324–327.

[141] Harp, S. A., Samad, T., and Guha, A. Towards the genetic synthesis of neural networks. In *Proceedings of the Third International Conference on Genetic Algorithms* (Pittsburgh, PA, USA, 1989), pp. 360–369.

[142] Harp, S. A., Samad, T., and Guha, A. Designing application-specific neural networks using the genetic algorithm. In *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds., vol. 2 of *Advances in Neural Information Processing Systems*. Morgan Kaufmann, Denver, CO, USA, 1990, pp. 447–454.

[143] Harwell, D. A google program can pass as a human on the phone. should it be required to tell people it's a machine?, 10 May 2018. Published on Medium by The Washington Post. https://medium.com/thewashingtonpost/a-google-program-can-pass-as-a-human-on-the-phone-d3ea263b9577.

[144] Hazan, E., Klivans, A., and Yuan, Y. Hyperparameter optimization: A spectral approach. *arXiv 1706.00764* (2018).

[145] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *arXiv 1512.03385* (2015).

[146] Hebb, D. O. *The Organization of Behavior*. Wiley, 1949.

[147] Herculano-Houzel, S. The human brain in numbers: A linearly scaled-up primate brain. *Frontiers in Human Neuroscience 3* (2009), 31.

[148] Hermundstad, A. M., Brown, K. S., Bassett, D. S., and Carlson, J. M. Learning, memory, and the role of neural network architecture. *PLOS Computational Biology 7*, 6 (2011).

[149] Hertel, L., Barth, E., Käster, T., and Martinetz, T. Deep convolutional neural networks as generic feature extractors. In *Proceedings of the 2015 International Joint Conference on Neural Networks* (Anchorage, AK, USA, 2015).

[150] Hintzelab. MABE: Modular Agent Based Evolution Framework, 2018. https://github.com/Hintzelab/MABE; last visited on 2018-06-28.

[151] Hirose, Y., Yamashita, K., and Hijiya, S. Back-propagation algorithm which varies the number of hidden units. *Neural Networks 4*, 1 (1991), 61–66.

[152] Ho, T. K. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition* (Montreal, Quebec, Canada, 1995), pp. 278–282.

[153] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural Computation 9*, 8 (1997), 1735–1780.

[154] Holland, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[155] Hopfield, J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences 79* (1982), 2554–2558.

[156] Hsu, J. Biggest neural network ever pushes AI deep learning. *IEEE Spectrum* (July 2015). https://spectrum.ieee.org/tech-talk/computing/software/biggest-neural-network-ever-pushes-ai-deep-learning.

[157] Huang, G. B., Zhu, Q. Y., and Siew, C. K. Extreme learning machine: theory and applications. *Neurocomputing 70*, 1–3 (2006), 489–501.

[158] Hubel, D., and Wiesel, T. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology 160* (1962), 106–154.

[159] Hunt, R. Amazon SageMaker Automatic Model Tuning: Using Machine Learning for Machine Learning – Amazon Web Services, 2018. https://aws.amazon.com/es/blogs/aws/sagemaker-automatic-model-tuning/; published on 2018-06-07.

[160] Ibai, H. Evolutionary approach to deep learning. In *Evolutionary Approach to Machine Learning and Deep Neural Networks: Neuro-Evolution and Gene Regulatory Networks*. Springer, 2018, pp. 105–178.

[161] Igel, C. Neuroevolution for reinforcement learning using evolution strategies. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation* (Kobe, Japan, 2003), pp. 2588–2595.

[162] Intersense. InterSense Wireless InertiaCube3, 2017. http://forums.ni.com/attachments/ni/280/4310/1/WirelessInertiaCube3.pdf; last visited on 2017-04-05.

[163] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. Population based training of neural networks. *arXiv 1711.09846* (2017).

[164] Janakiram, M. S. V. Why AutoML Is Set To Become The Future Of Artificial Intelligence – Forbes, 2018. https://www.forbes.com/sites/janakirammsv/2018/04/15/why-automl-is-set-to-become-the-future-of-artificial-intelligence/; published on 2018-04-15.

[165] Jarrett, K., Kavukcuoglu, K., Ranzato, M. A., and LeCun, Y. What is the best multi-stage architecture for object recognition? In *Proceedings of the 2009 International Conference on Computer Vision* (Kyoto, Japan, 2009), pp. 2146–2153.

[166] Jia, Y., Huang, C., and Darrell, T. Beyond spatial pyramids: receptive field learning for pooled image features. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition* (Providence, RI, USA, 2012), pp. 3370–3377.

[167] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia* (Orlando, FL, USA, 2014), pp. 675–678.

[168] Józefowicz, R., Zaremba, W., and Sutskever, I. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd Annual International Conference on Machine Learning* (Lille, France, 2015), vol. 37 of *JMLR Proceedings*, pp. 2342–2350.

[169] Kaggle. Kaggle: Your Home for Data Science. http://www.kaggle.com; last visited on 2017-03-21.

[170] Kamath, P., Singh, A., and Dutta, D. Neural architecture construction using EnvelopeNets. *arXiv 1803.06744* (2018).

[171] Kandasamy, K., Neiswanger, W., Schneidera, J., Poczos, B., and Xing, E. Neural architecture search with Bayesian optimisation and optimal transport. *arXiv 1802.07191* (2018).

[172] Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks, 2015. http://karpathy.github.io/2015/05/21/rnn-effectiveness/; published on 2015-05-21.

[173] Karunanithi, N., Das, R., and Whitley, D. Genetic cascade learning for neural networks. In *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks* (Baltimore, MD, USA, 1992), pp. 134–145.

[174] Kassahun, Y., Edgington, M., Metzen, J. H., Sommer, G., and Kirchner, F. Common genetic encoding for both direct and indirect encodings of networks. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (London, UK, 2007), pp. 1029–1036.

[175] Kassahun, Y., and Sommer, G. Efficient reinforcement learning through evolutionary acquisition of neural topologies. In *Proceedings of the 13th European Symposium on Artificial Neural Networks* (Bruges, Belgium, 2005), pp. 259–266.

[176] Kasun, L. L. C., Zhou, H., Huang, G. B., and Vong, C. M. Representational learning with extreme learning machine for big data. *IEEE Intelligent Systems 28*, 6 (2013), 31–34.

[177] Kégl, B., and Busa-Fekete, R. Boosting products of base classifiers. In *Proceedings of the 26th Annual International Conference on Machine Learning* (Montreal, Quebec, Canada, 2009), pp. 497–504.

[178] Keinan, A., Sandbank, B., Hilgetag, C. C., Meilijson, I., and Ruppin, E. Axiomatic scalable neurocontroller analysis via the shapley value. *Artificial Life 12*, 3 (2006), 333–352.

[179] Keras. Keras: the Python deep learning library, 2017. https://keras.io; last visited on 2017-12-08.

[180] Keysers, D., Deselaers, T., Gollan, C., and Ney, H. Deformation models for image recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 8 (2007), 1422–1435.

[181] Khan, M. M., Mendes, A., Zhang, P., and Chalup, S. K. Evolving multi-dimensional wavelet neural networks for classification using Cartesian genetic programming. *Neurocomputing 247* (2017), 39–58.

[182] Khaw, J. F. C., Lim, B. S., and Lim, L. E. N. Optimal design of neural networks using the Taguchi method. *Neurocomputing 7* (1995), 225–245.

[183] Kingma, D., and Ba, J. Adam: A method for stochastic optimization. *arXiv 1412.6980* (2014).

[184] Kitano, H. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems 4* (1990), 461–476.

[185] Koerich, A. L., and Kalva, P. R. Unconstrained handwritten character recognition using metaclasses of characters. In *Proceedings of the 2005 IEEE International Conference on Image Processing* (Genoa, Italy, 2005), pp. 542–545.

[186] Koutník, J., Schmidhuber, J., and Gomez, F. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *Proceedings of the 2014 ACM Genetic and Evolutionary Computation Conference* (Vancouver, British Columbia, Canada, 2014), pp. 541–548.

[187] Koza, J. R. Hierarchical genetic algorithms operating on populations of computer programs. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (Detroit, MI, USA, 1989), pp. 7768–774.

[188] Koza, J. R., and Rice, J. P. *Genetic Programming: The Movie*. MIT Press, 1992.

[189] Kramer, O. Evolution of convolutional highway networks. In *EvoApplications 2018: Applications of Evolutionary Computation*, K. Sim and P. Kaufmann, Eds., vol. 10784 of *Lecture Notes in Computer Science*. Springer, 2018, pp. 395–404.

[190] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 25 of *Advances in Neural Information Processing Systems*. Curran Associates, Lake Tahoe, NV, USA, 2012, pp. 1097–1105.

[191] Krogh, A., and Vedelsby, J. Neural network ensembles, cross validation, and active learning. In *Proceedings of the 7th International Conference on Neural Information Processing Systems* (Denver, CO, USA, 1994), G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds., vol. 7 of *Advances in Neural Information Processing Systems*, MIT Press, pp. 231–238.

[192] Labusch, K., Barth, E., and Matinetz, T. Simple method for high-performance digit recognition based on sparse coding. *IEEE Transactions on Neural Networks 19*, 11 (2008), 1985–1989.

[193] Lamblin, P. MILA and the future of Theano, 2017. https://groups.google.com/forum/#!topic/theano-users/7Poq8BZutbY; last visited on 2017-12-01.

[194] Laplace, P. S. *Théorie Analytique des Probabilités*. Courcier, 1820.

[195] Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Bengio, Y. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning* (Corvalis, OR, USA, 2007), pp. 473–480.

[196] Lasagne. Welcome to Lasagne, 2017. http://lasagne.readthedocs.io; last visited on 2017-12-08.

[197] Lauer, F., Suen, C. Y., and Bloch, G. A trainable feature extractor for handwritten digit recognition. *Pattern Recognition 40*, 6 (2007), 1816–1824.

[198] Lawrence, S., Giles, C. L., and Tsoi, A. C. What size neural network gives optimal generalization? Convergence properties of backpropagation. Tech. rep., Institute for Advanced Computer Studies, University of Maryland, June 1996.

[199] Le, Q., and Zoph, B. Using Machine Learning to Explore Neural Network Architecture – Google AI Blog, 2017. https://ai.googleblog.com/2017/05/using-machine-learning-to-explore.html; published on 2017-05-17.

[200] Le, Q. V. Building high-level features using large scale unsupervised learning. In *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (Vancover, British Columbia, Canada, 2013).

[201] Le, Q. V., Ngiam, J., Coates, A., Prochnow, B., and Ng, A. Y. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning* (Bellevue, WA, USA, 2011), pp. 265–272.

[202] LeCun, Y., and Bengio, Y. Convolutional networks for images, speech, and time series. In *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. MIT Press, 1998, pp. 255–258.

[203] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[204] LeCun, Y., Cortes, C., and Burges, C. J. C. The MNIST database of handwritten digits, 1998. http://yann.lecun.com/exdb/mnist/; last visited on 2017-05-17.

[205] LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds., vol. 2 of *Advances in Neural Information Processing Systems*. Morgan Kaufmann, Denver, CO, USA, 1990, pp. 598–605.

[206] Lee, C. Y., Gallagher, P. W., and Tu, Z. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (Cadiz, Spain, 2015), vol. 51 of *JMLR Proceedings*, pp. 464–472.

[207] Lee, C. Y., Xie, S., Gallagher, P. W., Zhang, Z., and Tu, Z. Deeply-supervised nets. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics* (San Diego, CA, USA, 2015), vol. 38 of *JMLR Proceedings*, pp. 562–570.

[208] Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning* (Montreal, Quebec, Canada, 2009), pp. 609–616.

[209] Lehman, J., and Stanley, K. O. Abandoning objectives: evolution through the search for novelty alone. *Evolutionary Computation 19*, 2 (2011), 189–223.

[210] Lessin, D., Fussel, D., and Miikkulainen, R. Open-ended behavioral complexity for evolved virtual creatures. In *Proceedings of the 2013 ACM Genetic and Evolutionary Computation Conference* (Amsterdam, Netherlands, 2013), pp. 335–342.

[211] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *Proceedings of the 5th International Conference on Learning Representations* (Toulon, France, 2017).

[212] Liang, M., and Hu, X. Recurrent convolutional neural network for object recognition. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition* (Boston, MA, USA, 2015), pp. 3367–3375.

[213] Liao, Z., and Carneiro, G. Competitive multi-scale convolution. *arXiv 1511.05635* (2015).

[214] Liao, Z., and Carneiro, G. On the importance of normalisation layers in deep learning with piecewise linear activation units. In *Proceedings of the 2016 IEEE Winter Conference on Applications of Computer Vision* (Lake Placid, NY, USA, 2015).

[215] Limache, C. R. E., and Portugal-Zambrano, C. E. A neuroevolutive approach to the normal/abnormal classification in digital MR brain images. In *Proceedings of the XLII Latin American Computing Conference* (Valparaiso, Chile, 2016), pp. 1–10.

[216] Lin, M., Chen, Q., and Yan, S. Network in network. In *Proceedings of the 2nd International Conference on Learning Representations* (Banff, Alberta, Canada, 2014).

[217] Lindgren, K., Nilsson, A., Nordahl, M. G., and Rade, I. Regular language inference using evolving neural networks. In *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks* (Baltimore, MD, USA, 1992), pp. 75–86.

[218] Linn, A. Microsoft releases beta of Microsoft Cognitive Toolkit for deep learning advances, 2016. https://blogs.microsoft.com/ai/2016/10/25/microsoft-releases-beta-microsoft-cognitive-toolkit-deep-learning-advances; published on 2016-10-25.

[219] Linn, A. Microsoft releases CNTK, its open source deep learning toolkit, on GitHub, 2016. https://blogs.microsoft.com/ai/2016/01/25/microsoft-releases-cntk-its-open-source-deep-learning-toolkit-on-github/; published on 2016-01-25.

[220] Linnainmaa, S. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics 16*, 2 (1976), 146–160.

[221] Lipton, Z. C., and Berkowitz, J. A critical review of recurrent neural networks for sequence learning. *arXiv 1506.00019* (2015).

[222] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. *arXiv 1712.00559* (2018).

[223] Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. Hierarchical representations for efficient architecture search. In *Proceedings of the 6th International Conference on Learning Representations* (Vancouver, British Columbia, Canada, 2018).

[224] Liu, Y., Yao, X., and Higuchi, T. Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation 4*, 4 (2000), 380–387.

[225] Loshchilov, I., and Hutter, F. CMA-ES for hyperparameter optimization of deep neural networks. *arXiv 1604.07269* (2016).

[226] Mairal, J., Bach, F., and Ponce, J. Task-driven dictionary learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 4 (2012), 791–804.

[227] Mairal, J., Koniusz, P., Harchaoui, Z., and Schmid, C. Convolutional kernel networks. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26 of *Advances in Neural Information Processing Systems*. Curran Associates, Montreal, Quebec, Canada, 2013, pp. 2627–2635.

[228] Mani, G. Learning by gradient descent in function space. In *Proceedings of the 1990 IEEE International Conference on System, Man, and Cybernetics* (Los Angeles, CA, USA, 1990), pp. 242–246.

[229] Mariyama, T., Fukushima, K., and Matsumoto, W. Automatic design of neural network structures using AiS. In *Neural Information Processing*, A. Hirose, S. Ozawa, K. Doya, K. Ikeda, M. Lee, and D. Liu, Eds., vol. 9948 of *Lecture Notes in Computer Science*. Springer, Kyoto, Japan, 2016, pp. 280–287.

[230] Markoff, J. Computer Wins on 'Jeopardy!': Trivial, it's not. *The New York Times* (12 February 2011). http://www.nytimes.com/2011/02/17/science/17jeopardy-watson.html?_r=1&pagewanted=all&.

[231] Martin, C. H. TensorFlow reproductions: big deep simple MNIST, 8 June 2016. https://calculatedcontent.com/2016/06/08/tensorflow-reproductions-big-deep-simple-mnist/.

[232] Maynard Smith, J. Optimization theory in evolution. *Annual Review of Ecology and Systematics 9* (1978), 31–56.

[233] McCarthy, J. What is artificial intelligence? Basic questions, 2007. http://www-formal.stanford.edu/jmc/whatisai/node1.html; last visited on 2017-03-04.

[234] McCulloch, W. S., and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5*, 4 (1943), 115–133.

[235] McDonnell, M. D., Tissera, M. D., Vladusich, T., van Schaik, A., and Tapson, J. Fast, simple and accurate handwritten digit classification by training shallow neural network classifiers with the 'extreme learning machine' algorithm. *PLoS ONE 10*, 8 (2015).

[236] McFonnell, M. D., and Vladusich, T. Enhanced image classification with a fast-learning shallow convolutional neural network. In *Proceedings of the 2015 International Joint Conference on Neural Networks* (Killarney, Ireland, 2015).

[237] Meier, U., Cireşan, D. C., Gambardella, L. M., and Schmidhuber, J. Better digit recognition with a committee of simple neural nets. In *Proceedings of the 2011 International Conference on Document Analysis and Recognition* (Beijing, China, 2011), pp. 1250–1254.

[238] Mendoza, H., Klein, A., Feurer, M., Springenberg, J. T., and Hutter, F. Towards automatically-tuned neural networks. In *Proceedings of the Workshop on Automatic Machine Learning* (New York City, NY, USA, 2016), pp. 58–65.

[239] Miihlenbein, H., and Kindermann, J. The dynamics of evolution and learning – towards genetic neural networks. In *Connectionism in perspective*, R. Pfeifer, Z. Schreter, F. Fogelman-Soulié, and L. Steels, Eds. North Holland, 1989, pp. 173–197.

[240] Miikkulainen, R. Neuroevolution. In *Encyclopedia of Machine Learning and Data Mining*, C. Sammut and G. I. Web, Eds. Springer, 2017, pp. 899–904.

[241] Miikkulainen, R. Topology of a neural network. In *Encyclopedia of Machine Learning and Data Mining*, C. Sammut and G. I. Web, Eds. Springer, 2017, pp. 1281–1281.

[242] Miikkulainen, R., Bryant, B. D., Cornelius, R., Karpov, I. V., Stanley, K. O., and Yong, C. H. Computational intelligence in games. In *Computational Intelligence: Principles and Practice*, G. Y. Yen and D. B. Fogel, Eds. IEEE Computational Intelligence Society, 2006, pp. 155–191.

[243] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., and Hodjat, B. Evolving deep neural networks. *arXiv 1703.00548* (2017).

[244] Milgram, J., Cheriet, M., and Sabourin, R. Estimating accurate multi-class probabilities with support vector machines. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks* (Montreal, Quebec, Canada, 2005), pp. 1906–1911.

[245] Miller, G. F., Todd, P., and Hedge, S. U. Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms* (Pittsburgh, PA, USA, 1989), pp. 379–384.

[246] Min, R., Stanley, D. A., Yuan, Z., Bonner, A., and Zhang, Z. A deep non-linear feature mapping for large-margin kNN classification. *arXiv 0906.1814* (2009).

[247] Ministerio de Educación, Cultura y Deporte. Resolución de 2 de marzo de 2016, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se conceden ayudas de movilidad para estancias breves en otros centros españoles y extranjeros y para traslados temporales a centros extranjeros a beneficiarios del subprograma de formación del profesorado universitario, 2016. Published on 2016-03-04.

[248] Ministerio de Educación, Cultura y Deporte. Resolución de 13 de diciembre de 2017, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se conceden ayudas por matrícula en programas de doctorado a beneficiarios del subprograma de formación del profesorado universitario, correspondientes al curso 2017-2018, 2017. Available at: https://www.mecd.gob.es/mecd/servicios-al-ciudadano-mecd/catalogo/general/educacion/288078/ficha.html. Published on 2017-12-19.

[249] Minsky, M. L. *Theory of neural-analog reinforcement systems and its application to the brain-model problem*. PhD thesis, Princeton University, 1954.

[250] Minsky, M. L., and Papert, S. A. *Perceptrons: an introduction to computational geometry*. MIT Press, 1969.

[251] Mishkin, D., and Matas, J. All you need is a good init. In *Proceedings of the 4th International Conference on Learning Representations* (San Juan, Puerto Rico, 2016).

[252] Mishkin, D., Sergievskiy, N., and Matas, J. Systematic evaluation of CNN advances on the ImageNet. *arXiv 1606.02228* (2016).

[253] Mitchell, T. M. *Machine Learning*. McGraw Hill, 1997.

[254] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature 518* (2015), 529–533.

[255] Montana, D. J., and Davis, L. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th Joint International Conference on Artificial Intelligence* (Detroit, MI, USA, 1989), pp. 762–767.

[256] Moor, J. The Dartmouth College Artificial Intelligence Conference: The next fifty years. *AI Magazine 27*, 4 (2006), 87–89.

[257] Mouret, J.-B., and Doncieux, S. Encouraging behavioral diversity in evolutionary robotics: an empirical study. *Evolutionary Computation 20*, 1 (2012), 91–133.

[258] Moya Rueda, F., and Fink, G. A. Learning attribute representation for human activity recognition. *arXiv abs:1802.00761* (2018).

[259] Moya Rueda, F., Grzeszick, R., Fink, G. A., Feldhorst, S., and ten Hompel, M. Convolutional neural networks for human activity recognition using body-worn sensors. *Informatics 5*, 2 (2018), 26.

[260] Mozer, M. C., and Smolensky, P. Skeletonization: a technique for trimming the fat from a network via relevance assessment. In *Proceedings of the International Conference on Neural Information Processing Systems*, D. S. Touretzky, Ed., vol. 1 of *Advances in Neural Information Processing Systems*. Morgan Kaufmann, Denver, CO, USA, 1989, pp. 107–115.

[261] Myers, A. Stanford's John McCarthy, seminal figure of artificial intelligence, dies at 84. *Stanford News* (8 July 2011). https://news.stanford.edu/news/2011/october/john-mccarthy-obit-102511.html.

[262] Nand, R., and Chandra, R. Reverse neuron level decomposition for cooperative neuro-evolution of feedforward networks for time series prediction. In *Artificial Life and Computational Intelligence*, T. Ray, R. Sarker, and X. Li, Eds., vol. 9592 of *Lecture Notes in Computer Science*. Springer, Canberra, Australia, 2016, pp. 171–182.

[263] Negrinho, R., and Gordon, G. Deeparchitect: Automatically designing and training deep architectures. *arXiv 1704.08792* (2017).

[264] Nelder, J. A., and Mead, R. A simplex method for function minimization. *The Computer Journal 7*, 4 (1965), 308–313.

[265] Netflix. Netflix Prize. http://www.netflixprize.com; last visited on 2017-03-21.

[266] New York Times. New Navy device learns by doing; psychologist shows embryo of computer designed to read and grow wiser, 8 July 1958. http://www.nytimes.com/1958/07/08/archives/new-navy-device-learns-by-doing-psychologist-shows-embryo-of.html.

[267] NIST. NIST Special Database 19, 2017. https://www.nist.gov/srd/nist-special-database-19; last visited on 2017-06-11.

[268] Nitschke, G., and Parker, A. How to best automate intersection management. In *Proceedings of the 2017 IEEE Congress on Evolutionary Computation* (San Sebastian, Spain, 2017), pp. 1247–1254.

[269] Nolfi, S., and Floreano, D. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, 2000.

[270] Novet, J. Skymind launches with open-source, plug-and-play deep learning features for your app, 2014. https://venturebeat.com/2014/06/02/skymind-launches-with-open-source-plug-and-play-deep-learning-features-for-your-app/; published on 2014-06-02.

[271] NVIDIA. NVIDIA GeForce GTX 1080 Whitepaper – Gaming Perfected. http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf; last visited on 2017-08-16.

[272] NVIDIA Developer. CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit; last visited on 2017-05-29.

[273] NVIDIA Developer. NVIDIA cuDNN. https://developer.nvidia.com/cudnn; last visited on 2017-05-29.

[274] Odri, S. V., Petrovacki, D. P., and Krstonosic, G. A. Evolutional development of a multilevel neural network. *Neural Networks 6*, 4 (1993), 583–595.

[275] O'Keeffe, E. Natural Computing – PonyGE. https://github.com/eokeeffe/Natural-Computing/tree/master/ponyge-0.1.5/ponyge-0.1.5; last visited on 2017-05-29.

[276] Oliveira, L. E. S., Sabourin, R., Bortolozzi, F., and Suen, C. Y. Automatic recognition of handwritten numerical strings: a recognition and verification strategy. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 11 (2002), 1438–1454.

[277] Ordóñez, F. J., and Roggen, D. Deep convolutional and LSTM recurrent neural neworks for multimodal wearable activity recognition. *Sensors 16*, 1 (2016).

[278] Parker, G. A., and Maynard Smith, J. Optimality theory in evolutionary biology. *Nature 348* (1990), 27–33.

[279] Peng, Y., and Yin, H. Markov random field based convolutional neural networks for image classification. In *IDEAL 2017: Intelligent Data Engineering and Automated Learning*, H. Yin, Y. Gao, S. Chen, Y. Wen, G. Cai, T. Gu, J. Du, A. Tallón-Ballesteros, and M. Zhang, Eds., vol. 10585 of *Lecture Notes in Computer Science*. Springer, Guilin, China, 2017, pp. 387–396.

[280] Peralta, J., Li, X., Gutierrez, G., and Sanchis, A. Time series forecasting by evolving artificial neural networks using genetic algorithms and differential evolution. In *Proceedings of the 2010 IEEE World Congress on Computational Intelligence* (Barcelona, Spain, 2010), pp. 3999–4006.

[281] Pfeifer, R., and Bongard, J. *How the Body Shapes the Way We Think – A New View of Intelligence*. MIT Press, 2007.

[282] Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. *arXiv 1802.03268* (2018).

[283] Pichai, S. Ai at google: our principles, 2018. https://blog.google/technology/ai/ai-principles/; published on 2018-06-07.

[284] Pirkl, G., Stockinger, K., Kunze, K., and Lukowicz, P. Adapting magnetic resonant coupling based relative positioning technology for wearable activity recognition. In *Proceedings of the 2008 International Symposium on Wearable Computers* (Pittsburgh, PA, USA, 2008), pp. 47–54.

[285] Pomerleau, D. A. ALVINN: an autonomous land vehicle in a neural network. In *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, D. S. Touretzky, Ed., vol. 1 of *Advances in Neural Information Processing Systems*. Morgan Kaufmann, Denver, CO, USA, 1989, pp. 305–313.

[286] Prechelt, L. Neural Net FAQ, 1995. https://www.cs.cmu.edu/Groups/AI/util/html/faqs/ai/neural/faq.html; last modified on 1995-02-23.

[287] Prellberg, J., and Kramer, O. Lamarckian evolution of convolutional neural networks. *arXiv 1806.08099* (2018).

[288] Prellberg, J., and Kramer, O. Limited evaluation evolutionary optimization of large neural networks. *arXiv 1806.09819* (2018).

[289] Project, O. Activity Recognition Challenge, 27 May 2011. http://opportunity-project.eu/challenge; published on May 27, 2011.

[290] PyTorch. Tensors and dynamic neural networks in Python with strong GPU acceleration, 2017. http://pytorch.org; last visited on 2017-12-07.

[291] Quinlan, J. R. Induction of decision trees. *Machine Learning 1*, 1 (1986), 81–106.

[292] Quinlan, J. R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[293] Radtke, P. V. W., Sabourin, R., and Wong, T. Using the RRT algorithm to optimize classification systems for handwritten digits and letters. In *Proceedings of the 2008 ACM Symposium on Applied Computing* (Fortaleza, Ceara, Brazil, 2008), pp. 1748–1752.

[294] Ramón Cajal, S. The croonian lecture: la fine structure des centres nerveux. *Proceedings of the Royal Society of London 55* (1894), 444–468.

[295] Ramón Cajal, S. *Textura del sistema nervioso del hombre y de los vertebrados*, vol. I. Imprenta y Librería de Nicolás Moya, 1899.

[296] Ranzato, M. A., Huang, F. J., Boureau, Y. L., and LeCun, Y. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition* (Minneapolis, MN, USA, 2007).

[297] Ranzato, M. A., Poultney, C., Chopra, S., and LeCun, Y. Efficient learning of sparse representations with an energy-based model. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, B. Schölkopf, J. Platt, and T. Hofmann, Eds., vol. 19 of *Advances in Neural Information Processing Systems*. Curran Associates, Vancouver, British Columbia, Canada, 2006, pp. 1137–1144.

[298] Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. *arXiv 1802.01548* (2018).

[299] Real, E., Moore, S., Selle, A., Saxena, S., Leon-Suematsu, Y., Tan, J., Le, Q. V., and Kurakin, A. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning* (Sydney, Australia, 2017), vol. 70 of *JMLR Proceedings*.

[300] Rechenberg, I. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Technische Universität Berlin, 1971.

[301] ResearchGate. Alejandro Baldominos, 2018. https://www.researchgate.net/profile/Alejandro_Baldominos; last visited on 2018-05-17.

[302] Risi, S., and Stanley, K. O. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life 18*, 4 (2012), 331–363.

[303] Risi, S., and Togelius, J. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games 9*, 1 (2017), 25–41.

[304] Roggen, D., Bächlin, M., Schümm, J., Holleczek, T., Lombriser, C., Tröster, G., Widmer, L., Majoe, D., and Gutknecht, J. An educational and research kit for activity and context recognition from on-body sensors. In *Proceedings of the 2010 International Conference on Body Sensor Networks* (Singapore, 2010), pp. 277–282.

[305] Roggen, D., Calatroni, A., Rossi, M., Holleczek, T., Förster, K., Tröster, G., Lukowicz, P., Bannach, D., Pirkl, G., Ferscha, A., Doppler, J., Holzmann, C., Kurz, M., Holl, G., Chavarriaga, R., Sagha, H., Bayati, H., Creatura, M., and del R. Millàn, J. Collecting complex activity datasets in highly rich networked sensor environments. In *Proceedings of the Seventh International Conference on Networked Sensing Systems* (Kassel, Germany, 2010).

[306] Roggen, D., Tröster, G., Lukowicz, P., Ferscha, A. ., and del R. Millán, J. OPPORTUNITY Deliverable D5.1: Stage 1 Case Study Report and Stage 2 Specification. Tech. rep., University of Passau, 2010.

[307] Rosenblatt, F. The perceptron–a perceiving and recognizing automaton. Tech. rep., Cornell Aeronautical Laboratory, 1957.

[308] Rumelhart, D., Hinton, G., and Williams, R. J. Learning representations by back-propagating errors. *Nature 323* (1986), 533–536.

[309] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., and Berg, A. C. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision 115*, 3 (2015), 211–252.

[310] Ryan, C., Collins, J. J., and O'Neill, M. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming*, W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, Eds., vol. 1391 of *Lecture Notes in Computer Science*. Springer, Paris, France, 1998, pp. 83–96.

[311] Sagha, H., Digumarti, S. T., del R. Millán, J., Chavarriaga, R., Calatroni, A., Roggen, D., and Tröster, G. Benchmarking classification techniques using the Opportunity human activity dataset. In *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics* (Anchorage, AK, USA, 2011), pp. 36–40.

[312] Salakhutdinov, R., and Hinton, G. Deep Boltzmann machines. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics* (Clearwater Beach, FL, USA, 2009), vol. 5 of *JMLR Proceedings*, pp. 448–455.

[313] Sammut, C., and Webb, G. I., Eds. *Encyclopedia of Machine Learning and Data Mining*. Springer, 2017.

[314] Samothrakis, S., Perez-Liebana, D., Lucas, S. M., and Fasli, M. Neuroevolution for general video game playing. In *Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games* (Tainan, Taiwan, 2015), pp. 200–207.

[315] Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development 3*, 3 (1959), 210–229.

[316] Sato, I., Nishimura, H., and Yokoi, K. Apac: Augmented pattern classification with neural networks. *arXiv 1505.03229* (2015).

[317] Saxena, S., and Verbeek, J. Convolutional neural fabrics. In *Proceedings of the 29th International Conference on Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29 of *Advances in Neural Information Processing Systems*. Curran Associates, Barcelona, Spain, 2016, pp. 4053–4061.

[318] Schaffer, J. D., Caruana, R. A., and Eshelman, L. J. Using genetic search to exploit the emergent behavior of neural networks. *Physica D: Nonlinear Phenomena 42*, 1–3 (1990), 244–248.

[319] Schaffer, J. D., Whitley, D., and Eshelman, L. J. Combinations of genetic algorithms and neural networks: a survey of the state of the art. In *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks* (Baltimore, MD, USA, 1992), pp. 1–37.

[320] Schapire, R. E. The strength of weak learnability. *Machine Learning 5*, 2 (1990), 197–227.

[321] Schiffmann, W., Joost, M., and Werner, R. Performance evaluation of evolutionarily created neural network topologies. In *Parallel Problem Solving from Nature*, H. P. Schwefel and R. Männer, Eds., vol. 496 of *Lecture Notes in Computer Science*. Springer, Dortmund, Germany, 1991, pp. 244–248.

[322] Scholz, M. A learning strategy for neural networks based on a modified evolutionary strategy. In *Parallel Problem Solving from Nature*, H. P. Schwefel and R. Männer, Eds., vol. 496 of *Lecture Notes in Computer Science*. Springer, Dortmund, Germany, 1991, pp. 314–318.

[323] Schrum, J. B. *Evolving multimodal behavior through modular multiobjective neuroevolution*. PhD thesis, The University of Texas at Austin, 2014.

[324] Schwefel, H. P. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technische Universität Berlin, 1974.

[325] Searle, J. Computing machinery and intelligence. *Behavioral and Brain Sciences 3*, 3 (1980), 417–424.

[326] Serre, T., Kreiman, G., Kouh, M., Cadieu, C., Knoblich, U., and Poggio, T. A quantitative theory of immediate visual recognition. In *Computational Neuroscience: Theoretical Insights into Brain Function*, P. Cisek, T. Drew, and J. Kalaska, Eds., vol. 165 of *Progress in Brain Research*. Elsevier Science, 2007, pp. 33–56.

[327] Shi, S., Wang, Q., Xu, P., and Chu, X. Benchmarking state-of-the-art deep learning software tools. *arXiv 1608.07249* (2017).

[328] Siebel, N. T., and Sommer, G. Evolutionary reinforcement learning of artificial neural networks. *International Journal of Hybrid Intelligent Systems 4*, 3 (2007), 171–183.

[329] Sietsma, J., and Dow, R. J. F. Creating artificial neural networks that generalize. *Neural Networks 4*, 1 (1991), 67–79.

[330] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Graepel, T., Lillicrap, T., Leach, M., Kavukcuoglu, K., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature 529* (2016), 484–489.

[331] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of Go without human knowledge. *Nature 550* (2017), 354–359.

[332] Silverman, B. W., and Jones, M. C. E. Fix and J.L. Hodges (1951): An important contribution to nonparametric discriminant analysis and density estimation: Commentary on Fix and Hodges (1951). *International Statistical Review 57*, 3 (1989), 233–238.

[333] Simard, P., Steinkraus, D., and Platt, J. C. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the 7th International Conference on Document Analysis and Recognition – Volume 2* (Edinburgh, UK, 2003), pp. 958–963.

[334] Singh, S., Paul, A., and Arun, M. Parallelization of digit recognition system using deep convolutional neural network on CUDA. In *Proceedings of the 2017 Third International Conference on Sensing, Signal Processing and Security* (Chennai, India, 2017), pp. 379–383.

[335] Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 25 of *Advances in Neural Information Processing Systems*. Curran Associates, Lake Tahoe, NV, USA, 2012, pp. 2951–2959.

[336] Solomonoff, R. J. The time scale of Artificial Intelligence. *Reflections on Social Effects, Human Systems Management 5* (1985), 149–153.

[337] Somerville, H. Uber debuts self-driving vehicles in landmark Pittsburgh trial. *Reuters* (14 September 2016). http://www.reuters.com/article/us-uber-autonomous-idUSKCN11K12Y.

[338] Sossa, H., Garro, B. A., Villegas, J., Avilés, C., and Olague, G. Automatic design of artificial neural networks and associative memories for pattern classification and pattern restoration. In *Pattern Recognition*, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, J. A. Olvera López, and K. L. Boyer, Eds., vol. 7329 of *Lecture Notes in Computer Science*. Springer, Huatulco, Mexico, 2012, pp. 23–34.

[339] Spencer, H. *The Principles of Psychology*. D. Appleton and Company, 1872.

[340] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[341] Srivastava, R. K., Greff, K., and Schmidhuber, J. Training very deep networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28 of *Advances in Neural Information Processing Systems*. Curran Associates, Montreal, Quebec, Canada, 2015, pp. 2377–2385.

[342] Stanley, K. O. Neuroevolution: A different kind of deep learning. The quest to evolve neural networks through evolutionary algorithms – O'Reilly, 2017. https://www.oreilly.com/ideas/neuroevolution-a-different-kind-of-deep-learning; published on 2017-07-13.

[343] Stanley, K. O., and Clune, J. Welcoming the Era of Deep Neuroevolution – Uber, 2017. https://eng.uber.com/deep-neuroevolution/; published on 2017-12-18.

[344] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life 15*, 2 (2009), 185–212.

[345] Stanley, K. O., and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary Computation 10*, 2 (2002), 99–127.

[346] Stanley, K. O., and Miikkulainen, R. Competitive co-evolution through evolutionary complexification. *Journal of Artificial Intelligence Research 21* (2004), 63–100.

[347] Stiefmeier, T., Roggen, D., Ogris, G., Lukowicz, P., and Tröster, G. Wearable activity tracking in car manufacturing. *IEEE Pervasive Computing 7*, 2 (2008), 42–50.

[348] Stigler, S. M. An attack on Gauss published by Legendre in 1820. *Historia Mathematica 4* (1977), 31–35.

[349] Stigler, S. M. Gauss and the invention of least squares. *Annals of Statistics 9* (1981), 465–474.

[350] Stiles, J., and Jernigan, T. L. The basics of brain development. *Neuropsychology Review 20*, 4 (2010), 327–348.

[351] Suganuma, M., Shirakawa, S., and Nagao, T. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the 2017 ACM Genetic and Evolutionary Computation Conference* (Berlin, Germany, 2017).

[352] Sun, Y., Xue, B., and Zhang, M. Evolving deep convolutional neural networks for image classification. *arXiv 1710.10741* (2017).

[353] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. Inception-v4, Inception-ResNet and the impact of residual connections on learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence* (San Francisco, CA, USA, 2017), pp. 4278–4284.

[354] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition* (Boston, MA, USA, 2015), pp. 1–9.

[355] T3chFest. Augmented Machine Learning – Automatización de Data Science, 2018. https://t3chfest.uc3m.es/2018/programa/augmented-machine-learning-automatizacion-data-science/?lang=es; last visited on 2018-06-11.

[356] Talbi, E. G. *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009.

[357] TensorFlow. Deep MNIST for Experts. https://www.tensorflow.org/get_started/mnist/pros; last updated on 2017-11-02.

[358] Tesauro, G. Temporal difference learning and TD-Gammon. *Communications of the ACM 38*, 3 (1995), 58–68.

[359] Thom, M., and Palm, G. Sparse activity and sparse connectivity in supervised learning. *Journal of Machine Learning Research 14* (2013), 1091–1143.

[360] Thorndike, E. L. Animal intelligence: An experimental study of the associative processes in animals. *Psychological Review 2*, 4 (1898).

[361] Thorndike, E. L. *The Fundamentals of Learning*. AMS Press New York, 1932.

[362] Tieleman, T., and Hinton, G. Neural networks for machine learning, lecture 6.5 – RMSProp, 2012. Coursera, video available in http://www.youtube.com/watch?v=O3sxAc4hxZU.

[363] Tirumala, S. S., Ali, S., and Ramesh, C. P. Evolving deep neural networks: A new prospect. In *Proceedings of the 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery* (Changsha, China, 2016), pp. 69–74.

[364] Togelius, J., and Lucas, S. Evolving robust and specialized car racing skills. In *Proceedings of the 2006 IEEE Conference on Evolutionary Computation* (Vancover, British Columbia, Canada, 2006), pp. 1187–1194.

[365] Torreele, J. Temporal processing with recurrent networks: An evolutionary approach. In *Proceedings of the Fourth International Conference on Genetic Algorithms* (San Diego, CA, USA, 1991), pp. 555–561.

[366] Tsoulos, I., Gavrilis, D., and Glavas, E. Neural network construction and training using grammatical evolution. *Neurocomputing 72*, 1–3 (2008), 269–277.

[367] Turing, A. M. Computing machinery and intelligence. *Mind 59* (1950), 433–460.

[368] Universidad Carlos III de Madrid – Consejo Social. Premios de Excelencia 2017, 2018. https://www.uc3m.es/ss/Satellite/ConsejoSocial/es/TextoDosColumnas/1371227802145/; published on 2018-05-18.

[369] Vafeias, S. Design by Evolution: How to evolve your neural network with AutoML – KDnuggets, 2017. https://www.kdnuggets.com/2017/07/design-evolution-evolve-neural-network-automl.html; published on 2017-07-01.

[370] Valsalam, V., Hiller, J., MacCurdy, R., Lipson, H., and Miikkulainen, R. Constructing controllers for physical multilegged robots using the ENSO neuroevolution approach. *Evolutionary Intelligence 5*, 1 (2012), 45–56.

[371] van Schaik, A., and Tapson, J. Online and adaptive pseudoinverse solutions for ELM weights. *Neurocomputing 149A* (2015), 233–238.

[372] Vargas, D. V., and Murata, J. Spectrum-diverse neuroevolution with unified neural models. *IEEE Transactions on Neural Networks and Learning Systems PP*, 99 (2016), 1–15.

[373] Verbancsics, P., and Harguess, J. Image classification using generative neuroevolution for deep learning. In *Proceedings of the 2015 IEEE Winter Conference on Applications of Computer Vision* (Waikoloa Beach, HI, USA, 2015), pp. 488–493.

[374] Vinyard, J. Efficient overlapping windows with Numpy, 30 Aug 2012. http://www.johnvinyard.com/blog/?p=268; published on Oct 2, 2012.

[375] Visin, F., Kastner, K., Cho, K., Matteucci, M., Courville, A., and Bengio, Y. ReNet: A recurrent neural network based alternative to convolutional networks. *arXiv 1505.00393* (2015).

[376] Vonk, E., Jain, L. C., and Johnson, R. P. *Automatic Generation of Neural Network Architecture Using Evolutionary Computation*, vol. 14 of *Advances in Fuzzy Systems – Applications and Theory*. World Scientific Publishing, 1997.

[377] Vonk, E., Jain, L. C., Veelenturf, L. P. J., and Johnson, R. P. Automatic generation of a neural network architecture using evolutionary computation. In *Proceedings of Electronic Technology Directions to the Year 2000* (Adelaide, Australia, 1995), pp. 144–149.

[378] Wan, L., Zeiler, M., Zhang, S., LeCun, Y., and Fergus, R. Regularization of neural networks using DropConnect. In *Proceedings of the 30th International Conference on Machine Learning* (Atlanta, GA, USA, 2013), vol. 28 of *JMLR Proceedings*, pp. 1058–1066.

[379] Wang, B., Sun, Y., Xue, B., and Zhang, M. Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. *arXiv 1803.06492* (2017).

[380] Wang, D., and Tan, X. Unsupervised feature learning with C-SVDDNet. *Pattern Recognition 60* (2016), 473–485.

[381] Wang, Z., Di Massimo, C., Tham, M. T., and Morris, A. J. A procedure for determining the topology of multilayer feedforward neural networks. *Neural Networks 7*, 2 (1994), 291–300.

[382] Watkins, C. *Learning from delayed rewards*. PhD thesis, King's College, 1989.

[383] Webb, G. I. Decision tree grafting from the all-tests-but-one partition. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence* (Stockholm, Sweden, 1999), vol. 2, pp. 702–707.

[384] Weber, B. Swift and slashing, computer topples Kasparov. *The New York Times* (12 May 1997). http://www.nytimes.com/1997/05/12/nyregion/swift-and-slashing-computer-topples-kasparov.html.

[385] Weiser, M. The computer for the 21st century. *Scientific American 265*, 3 (1991), 94–104.

[386] Werner, G. M., and Dyer, M. G. Evolution of communication in artificial organisms. In *Proceedings of the Workshop on Artificial Life* (Santa Fe, NM, USA, 1992), pp. 659–687.

[387] Whitley, D., Dominic, S., and Das, R. Genetic reinforcement learning with multi-layer neural networks. In *Proceedings of the Fourth International Conference on Genetic Algorithms* (San Diego, CA, USA, 1991), pp. 562–569.

[388] Whitley, D., and Hanson, T. Optimizing neural networks using faster, more accurate genetic search. In *Proceedings of the Third International Conference on Genetic Algorithms* (Pittsburgh, PA, USA, 1989), pp. 391–396.

[389] Widrow, B. An adaptive "Adaline" neuron using chemical "Memistors". Tech. rep., Office of Naval Research, October 1960.

[390] Williams, R., and Kerrup, K. The control of neuron number. *The Annual Review of Neuroscience 11* (1988), 423–453.

[391] Wistuba, M. Finding competitive network architectures within a day using UCT. *arXiv 1712.07420* (2017).

[392] Wong, G., Chandra, R., and Sharma, A. Memetic cooperative neuro-evolution for chaotic time series prediction. In *Neural Information Processing*, A. Hirose, S. Ozawa, K. Doya, K. Ikeda, M. Lee, and D. Liu, Eds., vol. 9949 of *Lecture Notes in Computer Science*. Springer, Kyoto, Japan, 2016, pp. 299–308.

[393] Xie, L., and Yuille, A. Genetic CNN. In *Proceedings of the 2017 IEEE International Conference on Computer Vision* (Venice, Italy, 2017).

[394] Xsens. IMU Inertial Measurement Unit – Xsens 3D motion tracking, 2017. https://www.xsens.com/tags/imu/; last visited on 2017-04-05.

[395] Xu, C., Lu, C., Liang, X., Gao, J., Zheng, W., Wang, T., and Yan, S. Multi-loss regularized deep neural network. *IEEE Transactions on Circuits and Systems for Video Technology 26*, 12 (2015), 2273–2283.

[396] Yang, J., Yu, K., and Huang, T. Supervised translation-invariant sparse coding. In *Proceedings of the 2010 IEEE Conference on Computer Vision and Pattern Recognition* (San Francisco, CA, USA, 2010), pp. 3517–3524.

[397] Yang, J. B., Nguyen, M. N., San, P. P., Li, X. L., and Krishnaswamy, S. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Proceedings of the 24th International Conference on Artificial Intelligence* (Buenos Aires, Argentina, 2015), pp. 3995–4001.

[398] Yang, Z., Moczulski, M., Denil, M., de Freitas, N., Smola, A., Song, L., and Wang, Z. Deep fried convnets. In *Proceedings of the 2015 IEEE International Conference on Computer Vision* (Santiago, Chile, 2015), pp. 1476–1483.

[399] Yao, R., Lin, G., Shi, Q., and Ranasinghe, D. C. Efficient dense labelling of human activity sequences from wearables using fully convolutional networks. *Pattern Recognition 78* (2018), 252–266.

[400] Yao, X. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems 8*, 4 (1993), 539—567.

[401] Yao, X. Evolving artificial neural networks. *Proceedings of the IEEE 87*, 9 (1999), 1423–1447.

[402] Yao, X., and Liu, Y. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks 8*, 3 (1997), 694–713.

[403] Young, S. R., Rose, D. C., Johnston, T., Heller, W. T., Karnowski, T. P., Potok, T. E., Patton, R. M., Perdue, G., and Miller, J. Evolving deep networks using HPC. In *Proceedings of the Machine Learning on HPC Environments* (Denver, CO, USA, 2017), pp. 3924–3928.

[404] Young, S. R., Rose, D. C., Karnowsky, T. P., Lim, S.-H., and Patton, R. M. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments* (Austin, TX, USA, 2015).

[405] Yudkowsky, E. Creating friendly AI 1.0: The analysis and design of benevolent goal architectures. Tech. rep., Machine Intelligence Research Institute, 2001.

[406] Zappi, P., Lombriser, C., Stiefmeier, T., Farella, E., Roggen, D., Benini, L., and Tröster, G. Activity recognition from on-body sensors: Accuracy-power trade-off by dynamic sensor selection. In *Wireless Sensor Networks*, R. Verdone, Ed., vol. 4913 of *Lecture Notes in Computer Science*. Springer, Bologna, Italy, 2008, pp. 17–33.

[407] Zeiler, M. D. ADADELTA: an adaptive learning rate method. *arXiv 1212.5701* (2012).

[408] Zeiler, M. D., and Fergus, R. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv 1301.3557* (2013).

[409] Zhang, S., Jiang, H., and Dai, L. Hybrid orthogonal projection and estimation (HOPE): A new framework to learn neural networks. *Journal of Machine Learning Research 17* (2016), 1–33.

[410] Zimmer, C. 100 trillion connections: New efforts probe and map the brain's detailed architecture. *Scientific American* (January 2011). https://www.scientificamerican.com/article/100-trillion-connections/.

[411] Zipf, G. K. *Human behavior and the principle of least effort*. Addison-Wesley Press, 1949.

[412] Zoph, B., and Le, Q. V. Neural architecture search with reinforcement learning. In *Proceedings of the 5th International Conference on Learning Representations* (Poughkeepsie, NY, USA, 2017).

[413] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. *arXiv 1707.07012* (2018).

[414] Zweifel. Physis-Shard: Artificial Intelligence Library, 2017. https://github.com/zweifel/Physis-Shard; last visited on 2017-06-27.

This page has been intentionally left blank.