

MDSAA

Master Degree Program in
Data Science and Advanced Analytics

A Genetic Programming Approach for Computer Vision

Classifying High-level Image Features from Convolutional Layers with
an Evolutionary Algorithm

Rui Filipe Martins Monteiro

Dissertation

presented as partial requirement for obtaining the Master Degree Program in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Title: A Genetic Programming Approach for Computer Vision

Subtitle: Classifying High-level Image Features from Convolutional Layers with an Evolutionary Algorithm

Rui Filipe Martins Monteiro

MDSAA

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

A GENETIC PROGRAMMING APPROACH FOR COMPUTER VISION

by

Rui Filipe Martins Monteiro

Dissertation presented as partial requirement for obtaining the Master's degree in Data Science and Advanced Analytics, with a Specialization in Data Science

Co Supervisor: Mauro Castelli

Co Supervisor: Leonardo Vanneschi

November 2022

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Rui Filipe Martins Monteiro

Lisbon, September 19th, 2022

ACKNOWLEDGEMENTS

I want to thank each professor and colleague that directly or indirectly inspired my work over the last five years at NOVA Information Management School.

A special acknowledgment to my supervisors, Professors Mauro Castelli and Leonardo Vanneschi, for giving me the feedback and assistance I needed to develop this thesis.

Finally, I would like to thank my family, especially my parents, for providing me with all the support I needed during my bachelor's and master's degrees, as it contributed to my development both as a student and a person. They always believed in me, and in my capabilities, which I will always be grateful for. *Obrigado!*

ABSTRACT

Computer Vision is a sub-field of Artificial Intelligence that provides a visual perception component to computers, mimicking human intelligence. One of its tasks is image classification and Convolutional Neural Networks (CNNs) have been the most implemented algorithm in the last few years, with few changes made to the fully-connected layer of those neural networks. Nonetheless, recent research has been showing their accuracy could be improved in certain cases by implementing other algorithms for the classification of high-level image features from convolutional layers. Thus, the main research question for this document is: To what extent does the substitution of the fully-connected layer in Convolutional Neural Networks for an evolutionary algorithm affect the performance of those CNN models? The proposed two-step approach in this study does the classification of high-level image features with a state-of-the-art GP-based algorithm for multiclass classification called M4GP. This is conducted using secondary data with different characteristics, to better benchmark the implementation and to carefully investigate different outcomes. Results indicate the new learning approach yielded similar performance in the dataset with a low number of output classes. However, none of the M4GP models was able to surpass the results of the fully-connected layers in terms of test accuracy. Even so, this might be an interesting route if one has a powerful computer and needs a very light classifier in terms of model size. The results help to understand in which situation it might be beneficial to perform a similar experimental setup, either in the context of a work project or concerning a novel research topic.

KEYWORDS

Computer Vision; Genetic Programming; Deep Learning; Convolutional Neural Networks

INDEX

1. Introduction	1
1.1. Computer Vision	1
1.2. Genetic Programming.....	2
1.3. Purpose of the Thesis	3
1.4. Outline of the Thesis.....	4
2. Artificial Neural Networks in Computer Vision	6
2.1. Artificial Neural Networks	6
2.1.1. Perceptron	6
2.1.2. Multilayer Perceptron.....	8
2.2. Convolutional Neural Networks for Image Classification.....	10
2.2.1. Computer Vision Challenges.....	12
2.3. Evolution of Convolutional Neural Network Architectures.....	13
2.3.1. AlexNet	14
2.3.2. ZFNet.....	14
2.3.3. GoogLeNet	15
2.3.4. VGG-16.....	16
2.3.5. ResNet.....	17
3. Multiclass Classification with Genetic Programming	19
3.1. M2GP: A Multidimensional GP Approach for Multiclass Classification Problems	19
3.1.1. Training Phase.....	19
3.1.2. Testing Phase	20
3.1.3. Downside	20
3.2. M3GP: M2GP with Multidimensional Populations.....	20
3.2.1. Initial Population.....	20
3.2.2. Mutation	20
3.2.3. Crossover	21
3.2.4. Pruning.....	21
3.2.5. Elitism.....	21
3.2.6. Downsides.....	21
3.3. eM3GP: Ensemble M3GP	22
3.3.1. Specialized Class Transformations.....	22
3.3.2. Ensemble Classifier	22
3.3.3. Ensemble Construction.....	22
3.4. M4GP: The State-of-the-art Algorithm.....	23
3.4.1. Program Representation.....	23

3.4.2. Initialization	23
3.4.3. Selection	23
3.4.4. Variation	24
3.4.5. Classification Strategy.....	24
3.4.6. Archiving Solutions	25
3.4.7. Discussion about M4GP	25
4. Image Classification Using Convolutional Neural Networks and Genetic Programming	27
4.1. Related Work.....	27
5. Methods	31
5.1. Scientific Procedure.....	31
5.2. Datasets.....	33
5.2.1. CIFAR-10.....	33
5.2.2. CIFAR-100.....	35
5.3. CNN Model Architectures.....	37
5.3.1. CNN 1	37
5.3.2. CNN 2	38
5.3.3. VGG-16.....	38
5.4. Image Preprocessing	39
5.5. Classification Algorithm: M4GP	40
5.6. Evaluation Metrics.....	41
5.7. Software Framework	41
6. Results and Discussion	43
6.1. Creation of Intermediate Datasets.....	43
6.1.1. CIFAR-10.....	43
6.1.2. CIFAR-100.....	45
6.2. M4GP Classification on Intermediate Datasets.....	47
6.2.1. CIFAR-10.....	47
6.2.2. CIFAR-100.....	48
6.3. Analysis of Results	49
6.3.1. CIFAR-10.....	49
6.4. Discussion	50
7. Conclusions	54
8. Limitations and Recommendations for Future Works.....	56
9. References.....	57
Appendix.....	60

LIST OF FIGURES

Figure 1.1 – Other CV tasks, from Li et al. (2017).....	1
Figure 1.2 – Architecture of the first published CNN, LeNet-5, from LeCun et al. (1998)	1
Figure 1.3 – Workflow of GP, from Bakurov (2018).....	3
Figure 2.1 – Perceptron, from Mitchell (1997).....	7
Figure 2.2 – The decision surface is represented by a two-input perceptron, from Mitchell (1997). (a) A set of training examples and the decision surface of a perceptron that classifies them correctly (b) A set of training examples that is not linearly separable. x_1 and x_2 are the perceptron inputs. Positive examples are indicated by “+”, negative by “-”	7
Figure 2.3 – Three-layer MLP, from Mohamed et al. (2015).....	8
Figure 2.4 – Error surface with weight space and a global minimum, from Iman et al. (2021).....	9
Figure 2.5 – Three of the most popular activation functions for MLPs, left-to-right: threshold function, sigmoid function, and hyperbolic tangent; from Kriesel (2007).....	10
Figure 2.6 – Main operations in a CNN – convolution, nonlinearity, pooling, and classification; from Kaplanoglou (2017).....	11
Figure 2.7 – The convolution operation – in this example, hyperlocal edges combine into local objects such as eyes or ears, which combine into the high-level concept of “cat”; from Chollet (2017)	12
Figure 2.8 – Visual understanding of Computer Vision challenges, from Grauman & Leibe (2011).....	13
Figure 2.9 – Example of within-class variation, from Fei-Fei et al. (2005)	13
Figure 2.10 – AlexNet architecture, with a delineation of responsibilities defined for training on two GPUs, from Krizhevsky et al. (2012)	14
Figure 2.11 – Architecture of ZFNet, an 8-layer CNN model, from Zeiler & Fergus (2014)	15
Figure 2.12 – Inception module, from Szegedy et al. (2015)	16
Figure 2.13 – GoogLeNet network, from Szegedy et al. (2015)	16
Figure 2.14 – VGG-16 network architecture, from Ferguson et al. (2017)	17
Figure 2.15 – Training error (left) and test error (right) on the CIFAR-10 dataset with 20-layer and 56-layer “plain” networks: the deeper network has higher training error, and thus test error; from He et al. (2016)	17
Figure 2.16 – Two types of residual blocks, from He et al. (2016).....	18
Figure 2.17 – A Residual Network with 34 parameter layers, from He et al. (2016)	18
Figure 3.1 – On the left, the clustering is obtained by an individual with low accuracy, while the one on the right has very high accuracy, from Muñoz et al. (2015); the large circles represent the class centroids	20
Figure 3.2 – Identification of the best transformation for each class based on TP and FP, from Silva et al. (2016).....	22
Figure 5.1 – This stage encompasses the creation of intermediate datasets, which can be considered as the flattened outputs of the convolutional layers on a CNN architecture	32
Figure 5.2 – This stage encompasses the classification of the intermediate datasets with M4GP	33

Figure 5.3 – Image examples for each one of the CIFAR-10’s classes; going left to right and top to bottom, the image classes are: “airplane”, “automobile”, “bird”, “cat”, “deer”, “dog”, “frog”, “horse”, “ship” and “truck”	34
Figure 5.4 – Examples for class “automobile” on CIFAR-10	35
Figure 5.5 – Image examples for 10 of CIFAR-100’s classes. Going left to right and top to bottom, the image classes are: “apples”, “aquarium fish”, “baby”, “bear”, “beaver”, “bed”, “bee”, “beetle”, “bicycle” and “bottles”	36
Figure 5.6 – Examples for class “apples” on CIFAR-100	37
Figure 5.7 – Raw images for classes “horse” and “ship” (on the top) and their preprocessed versions (on the bottom)	40
Figure 6.1 – CNN 1 training process (accuracy and loss) on CIFAR-10	44
Figure 6.2 – CNN 2 training process (accuracy and loss) on CIFAR-10	44
Figure 6.3 – CNN 1 training process (accuracy and loss) on CIFAR-100	46
Figure 6.4 – CNN 2 training process (accuracy and loss) on CIFAR-100	46
Figure A.1 – CNN 1’s architecture	60
Figure A.2 – CNN 2’s architecture	61

LIST OF TABLES

Table 6.1 – CIFAR-10 intermediate dataset results.....	43
Table 6.2 – CIFAR-100 intermediate dataset results.....	45
Table 6.3 – M4GP final classification results on CIFAR-10	47
Table 6.4 – Comparison of CNN 1’s intermediate data on CIFAR-10 (Janke et al., 2019).....	51
Table 6.5 – Comparison of CNN 2’s intermediate data on CIFAR-10 (Janke et al., 2019).....	51
Table 6.6 – Comparison of VGG-16’s intermediate data on CIFAR-10 (Janke et al., 2019)	52

LIST OF ABBREVIATIONS AND ACRONYMS

ADB	AdaBoost
AI	Artificial Intelligence
CGP	Cartesian Genetic Programming
CNN	Convolutional Neural Network
CV	Computer Vision
DL	Deep Learning
FA	Factor Analysis
GA	Genetic Algorithm
GBC	Gradient Boosting
GP	Genetic Programming
HSI	Hyperspectral Image
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
KNN	K-Nearest Neighbors
LLE	Locally Linear Embedding
LR	Logistic Regression
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NAS	Neural Architecture Search
PCA	Principal Component Analysis
RFE	Random Forest
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
XGB	XGBoost

1. INTRODUCTION

1.1. COMPUTER VISION

Computer Vision (CV) is a sub-field of Artificial Intelligence (AI) that started in the early 1970s and it provides a visual perception component to computers, mimicking human intelligence and providing them with intelligent behavior (Szeliski, 2021). Last years have seen incredible advances in the performance and reliability of CV algorithms, brought in part by the shift to Machine Learning (and more specifically Deep Learning) combined with the training on very large sets of real-world data. Moreover, there are various tasks in CV, the most common being image classification, with others shown in Figure 1.1, including semantic segmentation, classification with localization, object detection, and instance segmentation.

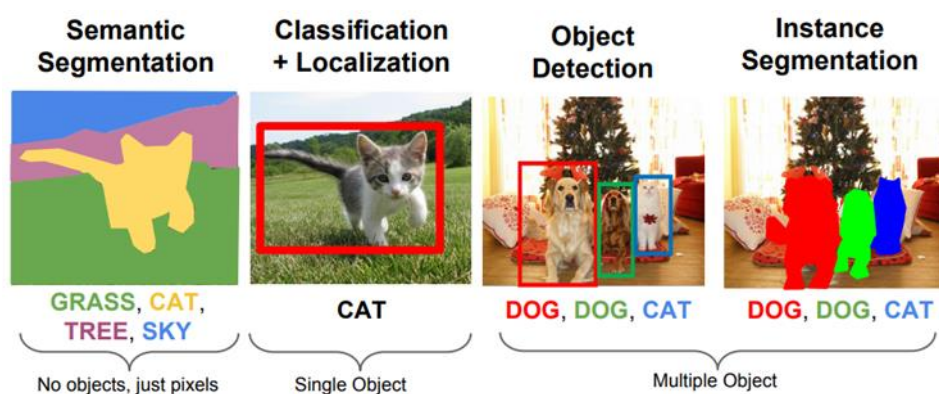


Figure 1.1 – Other CV tasks, from Li et al. (2017)

This document will focus on algorithms applied to image classification, in which the goal is to assign the input image a label from a fixed set of categories (Konushin & Artemov, 2021).

Convolutional Neural Networks (CNNs) have been tremendously successful in practical applications and have played an important role in the history of Deep Learning (DL), namely in the task of image classification (Goodfellow et al., 2016). CNNs were also some of the first neural networks to solve important commercial applications and remain at the forefront of commercial applications of DL today. The first published CNN (i.e., LeNet-5) was developed by LeCun et al. (1998). LeNet-5 was applied by a large telecommunications company in alphanumeric character recognition, to read checks automatically. Its architecture is shown in Figure 1.2.

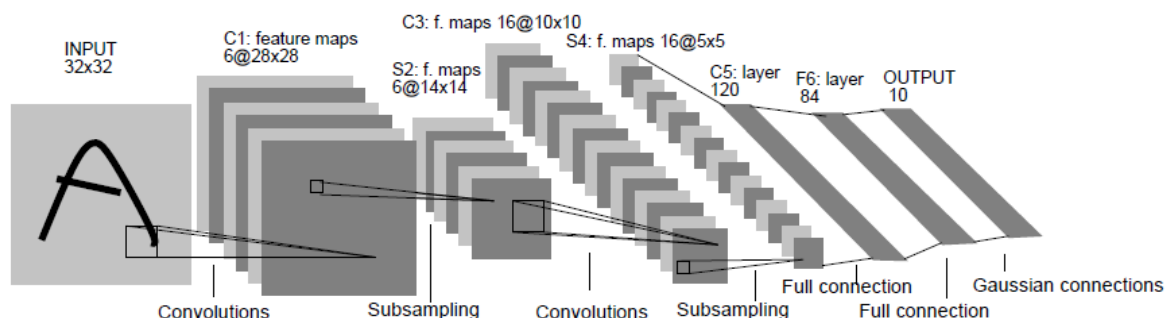


Figure 1.2 – Architecture of the first published CNN, LeNet-5, from LeCun et al. (1998)

CNNs were also used to win many contests. The current intensity of interest in DL began when Krizhevsky et al. (2012) won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), one of the most notorious object recognition challenges. At the time, AlexNet outperformed traditional approaches in the classification and localization tasks.

CNNs are, in fact, a specialized kind of neural network for processing data that has a known, grid-like topology, such as image data, which can be thought of as a 2D grid of pixels (Goodfellow et al., 2016). The name “Convolutional Neural Network” indicates that the network employs a mathematical operation called convolution, which is a specialized kind of linear operation used in place of general matrix multiplication.

1.2. GENETIC PROGRAMMING

Genetic Programming (GP), on the other hand, is a field of Computer Science that was popularized in the early 1990s (Willis et al., 1997). GP began as an attempt to discover how computers could solve problems without being explicitly programmed to do so, using an evolutionary algorithm very similar to genetic algorithms.

Koza (1992, 1994) was largely responsible for the popularization of GP. His algorithm was applied to symbolic regression, control, robotics, games, and classification (Willis et al., 1997). To solve a problem using GP, Koza (1992) states that it is necessary to specify the following (Willis et al., 1997):

- Terminal set: A set of input variables or constants
- Function set: A set of domain-specific functions used in conjunction with the terminal set to construct potential solutions to a given problem. This could consist of a set of basic mathematical functions or Boolean and conditional operators
- Fitness function: Fitness is a numeric value assigned to each member of the population to provide a measure of the appropriateness of a solution to the problem in question
- Control parameters: This includes the population size and the crossover and mutation probabilities
- Termination criterion: This is generally a predefined number of generations or an error tolerance on the fitness

It should be noted that the first three components determine the algorithm search space, while the final two components affect the quality and speed of the search.

Furthermore, the GP paradigm breeds computer programs to solve problems by executing the following three steps (Koza, 1992):

1. Generate an initial population of random compositions of the functions and terminals of the problem (i.e., computer programs)
2. Iteratively perform the following sub-steps until the termination criterion has been satisfied:
 - a. Execute each program in the population and assign it a fitness value according to how well it solves the problem

- b. Create a new population of computer programs by applying the following two primary operations. The operations are applied to computer programs in the population chosen with a probability based on fitness
 - i. Copy existing computer programs to the new population (also, mutation can be done instead of reproduction)
 - ii. Create new computer programs by genetically recombining randomly chosen parts of two existing programs (crossover)
 3. The best computer program that appeared in any generation (i.e., the “best-so-far” individual) is designated as the result of GP. This result may be a solution (or an approximate solution) to the problem

Figure 1.3 has a visual representation of the process described above, which is performed by evolutionary algorithms, including GP.

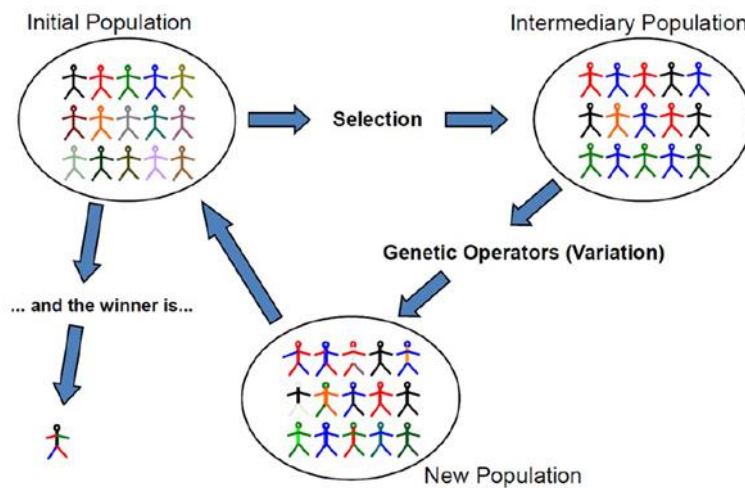


Figure 1.3 – Workflow of GP, from Bakurov (2018)

This thesis will focus on M4GP, which is a classification algorithm presented by La Cava et al. (2019) that learns multidimensional feature transformations using GP. M4GP uses a stack-based program representation that simplifies the construction of multidimensional solutions if compared to a tree-based representation, making its evolutionary process efficient and the final solutions more expressive, readable, and easy to understand (La Cava et al., 2019). It also incorporates a multi-objective parent selection and survival technique. Moreover, an archiving strategy was introduced to maintain a set of optimal trade-off solutions based on complexity and accuracy; the final model is selected from this archive using an internal validation set to reduce overfitting. An in-depth explanation of this algorithm will be given in Chapter 3.

1.3. PURPOSE OF THE THESIS

As Janke et al. (2019) said in the Conclusion of their paper, “the set of benchmark classification algorithms can be enlarged to include more classification algorithms to evaluate their predictive capability on high-level image features, i.e., the benchmark in this paper did not look at evolutionary algorithms”. Therefore, there is a gap in the current literature that justifies the research topic of this

document: the classification of high-level image features from convolutional layers with an evolutionary algorithm.

The research question for this thesis is:

“To what extent does the substitution of the fully-connected layer in Convolutional Neural Networks for an evolutionary algorithm affect the performance of those CNN models?”

Also, this research has five main objectives. First, to conduct a literature review that scientifically justifies the research topic. Second, to analyze the proficiency of fully-connected neural networks in the classification of digital images. Third, to assess if the usage of a state-of-the-art evolutionary algorithm for multiclass classification like M4GP improves the performance of CNNs. Fourth, to determine if the implementation of M4GP improves the performance of CNNs in specific cases and not in others, and to formulate in which situations it may (or may not) be beneficial to opt for M4GP. And fifth, finally, to ensure the generalizability of this work and the possibility of comparing the achieved results with previous (and future) research.

There will be five main methods to achieve all research objectives. First, to highlight the research gap found in the literature. Second, to swap the fully-connected layer in CNNs by an evolutionary algorithm, but only after the network trains, to enhance the process. Third, to benchmark the M4GP algorithm within Python, to reach the appropriate conclusions regarding this possible architecture. Fourth, use secondary data with different characteristics, to better benchmark this implementation and to carefully investigate the different situations and highlight them. Fifth, finally, to use only secondary data from well-known datasets in the field of CV (e.g., CIFAR).

The tested CNN architecture might yield better performance when compared to a vanilla CNN, as M4GP became one of the state-of-the-art GP-based algorithms for multiclass classification. Thus, there is a possibility that the substitution of the fully-connected layer in CNNs for M4GP will positively affect their performance. Nonetheless, one of the most important objectives of this research is to accurately describe in which conditions it might be beneficial to adopt the novel CNN architecture.

This thesis is intended to contribute to science by filling the gap in the current literature referred to above. By analyzing the results (described in Chapter 6), one can conclude if it might be beneficial to experiment using an evolutionary algorithm to classify high-level image features from convolutional layers, either in the context of a work project or concerning a novel research topic.

1.4. OUTLINE OF THE THESIS

The thesis has 9 chapters. Chapter 2 introduces concepts related to artificial neural networks, CNNs and it also presents some of the most important and well-known CNN architectures in the field of image classification¹. Chapter 3 presents four GP-based classification algorithms, among which M4GP. Chapter 4 highlights existing research projects that created GP approaches for Computer Vision. Chapter 5 describes the methodology choices to test the research question. Chapter 6 displays the results of this thesis, while also comparing them to some of the literature covered in the literature

¹ Still, to follow this thesis it is necessary to have prior knowledge of CNNs and Machine Learning

review. Chapter 7 has the conclusions. Chapter 8 describes limitations and avenues for future research. Finally, Chapter 9 cites the sources of information used in this thesis.

2. ARTIFICIAL NEURAL NETWORKS IN COMPUTER VISION

“Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples. Algorithms such as backpropagation use gradient descent to tune network parameters to best fit a training set of input-output pairs. ANN learning is robust to errors in the training data and has been successfully applied to problems such as interpreting visual scenes, speech recognition, and learning robot control strategies” (Mitchell, 1997). In fact, for certain types of problems (including image classification in the field of Computer Vision), ANNs are among the most effective algorithms.

In this Chapter, artificial neural networks will be described in 2.1. In 2.2, CNNs and some CV challenges will be briefly explained. Finally, in 2.3, some of the most important CNN architectures from past years will be highlighted, including AlexNet, ZFNet, GoogLeNet, VGG-16, and ResNet.

2.1. ARTIFICIAL NEURAL NETWORKS

There is a biological motivation for ANNs, as their study has been inspired in part by the observation that biological learning systems are built on very complex webs of interconnected neurons. The human brain, for example, is estimated to contain a densely interconnected network of approximately 10^{11} neurons, each connected, on average, to 10^4 others. Neuron activity is typically excited or inhibited through connections to other neurons (Mitchell, 1997).

In the late 1950s, Rosenblatt (1958) developed the perceptron, which is frequently regarded as the first modern ANN and one of the foundations for AI. Neural networks have had a tremendous evolution over the last decades, as they went all the way from Rosenblatt’s perceptron to DeepMind’s AlphaGo (Silver et al., 2016), a computer program that used deep neural networks and tree search to beat Lee Sedol in Go², who at the time was considered the best player in the world.

2.1.1. Perceptron

The perceptron is a single-layer neural network. The author inspired himself by the way neurons work together in the human brain (Rosenblatt, 1958). It is a binary classifier, so it is a function that decides whether an input (i.e., a vector of numbers) belongs to some specific class. The algorithm makes a prediction (to classify input) and if it is wrong, tweaks itself to make a more informed prediction next time, while it becomes more accurate over thousands or millions of iterations. Figure 2.1 illustrates the perceptron.

² Go is one of the oldest board games in the world; it is popular in China, Korea, and especially Japan

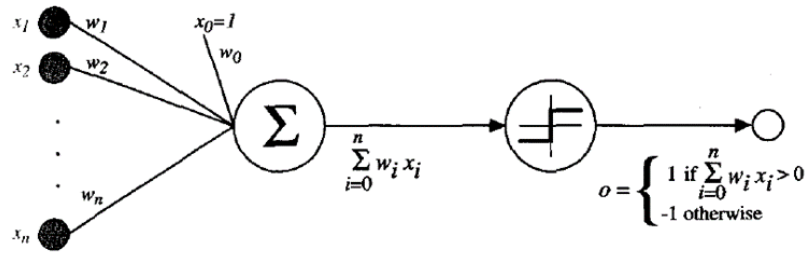


Figure 2.1 – Perceptron, from Mitchell (1997)

From the figure above, we can visualize that: $\{x_0 \dots x_n\}$ are the inputs of the model, $\{w_1 \dots w_n\}$ are the weights, w_0 is the bias, Σ represents a summation (weighted sum of inputs), the chart inside the circle represents the activation function (converts inputs to outputs) and the empty circle represents the outputs. Topics related to activation functions and the way to find values for the weights will be discussed more deeply in 2.1.2.

After a perceptron takes a vector of real-valued inputs, it outputs:

$$\begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases} \quad (2.1)$$

Learning a perceptron involves choosing values for the weights $\{w_0 \dots w_n\}$ (Mitchell, 1997). Also, we can view the perceptron as representing a hyperplane decision surface in the n -dimensional space of points. The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side. The equation for this decision hyperplane is:

$$\sum_{i=0}^n w_i x_i = 0, \text{ with } x_0 = 1 \quad (2.2)$$

There are some sets of positive and negative examples that cannot be separated by any hyperplane. Those that can be separated are called linearly separable sets.

Figure 2.2 illustrates the representational power of a perceptron.

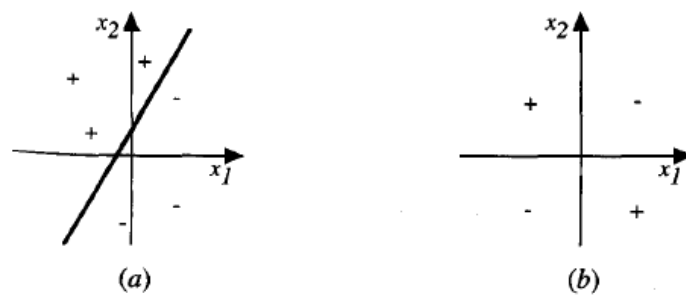


Figure 2.2 – The decision surface is represented by a two-input perceptron, from Mitchell (1997). (a) A set of training examples and the decision surface of a perceptron that classifies them correctly (b) A set of training examples that is not linearly separable. x_1 and x_2 are the perceptron inputs. Positive examples are indicated by "+", negative by "-"

2.1.2. Multilayer Perceptron

A multilayer perceptron (MLP) is a perceptron with two or more trainable weight layers, thus being more powerful than a single perceptron (Kriesel, 2007). MLPs have the advantage of being able to represent every Boolean function.

Neural network architectures have several parameters and properties which will now be described.

2.1.2.1. Layers

Apart from the input and output layers, MLPs also have one or more hidden layers, as can be seen in Figure 2.3.

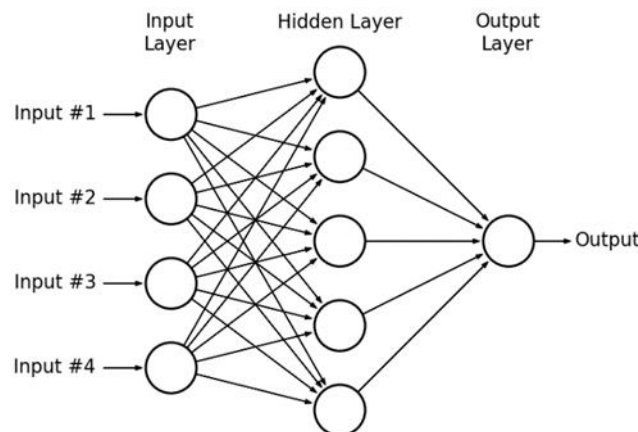


Figure 2.3 – Three-layer MLP, from Mohamed et al. (2015)

The three-layer MLP in the figure has one input layer (which in this case can be defined as a one-dimensional array of size 4), one hidden layer with 5 neurons, and one output layer of 1 neuron.

2.1.2.2. Learning

Backpropagation is a training algorithm that allows the adjustment of the weights in a multi-layer feedforward neural network. This is where weights in the network are tweaked over several iterations to minimize the error between the value predicted by the network and the true value. Generally, more weights lead to more flexibility in predicting complex patterns.

One of the main learning algorithms used to adjust weights and minimize error is Stochastic Gradient Descent (SGD).

Thus, neural networks are generally trained (i.e., optimized) with SGD, using backpropagation as a gradient computing technique.

2.1.2.3. Error and Error Surface

The error is calculated as the difference between the output and target, while the error surface is the change in the error for different weights (illustrated in Figure 2.4).

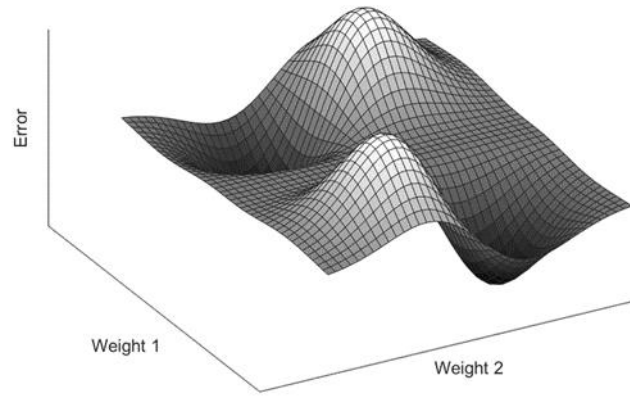


Figure 2.4 – Error surface with weight space and a global minimum, from Iman et al. (2021)

2.1.2.4. Activation Functions

The activation function³ transforms the network input, as well as the previous activation state, into a new activation state (Kriesel, 2007). It is often defined globally for all neurons or at least for a set of neurons and only the threshold values are different for each neuron.

Essentially, an activation function transforms inputs into outputs, and it is needed for the introduction of nonlinearity into the network.

The simplest activation function is the threshold function, which can only take on two values: if the input is above a certain threshold, the function changes from one value to another, but otherwise, it remains constant (Kriesel, 2007). This implies that the function is not differentiable at the threshold and for the rest, the derivative is 0. It is generally represented as:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \quad (2.3)$$

Another two options are the sigmoid function⁴, which maps to the range of values (0, 1), and the hyperbolic tangent which maps to (-1, 1) (Kriesel, 2007). The sigmoid function is represented as:

$$f(x) = \frac{1}{1 + e^x} \quad (2.4)$$

And the hyperbolic tangent as:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.5)$$

The three previously discussed functions can be visualized in Figure 2.5.

³ Also called transfer function

⁴ Also called logistic function

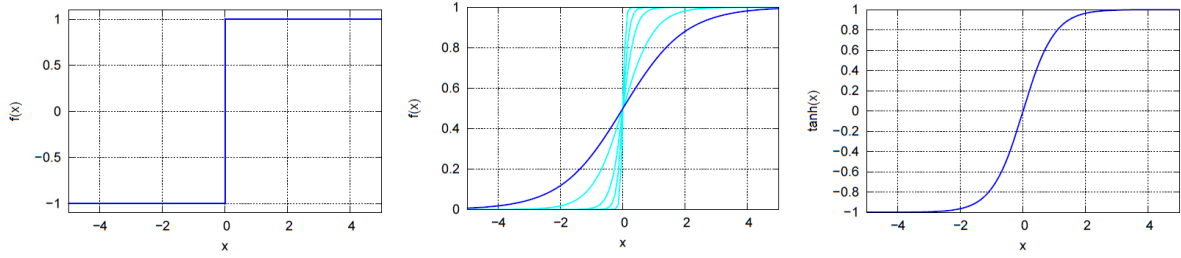


Figure 2.5 – Three of the most popular activation functions for MLPs, left-to-right: threshold function, sigmoid function, and hyperbolic tangent; from Kriesel (2007)

2.1.2.5. Loss Function

Typically, when training neural networks, we seek to minimize the error. As such, the objective function is often referred to as the loss function and the value calculated by the loss function is referred to as simply loss (Goodfellow et al., 2016).

In fact, “the cost function reduces all the various good and bad aspects of a possibly complex system down to a single number, a scalar value, which allows candidate solutions to be ranked and compared” (Reed & Marks, 1999).

The two main types of loss functions are the mean squared error (MSE) and cross-entropy.

MSE is commonly used in regression problems where a quantity is being predicted, and it can be viewed as the cross-entropy between the distribution of the model predictions and the distribution of the target variable (Goodfellow et al., 2016). Its formula is:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \tilde{y}_i)^2 \quad (2.6)$$

Where n is the number of sample data points, y represents the actual values and \tilde{y} the predicted values.

The cross-entropy loss function is often used for classification problems when outputs are interpreted as probabilities of membership in an indicated class (Reed & Marks, 1999). Its formula for multiclass classification problems is:

$$Loss = -\sum_{i=1}^{output\ size} y_i \times \log \hat{y}_i \quad (2.7)$$

2.1.2.6. Epochs

The epochs refer to the number of iterations of backpropagation to be performed over the network.

2.2. CONVOLUTIONAL NEURAL NETWORKS FOR IMAGE CLASSIFICATION

“ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras” (Mitchell, 1997).

A CNN is a class of artificial neural networks applied in the field of CV, specifically for image classification, for example, to recognize everyday objects, humans, and animals. The idea of CNNs was popularized by LeCun et al. (1998), where the LeNet-5 network was introduced for digit recognition.

When a computer takes an image as input, it only gets an array of pixel values. Depending on the resolution and size of the image, it will take, for example, a $32 \times 32 \times 3$ array of numbers (the 3 refers to RGB values, indicating it is a color image). Each of these numbers is given a value from 0 to 255, which describes the pixel intensity at that point. The idea is that the computer is fed an array of values to then output numbers that describe the probability of the image being a certain class. The general structure of a CNN can be seen in Figure 2.6.

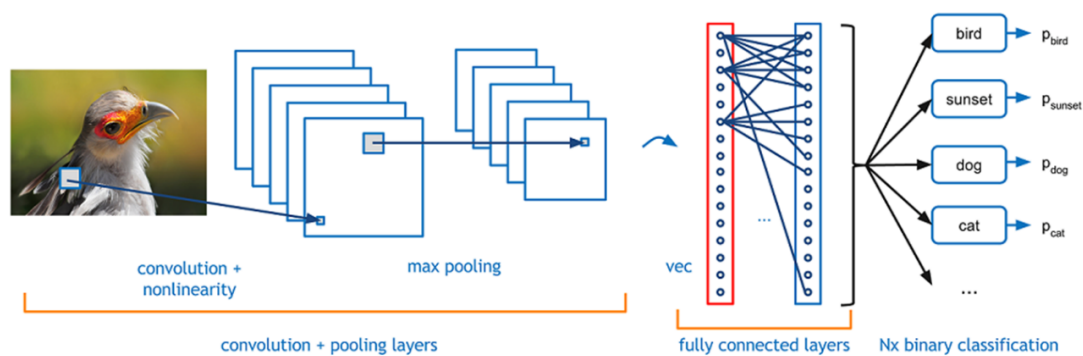


Figure 2.6 – Main operations in a CNN – convolution, nonlinearity, pooling, and classification; from Kaplanoglou (2017)

A CNN can perform image classification by looking for low-level features such as edges and curves, to then build up to more abstract concepts through a series of convolutional layers. The most crucial component of this type of deep network is, in fact, the use of trainable multi-layer convolutions (Szeliski, 2021).

Instead of connecting all the units in a layer to all the units in a preceding layer, the convolution operation leads CNNs to organize each layer into feature maps (Szeliski, 2021). In convolutional network terminology, the first argument to the convolution is often referred to as the input and the second argument as the kernel or feature detector, while the output is usually referred to as the feature map or activation map. The input is usually a multidimensional array of data, and the kernel is usually a multidimensional array of parameters (i.e., tensors) that are adapted by the learning algorithm. Feature maps can be considered parallel planes or channels.

In the convolution step, a filter slides over the input image to produce a feature map. The convolution of another filter over the same image gives a different feature map, so different filters generate different feature maps from the same original image. Thus, the greater the number of filters, the more image features get extracted and the better the network becomes at recognizing patterns in unseen images. It is also important to note that the convolution operation captures the local patterns in the original image (like edges, curves, or textures). Figure 2.7 illustrates the process of learning local patterns in CNNs.

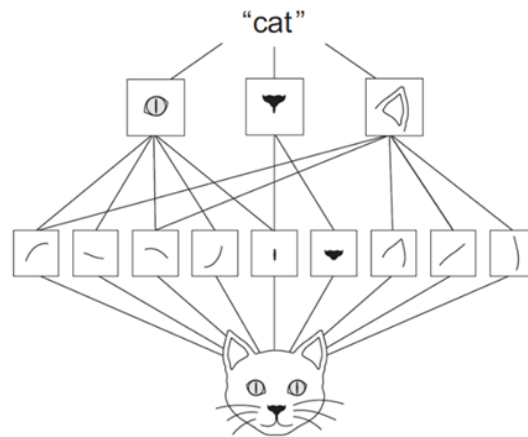


Figure 2.7 – The convolution operation – in this example, hyperlocal edges combine into local objects such as eyes or ears, which combine into the high-level concept of “cat”; from Chollet (2017)

In conclusion, the main purpose of convolution is to extract features from the input image while preserving the spatial relationship between pixels by learning image features using small squares of input data.

2.2.1. Computer Vision Challenges

In the field of Computer Vision, CNNs or other image classification algorithms might face challenging hurdles. These challenges are (Grauman & Leibe, 2011):

- Occlusion
- Scale Variation
- Deformation
- Background Clutter
- Over-illumination or Lack of Illumination
- Viewpoint Variation
- Object Pose

These are associated with difficult scene conditions (Figure 2.8). Instances of the same image class can generate very different images, depending on confounding variables like partial occlusions, unrelated background clutter, illumination conditions, camera viewpoint, and object pose. Different instances of objects from the same class can also exhibit significant variations in appearance.

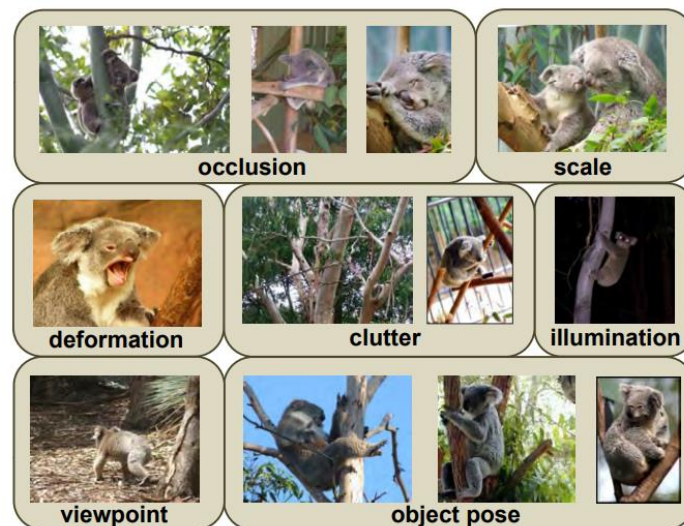


Figure 2.8 – Visual understanding of Computer Vision challenges, from Grauman & Leibe (2011)

Another challenge is the within-class variation. The concepts represented by image classes can be too broad, leading to several within-class variations. Figure 2.9 exemplifies how the class “chair” in an image dataset might have challenging differences in its instances.



Figure 2.9 – Example of within-class variation, from Fei-Fei et al. (2005)

Hence, image classification methods must be robust to the challenging conditions described above. Some image preprocessing techniques can be used to surpass them, even though the “within-class variation” challenge might be hard to surpass just by applying the referred techniques.

2.3. EVOLUTION OF CONVOLUTIONAL NEURAL NETWORK ARCHITECTURES

As previously said, LeCun et al. (1998) created the LeNet-5 network for digit recognition, the first published CNN, which was already discussed in Chapter 1. Still, it was only in 2012 that CNNs grew to prominence when AlexNet (Krizhevsky et al., 2012) won the ILSVRC-2012 competition in the classification and localization tasks.

ILSVRC has become, in fact, the standard benchmark for large-scale object recognition. The dataset used in the competitions comes from ImageNet, which is a dataset of over 15 million labeled high-

resolution images belonging to roughly 22 000 categories (Krizhevsky et al., 2012). The images were collected from the web and labeled by human labelers. ILSVRC uses a subset of ImageNet with roughly 1 000 images in each of the 1 000 categories. Overall, there are roughly 1.2 million training images, 50 000 validation images, and 150 000 testing images.

The competition has been active since 2010. In past years, several CNN architectures have won it, each one representing the state-of-the-art of that time. Some of the most important ones will be presented.

In 2012, AlexNet won the classification task with a top-5 test error rate of 15.3%, while the second-best entry achieved 26.2%.

Its architecture is similar to the one from LeNet-5: it contains 8 layers, where 5 are convolutional layers and 3 are fully-connected layers. It used the ReLU activation function to add nonlinearity and improve the convergence rate. Also, its 60 million parameters made AlexNet susceptible to overfitting, so the creators adopted a method of data augmentation and dropouts to avoid it. Moreover, AlexNet made use of overlapping pooling to reduce the error. The architecture is illustrated in Figure 2.10.

Figure 2.10 – AlexNet architecture, with a delineation of responsibilities defined for training on two GPUs, from Krizhevsky et al. (2012)

2.3.2. ZFNet

ZFNet's architecture is identical to AlexNet's (Figure 2.11). ZFNet is considered an extended version of AlexNet, with some modifications to filter size to achieve better accuracy. The idea was to use smaller filters in the convolutional layer, thus avoiding the loss of pixel information, even though the network was not able to decrease its computational cost.

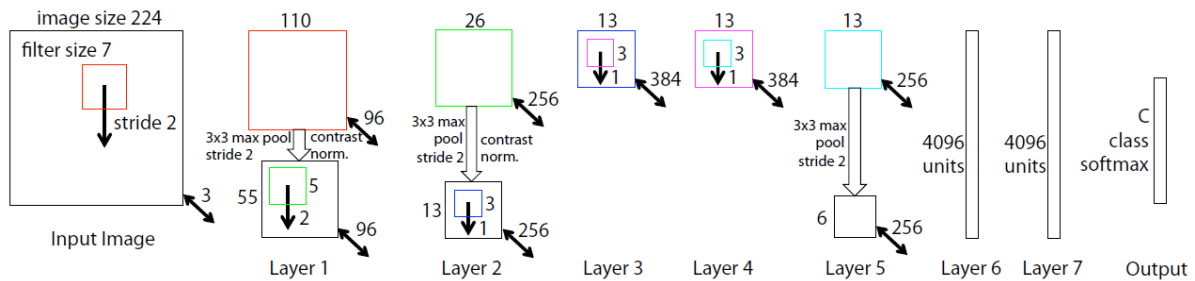


Figure 2.11 – Architecture of ZFNet, an 8-layer CNN model, from Zeiler & Fergus (2014)

It is important to note that from 2013 on, the ILSVRC training dataset remained the same, so comparisons can be made.

2.3.3. GoogLeNet

ILSVRC-2014 had two contenders that presented significant changes to the previous CNNs' structure. The first highlight goes to GoogLeNet (Szegedy et al., 2015), the winner, which achieved a top-5 error of 6.67%, showing deeper CNN architectures can reach better performance. It is a 22-layer network without fully-connected layers at its end and its main contribution was the Inception module. Thus, GoogLeNet is also known as Inception-v1.

This was an important milestone in the development of CNNs, since in previous years the most popular CNNs just stacked convolution layers deeper and deeper, trying to get better performance. The Inception module was, however, engineered to improve performance, both in terms of accuracy and speed.

The premise for creating this new architecture was the large variation in the location of information in an image. As an example, an object may occupy most of an image, a part of it, or very little space. Because of this, choosing the right kernel size for the convolution operation becomes difficult.

The solution brought by the Inception architecture (Figure 2.12) was to have filters with multiple sizes operating on the same level, essentially increasing the wideness of the network instead of its deepness. In the naïve Inception module, convolution is performed on input with three different sizes of filters (1×1 , 3×3 , 5×5). Additionally, max-pooling is performed. The outputs are concatenated and sent to the next Inception module.

To make the deep neural network less computationally expensive, the authors limited the number of input channels by adding an extra 1×1 convolution before the convolutions with greater spatial sizes.

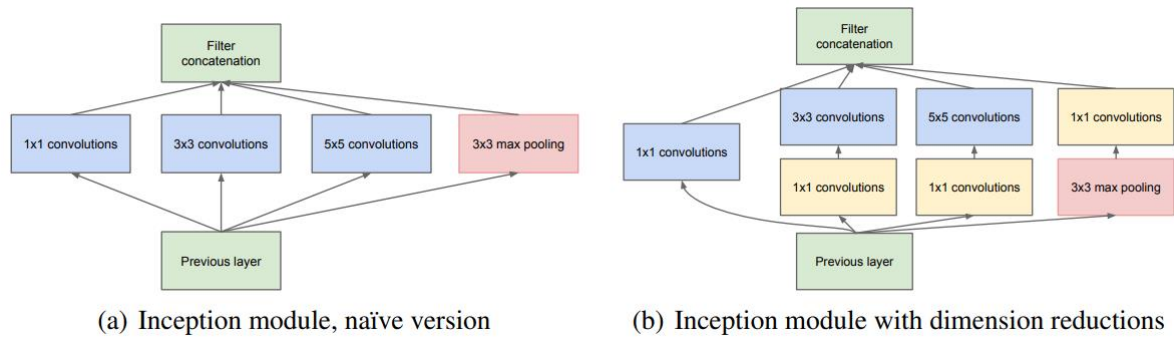


Figure 2.12 – Inception module, from Szegedy et al. (2015)

GoogLeNet’s architecture (Figure 2.13) starts with the input layer and the first layers in the network. Then, three stacks of Inception modules are found before a global average pooling layer, a fully-connected layer with dropout (40%), and the softmax output layer.

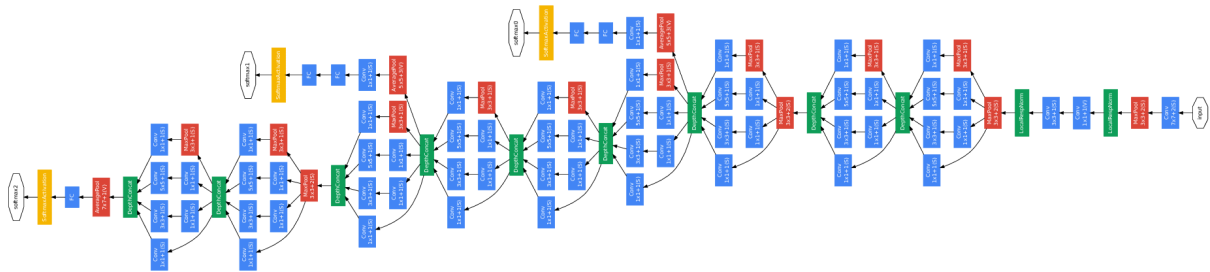


Figure 2.13 – GoogLeNet network, from Szegedy et al. (2015)

If we consider GoogLeNet as a deep network that gradually increases its width, its geometry can be imagined as an inverted pyramid.

2.3.4. VGG-16

VGG-16 (Simonyan & Zisserman, 2014), the main runner-up on ILSVRC-2014, reached a top-5 error of 7.3%.

Despite not winning the competition in 2014 (as said previously, GoogLeNet was the winner), the main contribution of the Visual Geometry Group (VGG) was to increase the network’s depth using an architecture with very small (3×3) convolution filters.

The architecture depicted in Figure 2.14 is VGG-16. The input to the first layer (conv1) is a fixed-size 224×224 RGB image. The image is passed through a stack of convolutional layers, where filters with a very small receptive field (3×3) were used. One of the configurations also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel. Spatial pooling is carried out by five max-pooling layers, which follow some of the convolutional layers. Max-pooling is performed over a 2×2 pixel window, with stride 2. Finally, three fully-connected layers follow a stack of convolutional layers, and the last layer is the softmax layer. Also, all hidden layers have ReLU non-linearity.

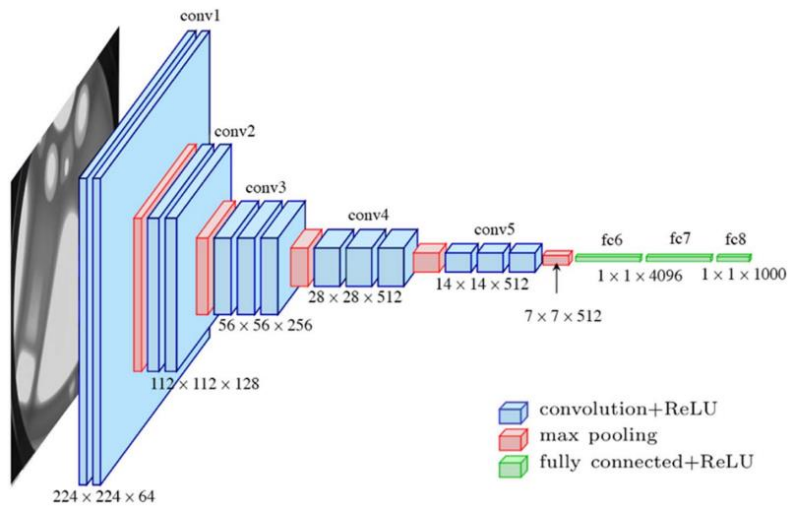


Figure 2.14 – VGG-16 network architecture, from Ferguson et al. (2017)

VGG’s results confirmed the importance of depth in visual representations.

It is also relevant to mention the difference between VGG-16 and VGG-19, another well-known model. The concept of the VGG-19 model is the same as the VGG-16, except that it is deeper and supports 19 layers. The “16” and “19” refer to the number of layers that have weights in the architecture, meaning VGG-19 has 3 more layers than VGG-16.

2.3.5. ResNet

ResNet (He et al., 2016) was the winner on ILSVRC-2015 with a top-5 test error rate of 3.57%, turning into the first model to beat human performance. Models in the past had already used numerous hidden layers and extremely deep neural networks, although those led to a problem of vanishing or exploding gradients. To counter this problem, Deep Residual Networks (i.e., ResNets) use residual blocks and skip connections to significantly increase the count of hidden layers. Essentially, the networks preserve the information that was obtained through learning in the form of residuals.

Adding more layers to a model leads its accuracy to saturate, then rapidly decay. It also generates higher training errors. This is the degradation problem, and it is not caused by overfitting. Thus, due to the problem of vanishing or exploding gradients, learning better networks is not as easy as stacking more layers (Figure 2.15).

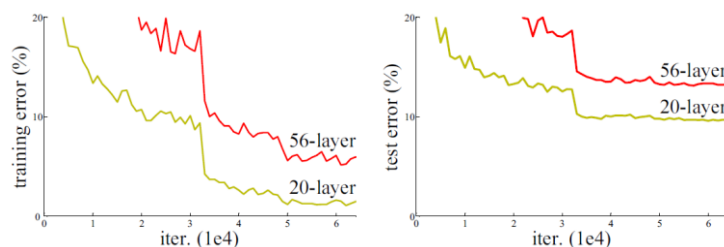


Figure 2.15 – Training error (left) and test error (right) on the CIFAR-10 dataset with 20-layer and 56-layer “plain” networks: the deeper network has higher training error, and thus test error; from He et al. (2016)

A residual block is a stack of layers set in such a way that the output of a layer is taken and added to another layer deeper in the block (see Figure 2.16). Instead of learning the output $H(x)$ of a given input x , the network learns $H(x)$ as a function:

$$H(x) = F(x) + x \quad (2.8)$$

Therefore, the network is only required to fit the residual:

$$F(x) = H(x) - x \quad (2.9)$$

This is smaller and easier to learn than the whole representation of $H(x)$ (Janke et al., 2019). This change helps the gradient flow between layers. A ResNet is formed by stacking several of these residual blocks together.

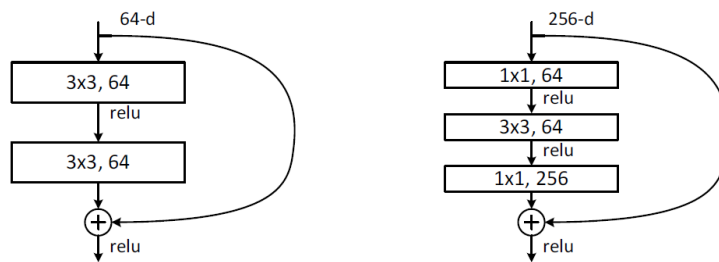


Figure 2.16 – Two types of residual blocks, from He et al. (2016)

Residual blocks enabled the evolution from a 22-layer CNN, such as GoogLeNet, to a very deep neural network like ResNet-152, which has 152 layers.

The best Residual Network's architecture included 152 layers with no dropout, a weight decay of 0.0001, batch normalization after each convolution, and the ReLU activation function. After the last residual block, there is a global average pooling layer, followed by a fully-connected layer. A ResNet is depicted in Figure 2.17.

Even though the depth of a ResNet could increase drastically, the number of needed parameters was still much lower if compared to those calculated for a corresponding VGG architecture.

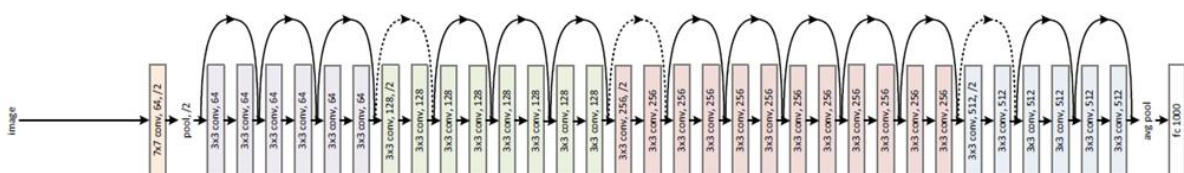


Figure 2.17 – A Residual Network with 34 parameter layers, from He et al. (2016)

Ultimately, residual blocks were the solution to train deeper networks.

3. MULTICLASS CLASSIFICATION WITH GENETIC PROGRAMMING

Several works about GP-based classification algorithms have been presented in the last two decades, although they have had poor performance when compared to state-of-the-art classifiers. However, M2GP was an exception which got introduced by Ingalalli et al. (2014). This multidimensional GP approach revealed competitive results in multiclass classification problems.

Classification problems are very important in the field of Machine Learning for various types of applications. Nevertheless, GP was never considered a state-of-the-art method for multiclass classification problems with a high number of classes, which was changed by M2GP.

The main goal of this chapter is to discuss M4GP, which is the newest algorithm from a “family” that includes three previous iterations: M2GP (3.1), M3GP (3.2), and eM3GP (3.3). M4GP, explained in 3.4, is the most relevant in our context for two reasons; first, because it is currently the state-of-the-art evolutionary algorithm for multiclass classification, and second, because it will be an integral part of this thesis. Nonetheless, its previous iterations will also be briefly described.

3.1. M2GP: A MULTIDIMENSIONAL GP APPROACH FOR MULTICLASS CLASSIFICATION PROBLEMS

Essentially, M2GP is a tree-based GP algorithm that finds “a transformation, such that the transformed data of each class can be grouped into unique clusters” (Silva et al., 2016). Still, the number of dimensions in which the clustering is performed is completely independent of the number of classes.

This method has two phases: training and testing (Ingalalli et al., 2014).

3.1.1. Training Phase

One of the first steps is to represent solutions similarly to regular tree-based GP, although the root node in the tree exists only to define the number of dimensions (d) for a new space, as the algorithm represents the input sample in a d -dimensional solution space (Silva et al., 2016).

The genetic operators are the regular subtree crossover and mutation, with the exception being that the root cannot be chosen as a crossing or mutation node. Also, fitness is based on classification accuracy.

The individuals can be evaluated after the d -dimensional solution space is mapped. Each individual is evaluated through the following process:

1. For each of the M classes in the data, the covariance matrix and the cluster centroid are calculated from the samples belonging to that class
2. The Mahalanobis distance between each sample and each of the M centroids is calculated
3. Each sample is then assigned the class whose centroid is closer

Figure 3.1 has a clustering example.

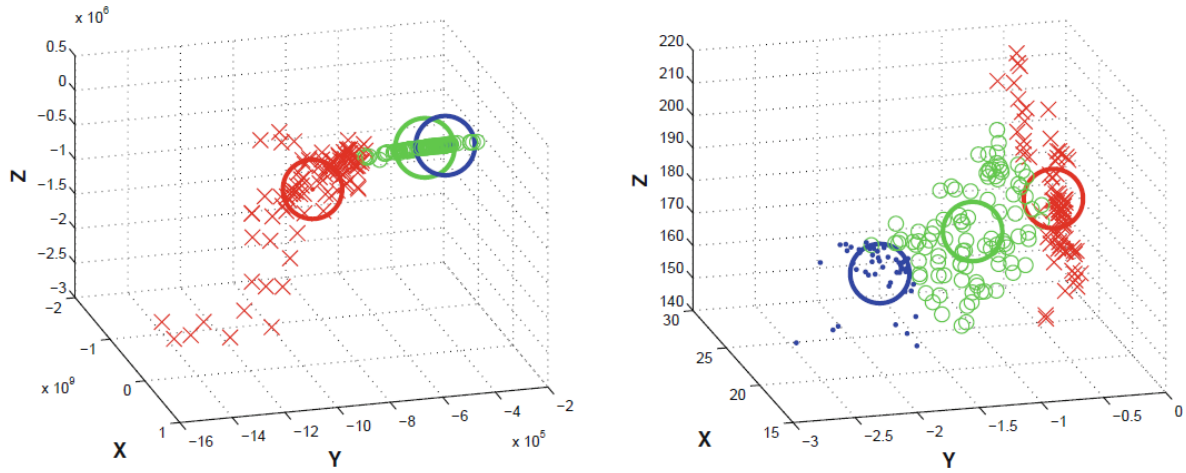


Figure 3.1 – On the left, the clustering is obtained by an individual with low accuracy, while the one on the right has very high accuracy, from Muñoz et al. (2015); the large circles represent the class centroids

3.1.2. Testing Phase

To classify unseen data, the tree is used to map the new samples into the new space, and then the covariance matrices and the cluster centroids are used to calculate the minimum Mahalanobis distance between each sample and each centroid (Silva et al., 2016).

3.1.3. Downside

The main downside of using M2GP is the fact the number of dimensions d is fixed at the beginning of each run, and that d may be the wrong number for a specific problem (Silva et al., 2016).

M2GP cannot add or remove dimensions during its evolution.

3.2. M3GP: M2GP WITH MULTIDIMENSIONAL POPULATIONS

M3GP (Muñoz et al., 2015) was created to tackle the greedy approach used by M2GP to find the number of dimensions the evolved solutions should have. Hence, M3GP evolves a population that may contain individuals of several different dimensions. This algorithm can be simplified as being “M2GP with multidimensional populations” (Muñoz et al., 2015).

This method will now be described in more detail, according to Muñoz et al. (2015).

3.2.1. Initial Population

The evolution starts with a random population where all the individuals only have one dimension, thus ensuring M3GP starts by looking for simpler solutions.

3.2.2. Mutation

This is the only way of adding or removing dimensions. One of three actions is executed with equal probability:

- A randomly created new tree replaces a randomly chosen branch of the parent tree (standard subtree mutation)
- A randomly created new tree is added as a new branch of the root node, effectively adding one dimension to the parent tree
- A complete branch of the root node is randomly removed, effectively removing one dimension from the parent tree

3.2.3. Crossover

One of two actions is executed with equal probability:

- A random node, excluding the root, is chosen in each of the parents and the respective branches are swapped (standard subtree crossover)
- Swapping of dimensions, where a random complete branch of the root node is chosen in each parent and swapped between each other, effectively swapping dimensions between the parents

3.2.4. Pruning

Although increasing the number of dimensions may be beneficial to the performance of this method, it may also degrade an individual's fitness. Thus, the best individual in each generation can get pruned, i.e., its detrimental dimensions may get removed.

The pruning procedure removes the first dimension and reevaluates the tree. If the fitness improves, the pruned tree replaces the original and goes through pruning of the next dimension. Otherwise, the pruned tree is discarded, and the original tree goes through pruning of the next dimension. The procedure stops after pruning the last dimension. This shifts the distribution of the number of dimensions to lower values, without harming fitness.

3.2.5. Elitism

The best individual of any generation is always copied to the next one⁵.

3.2.6. Downsides

There are three main downsides to using M3GP (Silva et al., 2016).

Firstly, the method assumes that a single transformation will simplify the classification problem for all the classes; however, this might not be the case. The optimal data transformation may be, in fact, class-dependent. Also, M3GP seemed to suffer from bloat at the dimension level. Finally, it seemed to suffer from overfitting.

⁵ This individual is already optimized, as it went through pruning

3.3. EM3GP: ENSEMBLE M3GP

To address M3GP's drawbacks, ensemble M3GP (or eM3GP for short) was proposed. In this algorithm, the classification is done using M different transformations, one for each class, in a multiclass problem with M classes (Silva et al., 2016).

The improvements included in eM3GP will now be discussed (Silva et al., 2016).

3.3.1. Specialized Class Transformations

First, we are interested in identifying the best transformation for each class and in building a set of specialized class transformations (S). At the beginning of the search, when the first individual is evaluated, M copies are stored in S (M represents the number of classes). For all subsequent transformations evolved during the search, we compute the number of True Positives (TP) and False Positives (FP) it produces for each class. If at least one of these numbers improves (TP gets higher or FP gets lower) and neither value deteriorates relative to the best transformations, then the new tree replaces the previous within S .

This is done for every individual evaluated during the run – Figure 3.2 illustrates this process.

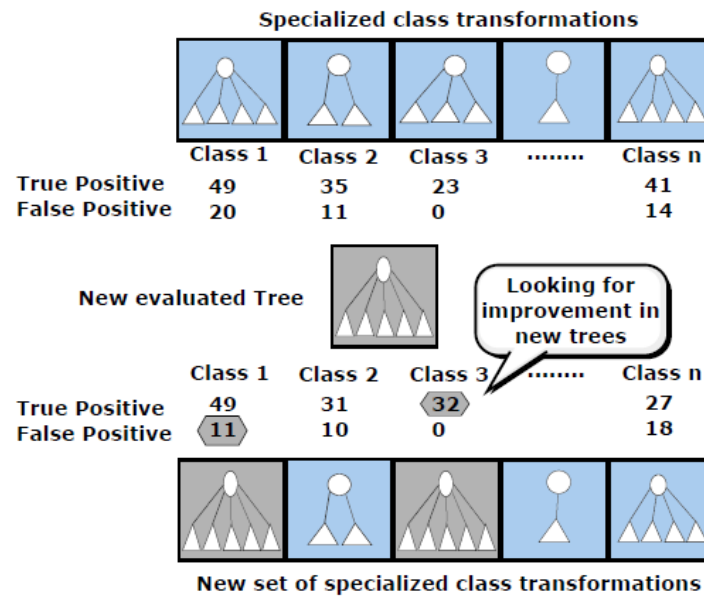


Figure 3.2 – Identification of the best transformation for each class based on TP and FP, from Silva et al. (2016)

3.3.2. Ensemble Classifier

This proposal builds an ensemble of transformations for each new individual, by combining it with the transformations stored in S . Each one represents a d -dimensional transformation, such that given a data point the transformation is used to compute the distance to the i^{th} class cluster. After computing the distance to each class cluster, the minimum distance determines the class label assigned.

3.3.3. Ensemble Construction

Finally, an ensemble E is constructed for each tree and used to assign fitness.

First, every element in the ensemble is set equal to k (k is each tree). Then, the accuracy of E is computed. Afterward, in a random order, we replace each element with its corresponding specialized transformation in S . After each replacement, we compute the accuracy of the ensemble E . If the accuracy improves then the change is kept, otherwise, it is reversed.

3.4. M4GP: THE STATE-OF-THE-ART ALGORITHM

M4GP is a significant improvement over its three predecessors, although it is inspired by them, thus their importance.

The main changes and improvements in M4GP include its program representation, parent selection, survival technique, and archiving strategy (La Cava et al., 2019).

3.4.1. Program Representation

Instead of the typical tree-based GP representation, M4GP uses a novel stack-based representation. M4GP's predecessors used special nodes to allow for multiple outputs at the root, which introduced unnecessary complexity. Thus, the state-of-the-art algorithm simplified the encoding.

Programs are evaluated by stack operations. For example, the program $\Phi(x) = [x_1, x_2, x_1^2, x_2^2, x_1x_2]$ can be constructed as follows:

$$i = [x_1 \ x_2 \ x_1 \ x_1 * \ x_2 \ x_2 * \ x_1 \ x_2 *]$$
 (3.1)

This is because the evaluation proceeds left to right, pushing and pulling instructions to and from a single stack. "Arguments such as x_1 are pushed to the stack, and operators such as $*$ pull arguments from the stack and push the result. At the end of a program's execution, the stack state is interpreted as the multi-dimensional transformation" (La Cava et al., 2019).

This program representation makes the evolutionary process of the algorithm more efficient and the final solutions more expressive, readable, and easy to understand.

3.4.2. Initialization

M4GP initializes a population of programs of various sizes and dimensionality. Each equation in a program is initialized recursively in an analogous fashion to the grow method but limited by the number of nodes rather than depth.

The grow method, used to generate the initial random population, encompasses growing trees that are variably shaped. "The length of a path between an endpoint and the root is no greater than the specified maximum depth" (Koza, 1992).

3.4.3. Selection

The authors tested three population selection methods: tournament selection, lexicase selection, and age-fitness Pareto selection.

3.4.3.1. Tournament Selection

Selects two individuals from the current population and choose the one with better fitness as a parent for the next generation.

3.4.3.2. Lexicase Selection

This technique rewards individuals in the population for performing well on unique combinations of fitness cases (i.e., samples). Each parent selection begins with the entire population in the selection pool. The fitness cases are shuffled randomly, and the first case is considered. Any programs in the pool that do not have exactly the best fitness in the first case are removed from the selection pool. If more than one individual remains in the pool, the removal process is repeated with the next case. This process continues with additional fitness cases until either:

- Only one individual remains in the selection pool, in which case that individual is selected as a parent
- No more fitness cases are left, in which case a parent is selected from the remaining pool

This process is repeated for each parent selection. The selected parents are Pareto-optimal concerning the fitness cases.

The test cases can be thought of as filters, and each parent selection represents a random path through these filters.

This selection promotes individuals with diverse performances observed in previous experiments.

3.4.3.3. Age-Fitness Pareto Selection

Each individual is assigned an age equal to the number of generations since its oldest ancestor appeared. In each generation, one new individual is added to the population to provide a small amount of random restart. Selection for variation is random rather than guided, and during breeding, several offspring equal to the overall population size is created. Afterward, the set P consisting of the current population and the newly created individuals is culled to the original population size, N , using their accuracy and their age. In this case, a lower age is preferred, and the fitness is inverted so that a lower fitness is also better.

Program survival is based on the environmental selection scheme of the Strength Pareto Evolutionary Algorithm 2 (SPEA2), described in Zitzler et al. (2001).

3.4.4. Variation

The mutation and crossover operators are the same as in M3GP, which was already explained in 3.2.2. and 3.2.3., respectively.

3.4.5. Classification Strategy

In M4GP, the nearest centroid classifier is the used strategy, although it is not the traditional approach in GP.

The nearest centroid classifier works by partitioning the attributes into subsets $\{X_1 \dots X_k\}$, such that X_k is the subset of x with class label k . To classify a new sample $x' \in \mathbb{R}^p$, the distance of x' to each subset $\{X_1 \dots X_k\}$ is measured and the class label corresponding to the minimum distance is assigned as:

$$\hat{y}(x') = j, \text{ where } j = \arg \min_k D(x', X_k), \quad k = 1, \dots, K \quad (3.2)$$

D is the Mahalanobis distance between x' and the centroid of X_k , which is defined as follows:

$$D_M(x', X_k) = \sqrt{(x' - \mu_k) \Sigma_k^{-1} (x' - \mu_k)^T} \quad (3.3)$$

With $\mu_k \in \mathbb{R}^p$ being the centroid of X_k and $\Sigma_k \in \mathbb{R}^{p \times p}$ the within-class covariance matrix.

This strategy assumes the within-class samples to be normally distributed about their centroid and separated from the distributions of other classes.

Two other classification methods were tested: ‘bool’ and ‘float’, which make use of M4GP’s stack-based system. The first approach, ‘bool’, includes Boolean operators (*AND, OR, NOT, >, <, ≥, ≤, =, IF – THEN, IF – THEN – ELSE*) to evolve classification rules and interpret the result as a bitstring for classification. The second one, ‘float’, evolves a set of mathematical expressions, with the classification being made according to the index that has the highest value in the stack output. For example, if $\Phi(x)$ from 3.4.1. produced the output [1.5, 3, 2.25, 9, 4.5], class 4 would be assigned to the sample.

3.4.6. Archiving Solutions

In GP, typically, the individual with the highest training accuracy is chosen as the final model, to then be evaluated on the test set. Still, given the importance of interpretability, an archive of solutions that represent the best trade-offs of training accuracy and complexity is maintained on subsequent problems.

To choose a final model, a small validation set is split from the training set before the run. At the end of the run, the archive is evaluated on the validation set, and the individual with the best score is selected as the final model. This approach should also reduce overfitting and produce simpler models. The entire archive can also be studied by experts to get a better sense of the building blocks of good solutions to their problems, which can offer important insights.

3.4.7. Discussion about M4GP

In contrast to M2GP and M3GP, M4GP uses a stack-based representation that removes the need for explicit root nodes, preserves multidimensionality, and allows dimensionality to change. Also, it has multi-objective selection strategies and a Pareto archive for model selection and better interpretability.

In La Cava et al. (2019), the authors prove the algorithm performs significantly better than its predecessors across all benchmark problems, according to a Friedman test of rankings for test accuracy. Furthermore, advanced evolutionary selection methods (like lexicase selection and age-

fitness Pareto survival), were proved to perform better than tournament selection in producing accurate classifiers.

All these features foster M4GP as the state-of-the-art multiclass classification method with GP.

4. IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS AND GENETIC PROGRAMMING

In the last decade, several authors have investigated the usage of DL (i.e., CNNs) to perform feature extraction, which is the first phase of image classification.

Chen et al. (2014) was one of the first studies to do it, although nowadays some design choices might seem obsolete. In this work, the authors proposed a hybrid approach to image classification. The framework combined Principal Component Analysis (PCA), a DL architecture, and Logistic Regression. Specifically, stacked autoencoders were implemented as a DL architecture to extract features. The experimental results revealed the proposed framework produced competitive results on well-known datasets.

Chen et al. (2016) was a study where the use of different feature extractors and classifiers was motivated by the lack of training data. In this paper, a deep feature extraction method was presented for hyperspectral image (HSI) classification using a CNN. To study the potential of a CNN to be a feature extractor, that network was compared with PCA, Factor Analysis (FA), and Locally Linear Embedding (LLE). Three deep feature extraction architectures based on CNNs were proposed to extract the spectral, spatial, and spectral-spatial features of HSI. The classification task is performed with a traditional fully-connected layer. Support Vector Machines, K-Nearest Neighbors, and Logistic Regression are also considered to evaluate the classification performance achieved by using PCA, FA, and LLE as feature extractors. The authors proved the CNN was able to effectively extract features, which led to good performance results.

In Ayinde et al. (2019), the authors wanted to better understand the features extracted by a CNN. Particularly, the authors showed how size, choice of activation function, and weight initialization impact redundant feature extraction of deep neural network models. The results attained on well-known datasets showed the wider and deeper a network becomes, the higher its tendency to extract redundant features. In this paper, CNNs were used as universal feature extractors for transfer learning, so only the final classifier needed to be retrained.

Several previous works related to our research problem will be presented in 4.1.

4.1. RELATED WORK

Existing works that introduce GP approaches for Computer Vision will now be presented. The studies described below have combined CNNs and GP to perform image classification tasks, similarly to the method proposed in this thesis.

Young et al. (2015) is one of the first studies to combine CNNs with evolutionary algorithms. The authors trained a vanilla CNN, although they used an evolutionary algorithm to optimize some hyperparameters like the kernel size and the number of filters for the convolutional layers. A similar approach is presented by Bakhshi et al. (2020), where a conventional Genetic Algorithm (GA) was used to evolve optimized CNN architectures and to find the best combination of hyperparameters for multiple image classification tasks. This was implemented in hopes of reaching the optimized CNN

model automatically. The proposed GA optimized the number of convolutional blocks, the number of convolutional layers in each block, the size of feature maps for each block, and other training-related hyperparameters such as dropout rate, learning rate, weight decay, and momentum. Recently, C. Li et al. (2021) also applied a GA, this time to automatically select trainable layers of a transfer CNN model. The main idea was to encode the number of the trainable layers as a gene of individuals and update the population by genetic operations to obtain the best transfer CNN networks.

Fernando et al. (2016) proposed differentiable pattern-producing networks (DPPNs) to optimize the weights of a denoising autoencoder. It focused on the effectiveness of indirect coding for weight optimization, that is, the general structure of the network should be predefined.

In Zhu et al. (2018) a GP framework is introduced to automatically search for optimal CNN architectures, abbreviated as GP-CNAS. GP-CNAS encodes CNNs as trees where leaf nodes (GP terminals) are selected residual blocks and non-leaf nodes (GP functions) specify the block assembling procedure. One of its advantages is that the desired CNN architecture, with balanced depth and width, can be found within limited trials.

Suganuma et al. (2017), Banerjee & Mitra (2020), and Suganuma et al. (2020) presented methods to construct CNN architectures that were based on Cartesian Genetic Programming (CGP). In Suganuma et al. (2017), the approach adopted functional modules, such as convolutional blocks and tensor concatenation, as the node functions in CGP. The CGP encoding method was optimized to maximize the validation accuracy. Similarly, in Banerjee & Mitra (2020), the authors developed an algorithm that introduces crossover to standard CGP, to generate an optimal CNN architecture for a given dataset. The method was found to converge relatively faster than other related methods. Another similar approach is in Suganuma et al. (2020), where the method generated the CNN architectures based on the CGP encoding scheme with highly functional modules and used the evolutionary algorithm to efficiently find good architectures. It could find competitive architectures with a reasonable computational cost compared to other automatic design methods. Furthermore, McGhie et al. (2020) presented the first research into the use of tree-based GP to automate the architecture design of CNNs. While the approach did not outperform state-of-the-art hand-crafted CNN architectures, it showed that using tree-based GP was a viable approach. The intent of using GP was to reduce the amount of knowledge required to design a CNN for a given problem domain by automatically searching for the optimal architecture.

In Evans et al. (2018), another GP-based method was proposed. It combined key characteristics of CNNs (like convolution and pooling), with GP in a deep structure for image classification. The proposed method successfully utilized the flexibility of GP representations to automatically evolve a program that learned convolution filter coefficients, detected useful regions of an image, extracted features from those detected regions, and built a classifier. In terms of performance, it was unable to beat CNNs, although it had a much faster testing time and it evolved interpretable models (unlike CNNs). Moreover, Bi et al. (2019) had the goal of developing a GP-based evolutionary DL method for feature learning and image classification. The framework uses GP trees to transform each image in the training set into features (with the help of convolution and pooling layers). The transformed training set was normalized using min-max normalization and was fed into a linear SVM to perform classification. The experimental results showed it achieved significantly better performance in most comparisons and revealed the good interpretability of the evolved programs.

Chu & Chu (2019) presented a fusion method based on GP to integrate information extracted from multiple layers of a CNN. The preliminary results demonstrated the potential of this GP fusion scheme.

Sun et al. (2019) developed a new evolutionary approach to automatically evolve architectures and weights of CNNs for image classification problems. The model optimized by EvoCNN had fewer parameters, yet it could reach the best classification performance. At the time, the goals were achieved by proposing a new representation for the weight initialization strategy, a new genetic operator, and an efficient fitness evaluation method, among others.

Real et al. (2019) used an evolutionary algorithm to discover image classifier architectures. The authors evolved an image classifier – AmoebaNet-A –, which had comparable accuracy to existing state-of-the-art ImageNet models discovered with more complex architecture-search methods. Evolution was found to be a simple method to effectively discover high-quality architectures. At its large size, it set a new state-of-the-art accuracy.

Recently, Wendlinger et al. (2021) and Pinos et al. (2021) worked on Neural Architecture Search (NAS), which aims to automate the design process of neural network architectures. In Wendlinger et al. (2021), the authors stated that, in general, the method was difficult to reproduce due to differences in frameworks, hidden hyperparameters, and a large number of hyperparameters. Still, it was a valid approach for applications that require inductive transfer learning. A similar method is in Pinos et al. (2021), which proposed a multi-objective NAS method based on CGP for evolving CNNs. During this process, a suitable CNN architecture is evolved together with approximate multipliers to deliver the best trade-offs between accuracy and network size.

Pretorius & Pillay (2020) implemented a methodology that is very similar to what is going to be performed in this thesis, thus its relevance. The proposed hybrid approach combined a CNN for feature extraction and GP for classification. First, VGG-19 was used to extract features, as the CNN got trained on the initial dataset. The dense layers of the CNN were then removed up until the layer that represents the flattened feature maps. The goal was to generate a new dataset where each image was represented only by a 1D feature vector. In the end, GP was trained to produce a classifier that could predict the class of an image represented by its feature vector. The final objective was to evolve a classifier that categorized each frame in a movie or series as a good or bad thumbnail.

The approach presented by Pretorius & Pillay (2020), which combined a CNN for feature extraction and GP for classification, was outperformed by a ResNet-50 CNN, but it was still an important assessment. Perhaps if a state-of-the-art algorithm like M4GP is used for classification, the performance of a hybrid approach can be better than that of a vanilla CNN. This is the premise of this thesis.

Janke et al. (2019) developed a very important study proposing a two-step approach to assessing the possibility of improving the performance of CNNs by using different classification algorithms on high-level image features. The reason for the relevance of this paper in the context of this thesis was already highlighted in Chapter 1: the authors identified a research gap in current literature that should be filled by this thesis. The methodology implemented in this thesis is, in fact, very much related to the one in Janke et al. (2019). In the paper, initially, different CNN configurations were trained on several benchmark datasets. In the second step, the fully-connected layer that made the final prediction was removed from the CNN, such that the last remaining layer produced a flattened output from the last

layer of image filters. On this intermediate output, multiple classification algorithms were benchmarked. Even so, that benchmark did not test evolutionary algorithms like M4GP, which might lead to better performance when compared to MLP models and other multiclass classification algorithms (i.e., Logistic Regression, Support Vector Machine, XGBoost, Random Forest Classifier, K-Nearest Neighbors).

5. METHODS

This chapter describes the experimental setup used to test the research question:

“To what extent does the substitution of the fully-connected layer in Convolutional Neural Networks for an evolutionary algorithm affect the performance of those CNN models?”

The training process of each CNN model on each dataset benchmark was completed on a cloud computing UI⁶, while the classification algorithm was run on commodity hardware⁷, which limits the number of tested hyper-parameters to a reasonable degree. Nevertheless, all elements of the scientific procedure presented in this thesis are scalable and subject to computational means.

The procedure of this thesis assumes time is not a constraint in the training of the algorithms, as it will inevitably take longer to train than a vanilla CNN, so it only focuses on improving the accuracy and/or interpretability of the final model solutions. The proposed method comes at the cost of longer training time. On the other hand, the stored classifiers will be lighter-weight when compared to fully-connected layers (i.e., MLPs), because the latter are very complex and the GP-based classification algorithm implemented in this thesis makes use of a fairly lightweight nearest centroid classifier (La Cava et al., 2019).

Many novel implementations of neural networks have had a biological origin. This procedure does not, although it returns to the origins of Computer Vision when the classification of high-level image filters was not limited to MLPs and could be done with other classification algorithms.

After Chapter 4 highlighted previous work in this research area, the current chapter will describe the scientific procedure in 5.1, the dataset benchmarks in 5.2, the CNN model architectures in 5.3, the techniques applied for image preprocessing in 5.4, the classification algorithm and evaluation metrics that were chosen in 5.5 and 5.6, respectively, and the software framework in 5.7.

5.1. SCIENTIFIC PROCEDURE

The focus of this thesis is not to directly replace the fully-connected layer in the learning process of the CNNs, but rather to enhance this process. Firstly, a given CNN architecture is trained (including the fully-connected layer), so that the convolutional layers learn image features from the inputs. Afterward, the fully-connected layer is replaced with a different classification algorithm (i.e., M4GP), thus training the algorithm based on the high-level image features produced in the first step of this method.

The procedure that will be explained more thoroughly below might be similar to transfer learning⁸, but it is rather more of a learning enhancement. This is because the dataset to train the model on is the

⁶ Google Colaboratory: up to 12 GB of RAM, GPU used, accessible at colab.research.google.com/

⁷ Windows 10, Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz, 8 GB of RAM

⁸ Transfer Learning exists when a model developed for a task is reused as the starting point for a model on a second task (Yosinski et al., 2014)

same as the one to further train the classification algorithm⁹. Still, pre-trained models from transfer learning can be used to perform this possible enhancement of the learning process.

The first script of the procedure serves to generate the intermediate datasets from well-known Computer Vision datasets: CIFAR-10 and CIFAR-100, as described in 5.2. Before passing the input images to the CNN architectures (5.3 goes into detail on each chosen model), some transformations are applied to them, which are described in more detail in 5.4 (Image Preprocessing). After training, the CNNs are cut so that the fully-connected layers are removed and only the flattened output of the convolutional layers remains. The high-level image features come from the learning process on the convolutional layers. These features describe the output of the image filters present on the last convolutional layer of several tested CNN architectures. Moreover, by “intermediate datasets” we are referring to the combination of the flattened output from the trained convolutional layers with the correct classification (i.e., class label) of the input image. The first stage of our procedure can be visualized in Figure 5.1.

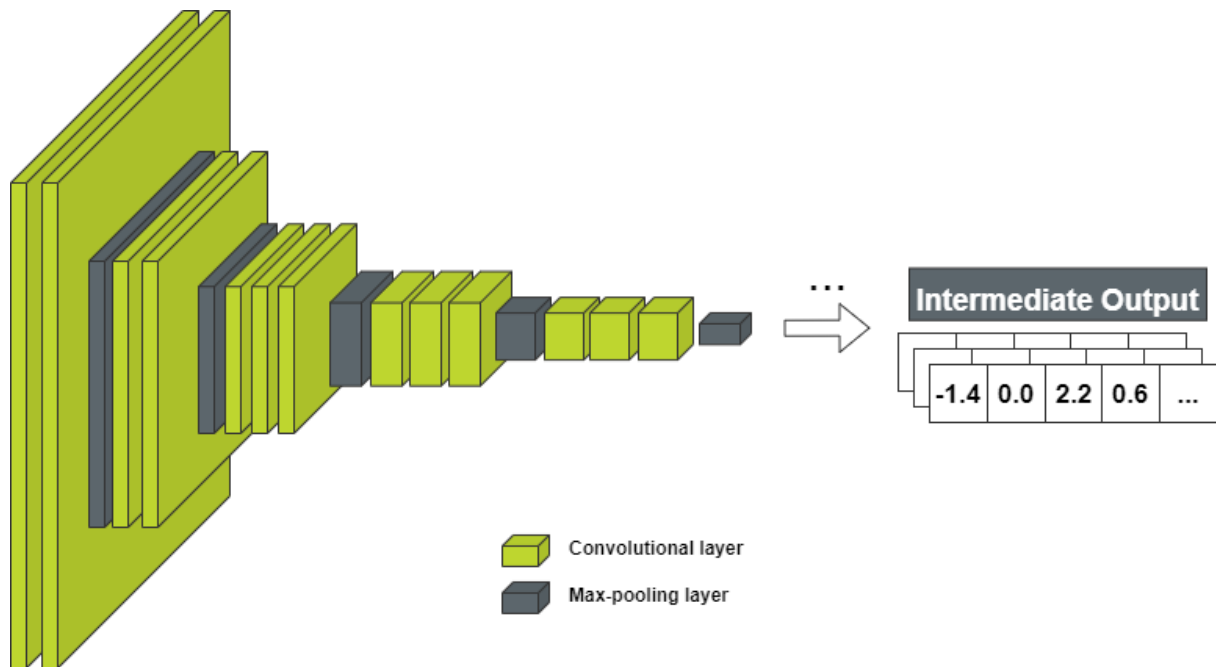


Figure 5.1 – This stage encompasses the creation of intermediate datasets, which can be considered as the flattened outputs of the convolutional layers on a CNN architecture

The networks were applied to generate good image filters, although the fine-tuning of its hyperparameters is not the focus of this thesis. The main goal is to discover if classifying image features with M4GP helps improve the image classification task of CNN models with an increased accuracy score and/or with better interpretability of results. Thus, the performance on the first step of the procedure will not have a significant influence on the results of the second step.

The second script serves to apply M4GP to classify the high-level image features in the intermediate datasets from the first stage (described above). This is done to understand if this classification stage can get better results and performance when compared to the vanilla CNN. The hyperparameters tested on M4GP are indicated in 5.5 (including the number of generations, population size, selection

⁹ In fact, to be more precise, it is a “subset” of the original dataset representing high-level image features

method, and elitism, among others). For the sake of making comparisons, the performance of the initially trained CNNs is also disclosed. The second stage of our procedure can be visualized in Figure 5.2.

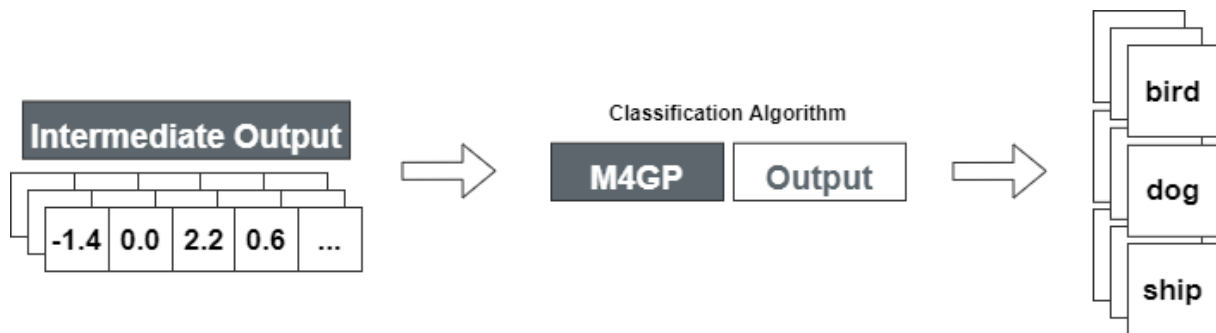


Figure 5.2 – This stage encompasses the classification of the intermediate datasets with M4GP

To ensure the quality and statistical significance of the results, some measures need to be applied. Firstly, the procedure is tested in more than one dataset. This is done to evaluate if there are certain aspects in the datasets (e.g., size and number of classes) that make M4GP perform better. Also, multiple M4GP hyperparameters are tested and explored. Furthermore, the train-test split on CIFAR data is 5:1, with 50 000 images for training and 10 000 for testing. Thus, the CNN models from the first stage learn from the training dataset. The intermediate datasets for both the training and the test dataset are then generated from those trained CNN models.

In the second stage, when training M4GP on intermediate data, its hyperparameters are obtained from the training dataset. Also, 10-fold cross-validation is applied, so 10% of the training data is used as validation data on each cross-validation run, which means all validation data comes from the training dataset of the first stage. M4GP's performance is evaluated on test data on which the algorithm was not trained. This method enables the comparability and statistical significance of results.

5.2. DATASETS

Two well-known datasets in the field of Computer Vision were chosen for this thesis: CIFAR-10 and CIFAR-100.

5.2.1. CIFAR-10

The CIFAR-10 dataset consists of 60 000 tiny color images (32×32 pixels), 50 000 for training, and 10 000 for testing, classified into 10 categories (Krizhevsky & Hinton, 2009). The dataset is balanced, thus having 6 000 examples for each of its 10 classes. The classes are mutually exclusive.

An overview of CIFAR-10 images for all 10 classes can be found in Figure 5.3.

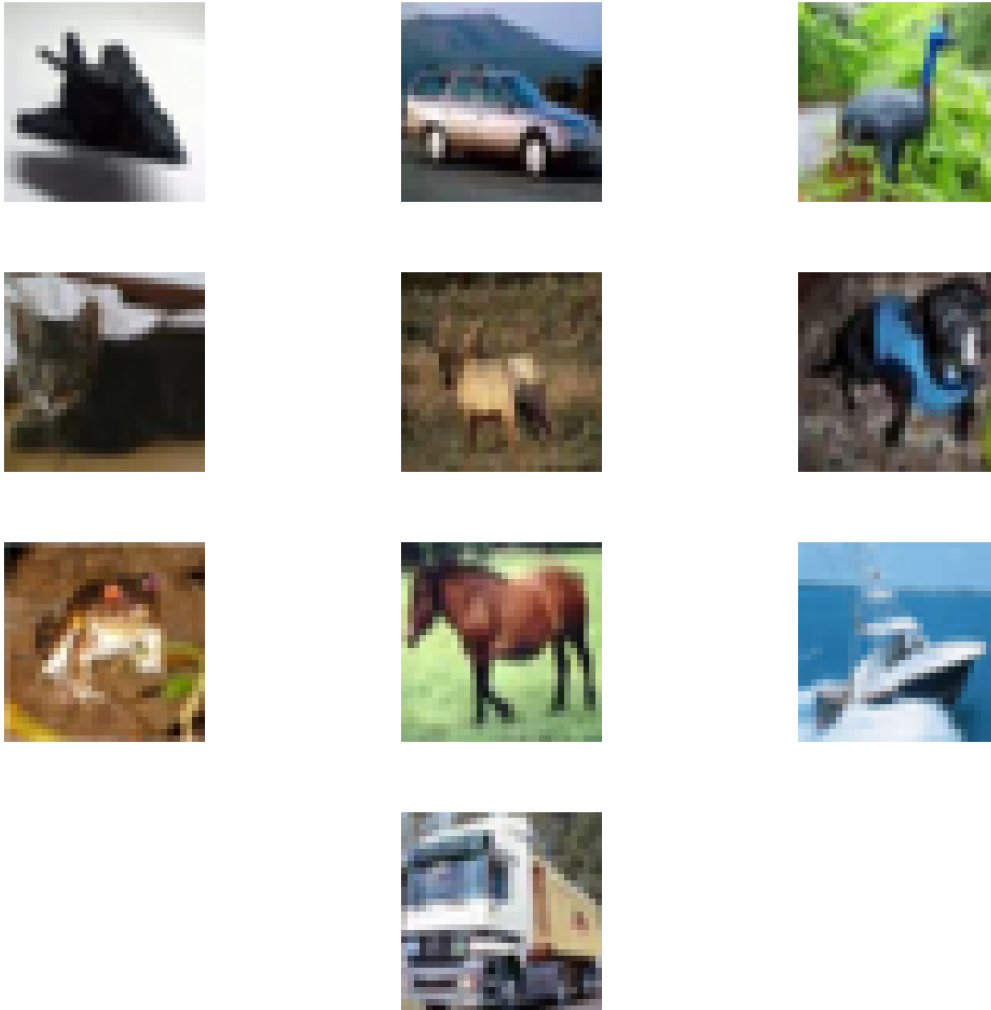


Figure 5.3 – Image examples for each one of the CIFAR-10's classes; going left to right and top to bottom, the image classes are: "airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship" and "truck"

Figure 5.4 has 6 images, which represent examples for the class "automobile" on CIFAR-10.

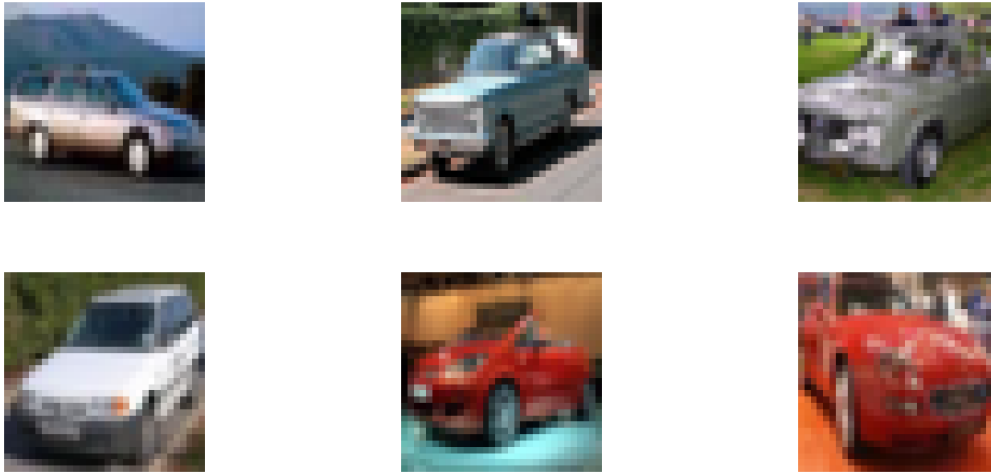


Figure 5.4 – Examples for class “automobile” on CIFAR-10

The Python version of the CIFAR-10 dataset has a size of 163 MB.

5.2.2. CIFAR-100

CIFAR-100 is an image dataset for fine-grained classification problems, compiled to contain 60 000 tiny color images distributed into 100 classes with superclasses (Krizhevsky & Hinton, 2009). Each class contains 600 images, 500 for the training phase and 100 for testing. As in CIFAR-10, the images have a size of 32×32 pixels and the classes are mutually exclusive.

Its classes come in 20 superclasses of 5 classes each. For example, the superclass “fruit and vegetables” consists of five classes “apples”, “mushrooms”, “oranges”, “pears”, and “sweet peppers”. The idea is that classes within the same superclass are similar (and thus harder to distinguish) than classes belonging to different superclasses.

An overview of CIFAR-100 images for 10 of the 100 classes can be found in Figure 5.5.

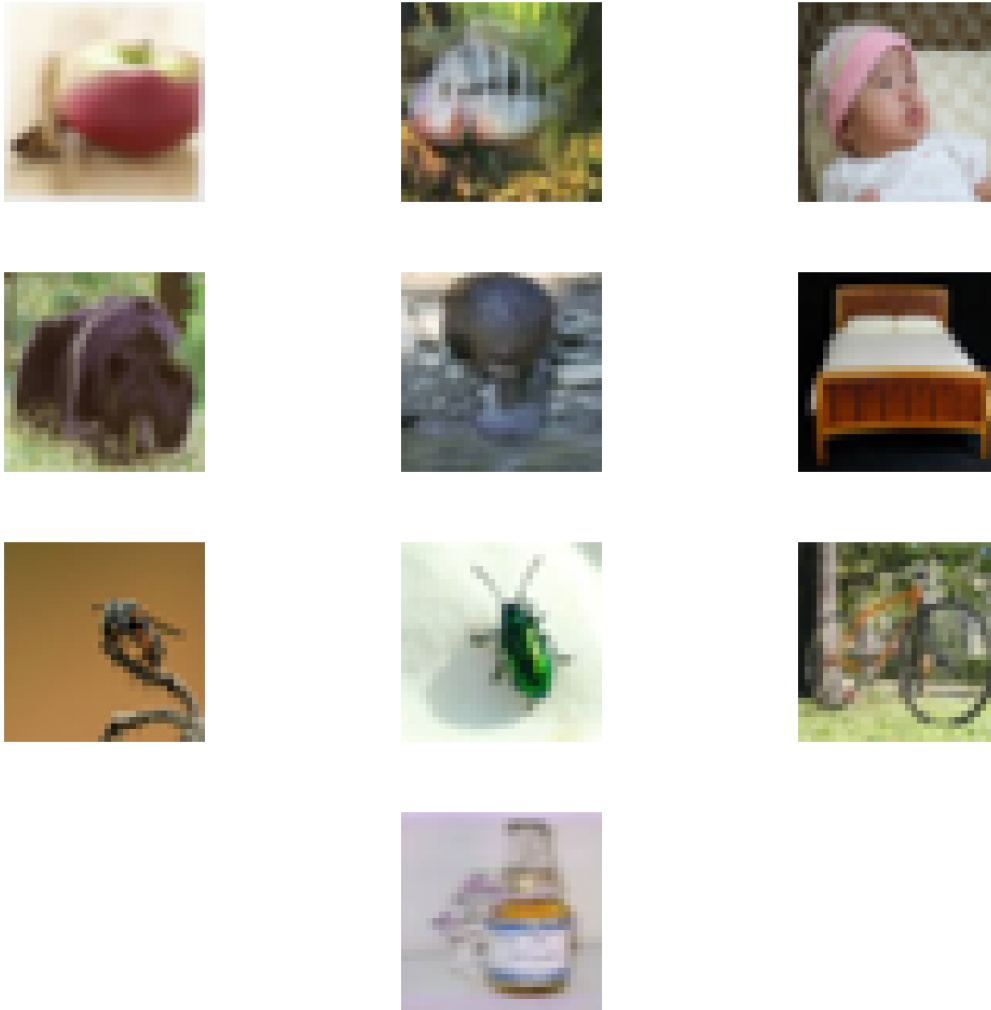


Figure 5.5 – Image examples for 10 of CIFAR-100’s classes. Going left to right and top to bottom, the image classes are: “apples”, “aquarium fish”, “baby”, “bear”, “beaver”, “bed”, “bee”, “beetle”, “bicycle” and “bottles”

Figure 5.6 has 6 images, which represent examples for the class “apples” on CIFAR-100.

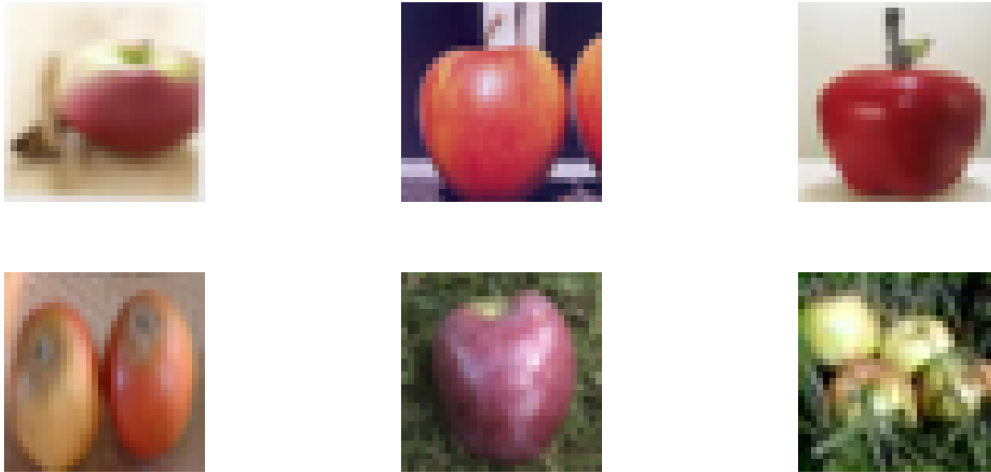


Figure 5.6 – Examples for class “apples” on CIFAR-100

The Python version of the CIFAR-100 dataset has a size of 161 MB.

5.3. CNN MODEL ARCHITECTURES

For the CIFAR-10 and CIFAR-100 datasets, the same CNN model architectures were implemented, because the images on these two datasets have the same input shape of 32×32 pixels. The only difference lies in the fully-connected layer since CIFAR-10 expects 10 output neurons and CIFAR-100 expects 100.

The models selected for CIFAR are two custom CNNs and VGG-16.

5.3.1. CNN 1

This model was created by Janke et al. (2019) for the CIFAR datasets.

CNN 1 can be seen in Figure A.1. Its architecture is a 10-layer CNN with a final fully-connected output layer. The convolutional layers use windows of 3×3 .

It has 6 convolutional layers, 3 max-pooling layers (1 following each pair of conv layers), and 2 dense layers (the last one being a softmax layer).

As the model gets deeper, the convolution filters also have an increasing depth. The two first convolution layers have a depth of 128, the two that come next have 256, and, finally, the last two have 512. Regarding these convolution layers, we can also say padding is used with strides being (1, 1); thus, the outputs have the same size as the inputs.

All hidden layers have ELU activation functions.

Additionally, it is important to note dropout layers are employed after each pair of conv layers and before the softmax layer. The rate on these layers increases as CNN 1 gets deeper, starting at 0.1, then 0.25, and, finally, 0.5.

All max-pooling layers used in this architecture have a 2×2 pooling window.

After flattening, the fully-connected layer starts with a dense layer with 1024 units, ELU activation, and dropout with a rate of 0.5. Finally, the model ends with a softmax layer with 10 or 100 units, depending on which of the two CIFAR datasets is being used.

CNN 1's architecture produces 2048 high-level intermediate features as input to the fully-connected layer. Also, it has 6 684 810 parameters, all of them being trainable.

5.3.2. CNN 2

This model was also created by Janke et al. (2019) for the CIFAR datasets.

CNN 2 can be seen in Figure A.2. It is constructed to be smaller than CNN 1 and it only has 7 hidden layers. Similarly to CNN 1, this has 2 convolutional layers followed by 1 max-pooling layer. The convolutional layers also use windows of 3×3 .

It has 4 convolutional layers, 2 max-pooling layers (1 following each pair of conv layers), and 2 dense layers (the last one being a softmax layer).

As the model gets deeper, the convolution filters also have an increasing depth. The two first convolution layers have a depth of 32 and the last two have 64. Regarding these convolution layers, we can also say padding is used with strides being (1, 1); thus, the outputs have the same size as the inputs.

All hidden layers have ReLU non-linearity.

Additionally, it is important to note dropout layers are employed after each pair of conv layers and before the softmax layer. The rate on these layers increases as CNN 2 gets deeper, starting at 0.25, and then 0.5.

All max-pooling layers used in this architecture have a 2×2 pooling window.

After flattening, the fully-connected layer starts with a dense layer with 512 units, ReLU activation, and dropout with a rate of 0.5. Finally, the model ends with a softmax layer with 10 or 100 units, depending on which of the two CIFAR datasets is being used.

CNN 2's architecture produces 2304 high-level intermediate features as input to the fully-connected layer. Also, it has 1 250 858 parameters, all of them being trainable.

5.3.3. VGG-16

This architecture was already explained in 2.3.4 and it can be visualized in Figure 2.14¹⁰. As said previously, VGG-16 was the main runner-up on ILSVRC-2014. The model implemented in this thesis is adapted to be used with the CIFAR datasets. This CNN is deeper than the ones from previous ILSVRC competitions, and it has small convolution filters (always 3×3).

¹⁰ The architecture implemented in our project is essentially equal to the one on the figure, although the fully-connected layer has some slight modifications

It has 13 convolutional layers, 5 max-pooling layers following some of the convolutional layers, and 2 dense layers (the last one being a softmax layer).

As the CNN gets deeper, the convolution filters also have an increasing depth. The two first convolution layers have a depth of 64, the two that come next have 128, the three that come next have 256, and, finally, the last six have 512 (VGG-16 will produce 512 high-level image features as input for the subsequent classifier). Regarding these convolution layers, we can also say padding is used with strides being (1, 1); thus, the outputs have the same size as the inputs. L2 regularization is also applied to decay weights with a regularization factor of 0.0005.

All hidden layers have ReLU non-linearity.

Afterward, batch normalization is applied to normalize the outputs of the convolution layers.

Additionally, it is important to note dropout layers are employed after many of the convolution layers, after the last max-pooling layer, and before the softmax layer. The rate on these layers increases as VGG-16 gets deeper, starting at 0.3, then 0.4, and, finally, 0.5.

All max-pooling layers used in this architecture have a 2×2 pooling window.

After flattening, the fully-connected layer starts with a dense layer with 512 units, an L2 regularization factor of 0.0005, and ReLU activation, followed by batch normalization and dropout with a rate of 0.5. Finally, the model ends with a softmax layer with 10 or 100 units, depending on which of the two CIFAR datasets is being used.

VGG-16's architecture has 15 001 418 parameters, among which 14 991 946 are trainable and 9 472 are non-trainable.

It is important to note that VGG-19 could have been used instead of VGG-16, although the CIFAR input images would need to be resized from 32×32 pixels to, for example, 48×48 . This is because the images from CIFAR are too small to be passed to VGG-19.

5.4. IMAGE PREPROCESSING

Image preprocessing involves the transformations done to an input image before passing it to a classifier. If raw data is passed to a classification algorithm, the accuracy of results might not be as good. Some of these preprocessing techniques can be mean normalization¹¹ (the training set is turned into organized data, normalized for each dimension), standardization¹² (the raw data gets organized for both mean and standard deviation of each of the dimensions on the training set), among others.

Image preprocessing techniques are mainly used to help surpass Computer Vision challenges. These challenges are described in 2.2.1.

¹¹ Mean normalization is calculated as: $X' = X - \mu$, where X' is the normalized data, X is the original data and μ is the mean vector across all features of X

¹² Standardization is calculated as: $X' = \frac{X - \mu}{\sigma}$, where X' is the normalized data, X is the original data, μ is the mean vector across all features of X and σ is the standard deviation vector across all features of X

For CIFAR-10 and CIFAR-100, the images were preprocessed with standardization. The mean and standard deviation for CIFAR-10 are 120.707 and 64.15, respectively. As for CIFAR-100, the mean and standard deviation are 121.936 and 68.389, respectively.

Figure 5.7 demonstrates the method, with examples for classes “horse” and “ship” with their respective preprocessed versions.



Figure 5.7 – Raw images for classes “horse” and “ship” (on the top) and their preprocessed versions (on the bottom)

These preprocessed versions of the CIFAR images were the ones used to train the CNN model architectures from 5.3.

5.5. CLASSIFICATION ALGORITHM: M4GP

After getting the intermediate datasets, M4GP (introduced in 3.4) was implemented to do classification. We will now discuss the hyperparameters tested using grid search.

Some different configurations are tested. For the number of generations g , each model uses 100 or 200. In terms of population size, 50 and 100 individuals are tested. The selection method can be Lexicase selection or Age-Fitness Pareto selection; Tournament selection was initially tested but did not lead to good results¹³. Elitism can be integrated or not (i.e., True or False). The probabilities of crossover and mutation are also explored: for crossover, it can be 50 or 80%, and for mutation, it can be 20 or 50%.

Also, we carried out some preliminary tests to understand which hyperparameters and parameter values should be included in the grid search. We concluded that, for this image classification problem,

¹³ Also, La Cava et al. (2019) proved that Tournament selection performs worse when producing accurate classifiers

the operator list and tournament size (if Tournament selection was applied) either caused the models to underperform or slowed down computation significantly.

Besides, some hyperparameters receive fixed values, namely: the algorithm is set to apply M4GP for classification (*classification*, *class_m4gp*), the fitness metric is set as the F1 score, the random state is used for reproducibility, and verbosity is activated. Also, it was necessary to set the fitness threshold for stopping execution as 0.01 to speed up the execution time. Without this setting, the algorithm would need an unreasonable amount of time to run, and we believe this setting did not have a substantial negative effect on the results, even though it is likely they could be improved if this threshold was not set. Finally, the constants (represented by the letter n) are not added to the list of operators, as previously we noticed several program representations included only constants, which was unreasonable in the context of the problem, leading to subpar results.

This classification algorithm is directly compared to the CNN's fully-connected layer (i.e., multilayer perceptron).

5.6. EVALUATION METRICS

The main evaluation metric used for this research is accuracy in the context of a multiclass classification problem.

We would have liked to use top- n -accuracy (top-2 and top-5, for example), which is the standard accuracy of the true image class being equal to any of the n most probable classes predicted by a classification model like M4GP. However, Python's implementation of M4GP used for this thesis does not have a *predict_proba* method¹⁴ (contrary to scikit-learn's classifiers), so it is not possible to calculate top- n -accuracy, only standard accuracy (i.e., top-1-accuracy).

Besides, the time taken to train the M4GP models on the intermediate datasets is calculated. It is important to note that, in terms of training time, it will always take longer to implement the approach from this thesis than to train a vanilla CNN.

The complexity of the models is generally measured in terms of the size of the produced models.

5.7. SOFTWARE FRAMEWORK

All the code used to test the research question of this thesis was written in Python, using various libraries. The key libraries used include:

- Keras¹⁵ (Chollet & others, 2015), which acts as an interface for TensorFlow¹⁶ (Abadi et al., 2016). This was mainly utilized to create, implement, and train Deep Learning models.

¹⁴ This scikit-learn method predicts the class probabilities for each data point

¹⁵ keras.io/

¹⁶ tensorflow.org/

- ANN Visualizer¹⁷, to create graphs for CNN models built on Keras. This enables us to visualize an architecture and all its layers.
- ellyn¹⁸ (La Cava et al., 2019), which is a GP system for regression and classification. This is a Python-wrapped version of ellen, a C++ library that does most of the computation. ellyn is used to implement M4GP (introduced in Chapter 3), a state-of-the-art GP-based classification algorithm needed to perform the stage described in 5.5. It is important to emphasize ellyn’s compatibility with scikit-learn because it can be used as a scikit-learn estimator, which makes it easy to do cross-validation and hyperparameter optimization.
- scikit-learn¹⁹ (Pedregosa et al., 2011), to implement cross-validation methods and evaluation metrics.

The framework is divided into two Jupyter Notebooks, as explained in 5.1. The first notebook is used to do data exploration (check the shape of the datasets and print image examples for each class), data transformation (apply an image preprocessing technique and one-hot encode the target values), neural network training (build CNN models – including the data augmentation step, the compile, and the training –, afterward we check the number of parameters and the test loss/accuracy) and, finally, we remove the fully-connected layers of each CNN model to create the intermediate datasets.

The second notebook is used to perform the final classification of images. Initially, the intermediate datasets from the first notebook are loaded, and then we train M4GP models on that data, thus producing the final models. The notebook also saves the models for future reuse and stores the evaluation metric results in a file.

This project can be found on GitHub at github.com/rfmmonteiro99/classify-img-features-with-gp.

¹⁷ github.com/RedaOps/ann-visualizer

¹⁸ cavalab.org/ellyn/

¹⁹ scikit-learn.org/stable/

6. RESULTS AND DISCUSSION

This chapter will present the results obtained from the implementation of the methodology presented in Chapter 5.

In 6.1, the results from creating the intermediate datasets are described. Then, the results from the classification task on these intermediate datasets are described in 6.2. The interpretations, analyses, and general conclusions from those results are described in 6.3. Finally, the discussion will be presented in 6.4.

6.1. CREATION OF INTERMEDIATE DATASETS

We will now analyze the training process behind the creation of intermediate data. To accomplish this, we trained CNNs on the initial datasets: CIFAR-10 and CIFAR-100.

The set of CNNs chosen for this step of the research was already explained in 5.3. It includes two custom architectures (CNN 1 and CNN 2) and a famous and commonly used architecture (VGG-16).

CNNs 1 and 2 were trained for 80 iterations, and they seem to have converged for both the train and test datasets. For VGG-16 we used pre-trained weights. Therefore, this model did not need to be retrained²⁰.

6.1.1. CIFAR-10

The results can be found in Table 6.1. The two custom CNNs took approximately 45 minutes to train²¹ (38 minutes for CNN 1 and 7 for CNN 2).

Table 6.1 – CIFAR-10 intermediate dataset results

Architectures	Train Accuracy	Test Accuracy	# Parameters	# Intermediate Features	Model Size (MB)
CNN 1	0.982	0.833	6 684 810	2 048	25.5
CNN 2	0.912	0.798	1 250 858	2 304	4.8
VGG-16	0.936	0.936	15 001 418	512	57.3

By analyzing the two charts from Figure 6.1, we can visualize the training process from CNN 1 on CIFAR-10 over 80 iterations. We can see that both accuracy and loss change very rapidly until the 10th iteration, with accuracy increasing to almost 0.8 in both the train and test sets and loss decreasing to approximately 0.65 (also in both sets). As for the accuracy on the train set, we can see it continually increases to a very high result of 0.98, stalling at around the 40th iteration. However, the same did not

²⁰ The VGG-16 pre-trained weights for CIFAR were retrieved from github.com/geifmany/cifar-vgg

²¹ Completed on Google Colaboratory: up to 12 GB of RAM, GPU used, accessible at colab.research.google.com/

happen on the test set, which stalled in accuracy at around 0.8, with some fluctuating values from the 10th iteration to the end of the training process. As for the loss values on the train set, there is a persistent decrease until a value of 0.05, stalling at around the 40th iteration, in a similar way to the accuracy. Still, once again, the same did not happen on the test set, which fluctuated between a value of 0.8 until the final iteration, even registering a slight increase in loss after the 20th iteration. There is a significant gap between training and test accuracy, while the test loss also registers a slight increase along training, although this is not completely detrimental to the quality of the architecture. CNN 1 is a model with 6 684 810 parameters, which produces 2 048 high-level image features for the intermediate dataset and has a size of 25.5 megabytes.

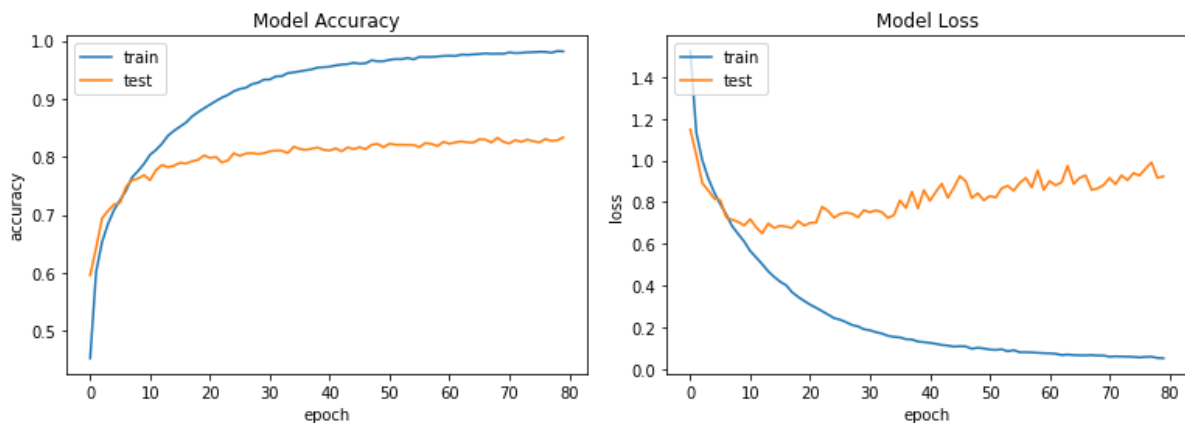


Figure 6.1 – CNN 1 training process (accuracy and loss) on CIFAR-10

By analyzing the charts from Figure 6.2, we can visualize the training process from CNN 2 on CIFAR-10 over 80 iterations. Until the 21st epoch, the accuracy and loss on both sets are very similar, similar to the training process of CNN 1. Afterward, the accuracy on the train set keeps increasing until it reaches 0.91 by the end of training and the loss keeps decreasing until it reaches 0.25. As for the test set, we see a shift after the 22nd epoch, with the accuracy stalling at around 0.79, finishing at 0.798, and the loss also stalling at around 0.63, finishing at 0.662. CNN 2 is a model with 1 250 858 parameters, which produces 2 304 high-level image features for the intermediate dataset and has a size of 4.8 megabytes.

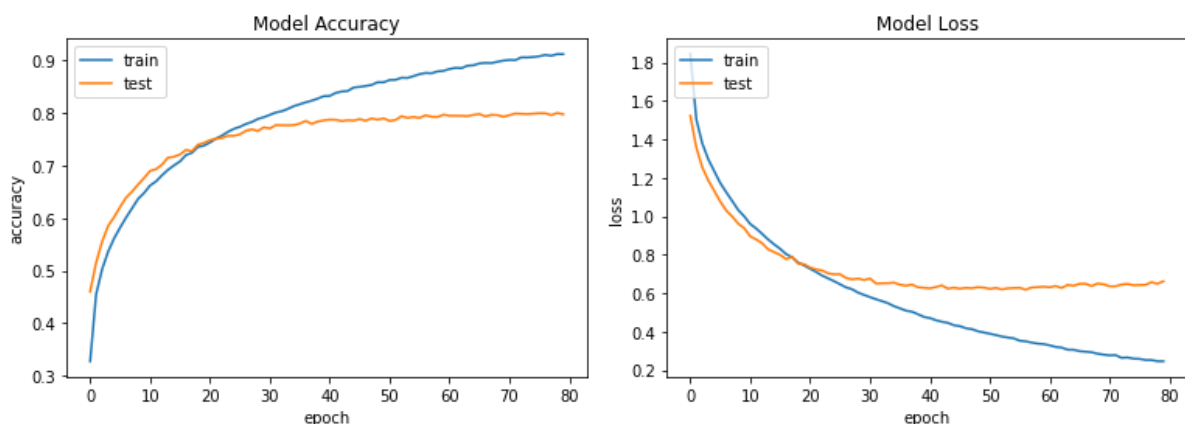


Figure 6.2 – CNN 2 training process (accuracy and loss) on CIFAR-10

As stated previously, pre-trained weights were used for VGG-16. This architecture reached the highest accuracy on the CIFAR-10 test set, with a score of 0.936. It is also the biggest model by a large margin,

having 15 001 418 parameters and a size of 57.3 megabytes, which is more than double compared to CNN 1. It produces 512 high-level image features.

Finally, we can compare the performance of CNNs 1 and 2. CNN 1 was able to reach slightly better test accuracy, with an extra 3.5 percentage points compared to CNN 2. The latter is a lighter model, which explains the worse performance on unseen data. These two CNNs have signs of possible overfitting, although the results on the test data did not worsen significantly, in some cases, they even slightly improved, so we can consider them for further analyses. VGG-16 has a much more complex architecture when compared to CNNs 1 and 2, which might explain why it was more proficient in observing patterns in unseen image data. VGG-16 got a test accuracy with an extra 10.3 and 13.8 percentage points compared to CNNs 1 and 2, respectively. As stated previously, the scope of this thesis is not to reach the best possible performance at this stage, so these results will be used for the next phase of the research.

6.1.2. CIFAR-100

The results can be found in Table 6.2. The two custom CNNs took approximately 46 minutes to train²² (38 minutes for CNN 1 and 8 for CNN 2).

Table 6.2 – CIFAR-100 intermediate dataset results

Architectures	Train Accuracy	Test Accuracy	# Parameters	# Intermediate Features	Model Size (MB)
CNN 1	0.923	0.567	6 777 060	2 048	25.8
CNN 2	0.638	0.484	1 297 028	2 304	5.0
VGG-16	0.705	0.705	15 047 588	512	57.5

By analyzing the two charts from Figure 6.3, we can visualize the training process from CNN 1 on CIFAR-100 over 80 iterations. This model's train accuracy continually increases until reaching 0.923 at the end of training. As for test accuracy, we see it increases together with train accuracy until the 11th epoch. Afterward, the test accuracy flattens at values around 0.50, as it does not accompany the strict increase in train accuracy, finishing at 0.57. As for loss, we also see similar values between train and test until the 10th epoch. Afterward, the test loss does not decrease as much when compared to train loss. The latter continually decreases until 0.24, although test loss starts slightly increasing (the lowest value was 1.79 at the 23rd epoch and it finishes with a value of 2.59). Even though this might indicate the model is overfitting (losing generalizability) after 20 epochs, we do not see any major negative effects in terms of test accuracy. CNN 1 is a model with 6 777 060 parameters, which produces 2 048 high-level image features for the intermediate dataset and has a size of 25.8 megabytes.

²² Completed on Google Colaboratory: up to 12 GB of RAM, GPU used, accessible at colab.research.google.com/

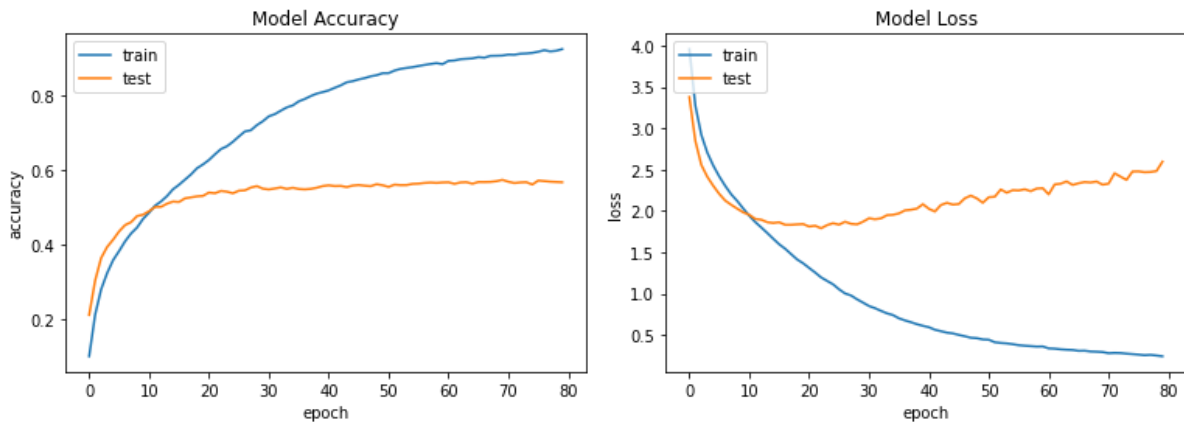


Figure 6.3 – CNN 1 training process (accuracy and loss) on CIFAR-100

By analyzing the charts from Figure 6.4, we can visualize the training process from CNN 2 on CIFAR-100 over 80 iterations. Train and test accuracy increased in similar ways until the 30th epoch. Afterward, the training accuracy keeps increasing until it reaches 0.638, although test accuracy starts to flatten at values around 0.45, finishing at 0.484. As for loss, we also see similar behavior in the train and test sets until the 30th epoch. Then, train loss keeps decreasing until 1.24, even though test loss starts to flatten at values of approximately 2.1, finishing at 2.08. CNN 2 is a model with 1 297 028 parameters, which produces 2 304 high-level image features for the intermediate dataset and has a size of 5 megabytes.

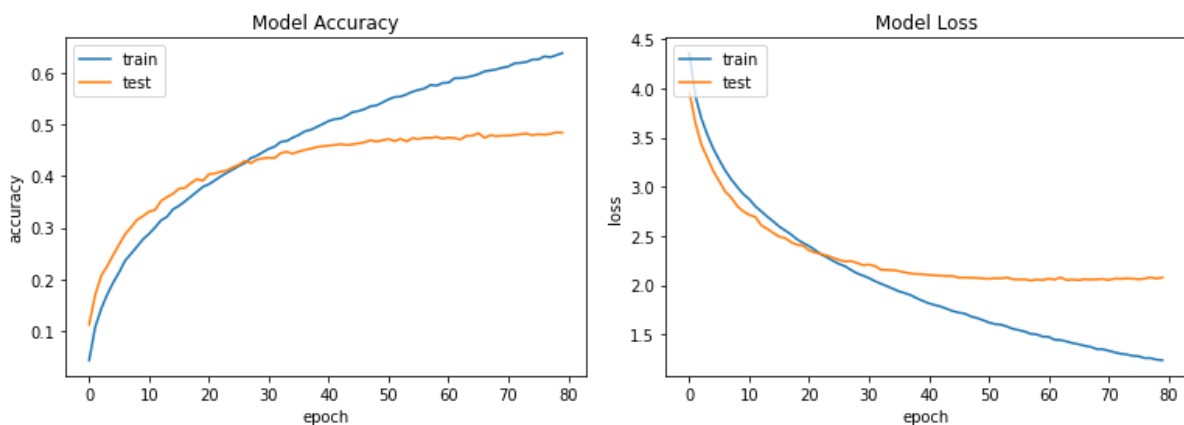


Figure 6.4 – CNN 2 training process (accuracy and loss) on CIFAR-100

As it was said previously, and similarly to CIFAR-10, pre-trained weights were used for VGG-16. This architecture reached the highest accuracy on the CIFAR-100 test set, with a score of 0.705. It is also the biggest model by a large margin, having 15 047 588 parameters and a size of 57.5 megabytes, which is more than double compared to CNN 1. It produces 512 high-level image features.

Finally, comparing CNNs 1 and 2, we can see CNN 1 reached better test accuracy, with an extra 8.3 percentage points compared to CNN 2. CNN 2 ended with much lower accuracy scores, probably because it has a less complex architecture. Even so, both CNNs converged on the test set. VGG-16 has a much more complex architecture when compared to CNNs 1 and 2, which might explain why it was more proficient in observing patterns in unseen image data. VGG-16 got a test accuracy with an extra 13.8 and 22.1 percentage points compared to CNNs 1 and 2, respectively. As stated previously, the

scope of this thesis is not to reach the best possible performance at this stage, so these results will be used for the next phase of the research.

6.2. M4GP CLASSIFICATION ON INTERMEDIATE DATASETS

The results from M4GP will be shown in tables, highlighting the classification accuracy on the test sets, run-time²³, and the size of the resultant classifiers, among others. In addition, some data about the 10-fold cross-validation will be displayed (95% confidence interval for the mean accuracy on the validation set). The hyperparameters on the best models will be reviewed. Finally, M4GP's program representations will also be discussed.

6.2.1. CIFAR-10

The results can be found in Table 6.3.

Table 6.3 – M4GP final classification results on CIFAR-10

Intermediate Data	Train Accuracy	Validation Accuracy - Confidence Interval	Test Accuracy	Run-time	Model Size (MB)
CNN 1	1	95% CI [1, 1]	0.830	33450.292 sec (9.29 hr)	44
CNN 2	0.995	95% CI [0.9949, 0.9967]	0.791	35663.360 sec (9.91 hr)	44
VGG-16	0.999	95% CI [0.9996, 0.9998]	0.926	36012.507 sec (10.0 hr)	40

By analyzing the previous table, in terms of accuracy, we can see that M4GP seems to be overfitted on training data. On CNN 1's intermediate dataset it got perfect training accuracy, on CNN 2's it got 0.995, and on VGG-16's it got 0.999 (almost perfect as well). Still, if we analyze the confidence intervals from validation accuracy on the 10-fold cross-validation, we can see the results with a significance level of 0.05 are also very high, which indicates the models consistently reached good performance on images that were not used on the training stage. Finally, on the test set's unseen images, we can say the results are very satisfactory. These accuracies are not as high as in the training set, but M4GP still reached 0.830 in CNN 1's intermediate dataset, 0.791 in CNN 2's, and 0.926 in VGG-16's.

We can also see the M4GP's run-times are comparable, all between 9 and 10 hours approximately.

Additionally, model sizes are very similar. We can say these are very light models, with a size of much less than a megabyte (M4GP for CNNs 1 and 2 is 44 kilobytes and for VGG-16 it is 40 kilobytes).

The hyperparameters for the best-performing M4GP models in each intermediate dataset from CIFAR-10 will now be presented. It is important to mention three of them were the same on all intermediate datasets after the grid search was performed. The number of generations could be 100 or 200, but all

²³ Obtained with Windows 10, Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz, 8 GB of RAM

grid searches selected 100. The population size could be 50 or 100, but all grid searches selected 100. As for the mutation rate, it could be 0.2 or 0.5, but all grid searches selected 0.5. Now, we can discuss three other hyperparameters that had different results depending on the dataset (elitism could be true or false, crossover rate could be 0.5 or 0.8, selection method could be Lexicase selection or Age-Fitness Pareto selection):

- CNN 1: On CNN 1's intermediate dataset, grid search integrated elitism, chose a 0.8 crossover rate, and Age-Fitness Pareto selection
- CNN 2: On CNN 2's intermediate dataset, grid search integrated elitism, chose a 0.5 crossover rate, and Lexicase selection
- VGG-16: On VGG-16's intermediate dataset, grid search did not integrate elitism, chose a 0.8 crossover rate, and Lexicase selection

Furthermore, M4GP does not have the typical tree-based GP representation, using stack-based representations instead, as said in Chapter 3. For these program representations (which can be seen in the verbose output of M4GP's grid search), in CIFAR-10, the classification algorithm seems to do a feature selection during the evolution of populations. This is likely because the features coming from CNN's filters are already very good discriminators, so the algorithm does not need to add transformative and novel equations to the stack. Similar results were obtained over all intermediate datasets. It is interesting to note it is extremely rare for M4GP to use arithmetic operators²⁴, which means it does very little "feature engineering" with the different high-level image filters. One of the most applied operators is *sqr*t (square root), although *sin* (sine), *cos* (cosine), *exp* (exponential), and *log* (logarithm) are also found. As an example, in CNN 2, some equations are just ' x_3 ', which means the algorithm chose this high-level image feature without changing it, but there is also ' $exp(sqr(|(x_6 * x_4)|))$ ', which is a more complex equation that multiplies feature x_4 by feature x_6 , gets the square root, and calculates the exponential of that result. Finally, in terms of features, M4GP explores all 10 features in CNN 1's intermediate dataset, and CNN 2's. In VGG-16's intermediate dataset, M4GP has many more to choose from (512 features), but it is clear the algorithm explores hundreds of options.

6.2.2. CIFAR-100

For the CIFAR-100 intermediate datasets, it was not possible to reach any results.

The Python framework presented previously was run in commodity hardware²⁵ to implement the methodology explained in Chapter 5. However, after running for approximately 150 hours²⁶, no results were obtained regarding the grid search with 10-fold cross-validation for CIFAR-100. This highlights a significant limitation of this thesis, coming from a combination of the low specifications of the used

²⁴ Addition (+), subtraction (−), multiplication (*) and division (/)

²⁵ Once again, these are the PC hardware specifications: Windows 10, Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz, 8 GB of RAM

²⁶ In 6.2.1., for CIFAR-10, we can see the total run-time for the three intermediate datasets was only 29.2 hours

hardware and the possible low efficiency of ellyn, the Python package used to implement M4GP. This limitation will be explained more thoroughly in Chapter 8.

Consequently, there will be no reference to CIFAR-100 in 6.3 (Analysis of Results) and 6.4 (Discussion).

6.3. ANALYSIS OF RESULTS

We now have the information we need to evaluate the results from 6.1 and 6.2, meaning we can make a comparison between the vanilla CNNs and the methodology applied in this thesis, which includes M4GP for the classification of high-level image features.

It is important to note there was not a considerable decrease in performance by running the classification algorithm separately (after training the vanilla CNNs). This indicates the separate learning process is valid, and it will not yield poor performance just because of this extra stage.

It is also important to mention that CIFAR-10's intermediate datasets have a low number of output classes (10), while CIFAR-100's have a high number of output classes (100), so we can draw conclusions and make comparisons between these two types of image classification task.

We will now provide the main conclusions regarding CIFAR-10.

6.3.1. CIFAR-10

None of the three M4GP models was able to surpass the results of the fully-connected layers from vanilla CNNs in terms of test accuracy.

Still, train accuracy increased considerably. For CNN 1, it increased to perfect accuracy (1.8 percentage point increase). As for CNN 2, it increased by 8.3 percentage points. Finally, for VGG-16 it increased by 6.3 percentage points. These increases could indicate the three models had a better understanding of the train dataset's high-level image features when compared to the fully-connected layers. As discussed previously, it could also indicate there was an overfit on training data, due to such high (and almost maximum) train accuracy. However, none of these theories is true.

In fact, for CNN 1, the test accuracy with M4GP decreased by 0.3 percentage points. As for CNN 2, it decreased by 0.7 percentage points. Finally, for VGG-16 it decreased by 1 percentage point. The decreases were minimal, although highest with VGG-16, which is the architecture with the most intermediate features (512) and the highest test accuracy in the initial training stage. Therefore, all architectures have almost the same performance, with none of the CNN+M4GP models from 6.2 being able to improve on the pre-existing test accuracy values.

In terms of run-times, these were all very lengthy on commodity hardware, adding more than 29 extra hours to the training time of our benchmark. If training is performed on a powerful computer, the run-times will, of course, diminish.

Furthermore, the M4GP models are very lightweight (less than 50 kilobytes each), which might be important if we need light classifiers for the image classification task we are tackling.

Thus, if performance is the sole importance in an image classification task like the one presented here (low number of output classes), we conclude that M4GP is not a good option for a separate classification training stage. On CIFAR-10, a dataset with a low number of output classes, the fully-connected layers from the vanilla CNNs are more proficient for the image classification task than M4GP. Thus, it is not beneficial to add the extra stage of creating an M4GP classifier, although it might be an interesting route if one has a powerful computer and needs a very light classifier in terms of model size.

6.4. DISCUSSION

At this point, we have the results needed to give an answer to the research question posed in Introduction and to reveal the main findings from this research in the next and final chapter, Conclusions. However, first, it is relevant to place these results in context by comparing them with the literature covered in the literature review.

The research project that is most comparable to this thesis is Janke et al. (2019), which was previously mentioned in the Introduction and Chapter 4. In their journal article, the authors answered the following research question: “Is it possible to improve the performance of the computer vision classification model by using different classification algorithms on high-level image features?”. It was proven that, overall, “other classification algorithms are usually more proficient than MLPs if the intermediate dataset has many input features or if the classification problem has many output classes”. Also, the authors concluded that it is “good practice to benchmark as many different classification algorithms and setups as possible, subject to time and computational constraints”. In terms of methodology, the referenced study is very similar to this thesis, although it did not investigate the usage of evolutionary algorithms to perform the classification of high-level image features, which ours did with M4GP. Thus, in a way, these two studies complement each other and are comparable on many levels.

We will now join the results from this research and Janke et al. (2019) in tables, to further compare the performance of different classification algorithms in CNN’s high-level image features²⁷.

Tables 6.4, 6.5, and 6.6 include the results for CIFAR-10’s intermediate data²⁸.

²⁷ In reality, it is not possible to make a direct comparison between the two studies, because neural networks are stochastic, as they make use of randomness during learning. Thus, even if the CNN architectures are the same on both research projects, most results on the intermediate datasets are slightly different. Still, we do not have access to the exact CNNs’ weights obtained on Janke et al. (2019), so we will compare both studies without focusing on this matter.

²⁸ The classification algorithms on the first column of each table are: Support Vector Machine (SVM), Logistic Regression (LR), K-Nearest Neighbors (KNN), Random Forest (RFE), AdaBoost (ADB), Gradient Boosting (GBC), XGBoost (XGB) and, lastly, M4GP

Table 6.4 – Comparison of CNN 1's intermediate data on CIFAR-10 (Janke et al., 2019)

Model	Test Accuracy	Avg. Validation Accuracy	Avg. Training Accuracy	Model Size (KB)
SVM	0.8454	0.9819	1.0000	497 200
LR	0.8469	0.9998	0.8130	191
KNN	0.7369	0.8245	0.8691	1 300 000
RFE	0.8120	0.9254	1.0000	1 990 000
ADB	0.7967	0.9119	0.9614	843
GBC	0.8175	0.9722	1.0000	1 200
XGB	0.8182	0.9527	0.9985	2 400
M4GP	0.830	1.000	0.99998	44

Table 6.5 – Comparison of CNN 2's intermediate data on CIFAR-10 (Janke et al., 2019)

Model	Test Accuracy	Avg. Validation Accuracy	Avg. Training Accuracy	Model Size (KB)
SVM	0.7952	0.7851	1.0000	724 600
LR	0.7622	0.7750	0.8130	191
KNN	0.6170	0.5789	1.0000	1 460 000
RFE	0.6994	0.6962	1.0000	206 300
ADB	0.6446	0.6456	0.6616	844
GBC	0.7022	0.7015	0.7818	1 300
XGB	0.7353	0.7331	1.0000	22 100
M4GP	0.791	0.996	0.996	44

Table 6.6 – Comparison of VGG-16’s intermediate data on CIFAR-10 (Janke et al., 2019)

Model	Test Accuracy	Avg. Validation Accuracy	Avg. Training Accuracy	Model Size (KB)
SVM	0.9349	0.9999	0.9999	3 500
LR	0.9346	0.9999	0.9999	46
KNN	0.9350	0.9999	0.9999	324 800
RFE	0.9350	0.9999	1.0000	868
ADB	0.9330	0.9998	1.0000	844
GBC	0.9318	0.9996	1.0000	1 100
XGB	0.9327	0.9998	1.0000	273
M4GP	0.926	0.9997	0.9998	40

In terms of similarities, we can see that M4GP’s performance on CNNs 1 and 2 (Tables 6.4 and 6.5, respectively)²⁹ is very comparable to the pool of algorithms existent in Janke et al. (2019). In terms of average training accuracy, on CNN 1, M4GP was only surpassed by Support Vector Machine, Random Forest, and Gradient Boosting. On CNN 2, by Support Vector Machine, K-Nearest Neighbors, Random Forest, and XGBoost. In both contexts, the referred classification algorithms reached perfect accuracy (100%), but M4GP also reached very high accuracy values (99.998% on CNN 1 and 99.6% on CNN 2), thus we can say performances are comparable.

As for average validation accuracy, we can see in Tables 6.4 and 6.5 that M4GP reached the best results. On CNN 1’s intermediate data, the only algorithms that achieved similar results were Support Vector Machine and Logistic Regression, with 98.19% and 99.98%, respectively, while M4GP got 100%. On CNN 2’s intermediate dataset M4GP won by a large margin of 99.6%. None of the other classifiers reached similarly high results. Finally, considering test accuracy, we can see that M4GP was the 3rd best algorithm in Table 6.4, after the Support Vector Machine and Logistic Regression, and it was the 2nd best in Table 6.5, only after the Support Vector Machine.

In terms of differences between both studies, we can say the evolutionary algorithm, M4GP, has very small model sizes. The only classification algorithm that gets close to M4GP in this regard is Logistic Regression, which in Table 6.6 has a size of 46 kilobytes for VGG-16’s intermediate data. Still, M4GP has a model with 6 kilobytes less even for this intermediate dataset. As said previously, these light classifiers might be useful if the model size is important for a specific image classification task. Random Forest, for example, generated a classifier with 1.99 gigabytes of size, as seen in Table 6.4, which is, approximately, 44 227 times larger than its M4GP counterpart (which is 44 kilobytes, the equivalent of 4.4×10^{-5} gigabytes). Another difference is M4GP’s performance on VGG-16’s intermediate dataset for CIFAR-10 (Table 6.6). This evolutionary algorithm was the worst performing in a pool of eight algorithms. It reached the lowest average training accuracy, average validation accuracy, and test accuracy. Its values are not very far from the results of Support Vector Machine, Logistic Regression,

²⁹ These intermediate datasets have the least amount of high-level image features (only 10 each)

K-Nearest Neighbors, Random Forest, AdaBoost, Gradient Boosting, and XGBoost, although we can see M4GP had weaknesses when dealing with this intermediate dataset, which had many more high-level image features than CNNs 1 and 2.

7. CONCLUSIONS

CNNs have been the most implemented algorithm in the image classification field of AI. The thesis' main objective was to analyze the proficiency of fully-connected neural networks. This was accomplished by assessing if the usage of a state-of-the-art evolutionary algorithm for multiclass classification like M4GP (Chapter 3) improved the performance of CNNs. The research question was formulated as:

“To what extent does the substitution of the fully-connected layer in Convolutional Neural Networks for an evolutionary algorithm affect the performance of those CNN models?”

To test this question, the experimental setup from Chapter 5 included two steps: the generation of intermediate datasets, and the classification of high-level image features in the intermediate datasets with M4GP.

In the first stage, the intermediate datasets were generated from two well-known image datasets: CIFAR-10 and CIFAR-100. Each consists of 60 000 colored images, while the classes are mutually exclusive. Before passing the input images to the CNN architectures (two custom architectures and VGG-16), some image preprocessing transformations were applied to them, namely standardization. After training, the CNNs were cut so that the fully-connected layers are removed and only the flattened output of the convolutional layers was returned. By “intermediate datasets” we refer to the combination of the flattened output from the trained convolutional layers with the correct class label of the input image.

As for the second stage, the high-level image features in the intermediate datasets were classified with M4GP to compare results with a vanilla CNN. The hyperparameters tested on M4GP using grid search included the number of generations, population size, selection method, elitism, and the probabilities of crossover and mutation. Accuracy was mainly used as the evaluation metric.

This approach was implemented in Python with Jupyter Notebooks, using libraries like Keras, ANN Visualizer, ellyn, and scikit-learn.

The results were reported in Chapter 6 and enabled us to compare vanilla CNNs with our methodology.

The results on CIFAR-10's intermediate datasets revealed none of the M4GP models was able to surpass the results of the fully-connected layers from vanilla CNNs in terms of test accuracy. Nevertheless, the decreases were almost negligible: for CNN 1, the test accuracy with M4GP decreased by 0.3 percentage points; for CNN 2, it decreased by 0.7 percentage points; finally, for VGG-16 it decreased by 1 percentage point. Therefore, all architectures had almost the same performance. The run-times were very lengthy on commodity hardware, adding more than 29 extra hours to the training time. Still, all M4GP models were very lightweight (less than 50 kilobytes each), which might be important if one needs light classifiers for an image classification task.

When compared to other classification algorithms³⁰, M4GP had some similarities and differences on CIFAR-10's intermediate datasets. In fact, on CNNs 1 and 2, M4GP's performance was very similar to

³⁰ Support Vector Machine, Logistic Regression, K-Nearest Neighbors, Random Forest, AdaBoost, Gradient Boosting and XGBoost

the pool of classifiers. Considering test accuracy, M4GP was the 3rd best algorithm on CNN 1, after the Support Vector Machine and Logistic Regression, and it was the 2nd best on CNN 2, only after the Support Vector Machine. As for differences, we can say M4GP always produced models with smaller sizes. Also, on VGG-16's intermediate dataset, M4GP was the worst-performing classifier.

As for CIFAR-100's intermediate datasets, no results were achieved due to a strong limitation of this study. This will be further analyzed in the next Chapter.

To answer our research question, we can conclude that if performance is the sole importance in an image classification task like the one discussed above (low number of output classes – CIFAR-10), then M4GP is not a good option for a separate classification training stage. On CIFAR-10, the fully-connected layers from the vanilla CNNs were more proficient for the image classification task than M4GP. It might be beneficial to add the extra stage of creating an M4GP classifier only if one has a powerful computer and needs a very light classifier in terms of model size.

8. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

As with most studies, the design of the current study was subject to limitations. The main one is related to an image dataset selected for the study, CIFAR-100, which was intended to have a high number of output classes (100 classes). This was chosen to make comparisons with CIFAR-10 (10 classes) concerning the methodology applied in this thesis. Nonetheless, *ellyn*, the Python library used to implement the M4GP algorithm, was unable to return any output from an M4GP grid search on CIFAR-100's intermediate datasets. In fact, after a very lengthy run-time of 150 hours in commodity hardware, no results were obtained, so it was not possible to keep the Python kernel running longer.

This came from a combination of the low specifications of the used hardware and the possible low efficiency of *ellyn*. This also negatively affected the run-time for M4GP on CIFAR-10's intermediate datasets, which summed up to more than 29 hours.

For future studies, it is suggested to use more image datasets, especially ones with a higher number of output classes, to better benchmark the applied methodology. It would be relevant to make a broader benchmark, with more image datasets, CNN architectures, and maybe even new GP-based classifiers if any research paper has introduced another state-of-the-art algorithm apart from M4GP.

It would also be important to improve the *ellyn* library and increase its efficiency when implementing M4GP in a large multiclass classification task.

9. REFERENCES

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., ... Zheng, X. (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*.
- Ayinde, B. O., Inanc, T., & Zurada, J. M. (2019). *On Correlation of Features Extracted by Deep Neural Networks*. 1–8.
- Bakhshi, A., Chalup, S., & Noman, N. (2020). Fast Evolution of CNN Architecture for Image Classification. In *Deep Neural Evolution* (pp. 209–229). Springer.
- Bakurov, I. (2018). *An Initialization Technique for Geometric Semantic Genetic Programming based on Demes Evolution and Despeciation*. (Master's thesis, Universidade NOVA de Lisboa, Lisbon, Portugal). Retrieved from Repositório da UNL. (<http://hdl.handle.net/10362/34384>).
- Banerjee, S., & Mitra, S. (2020). *Evolving Optimal Convolutional Neural Networks*. 2677–2683.
- Bi, Y., Xue, B., & Zhang, M. (2019). *An Evolutionary Deep Learning Approach Using Genetic Programming with Convolution Operators for Image Classification*. 3197–3204.
- Chen, Y., Jiang, H., Li, C., Jia, X., & Ghamisi, P. (2016). Deep Feature Extraction and Classification of Hyperspectral Images Based on Convolutional Neural Networks. *IEEE Transactions on Geoscience and Remote Sensing*, 54(10), 6232–6251.
- Chen, Y., Lin, Z., Zhao, X., Wang, G., & Gu, Y. (2014). Deep Learning-Based Classification of Hyperspectral Data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(6), 2094–2107.
- Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.
- Chollet, F., & others. (2015). *Keras*. GitHub. <https://github.com/keras-team/keras>
- Chu, W.-T., & Chu, H.-A. (2019). *A Genetic Programming Approach to Integrate Multilayer CNN Features for Image Classification*. 640–651.
- Evans, B., Al-Sahaf, H., Xue, B., & Zhang, M. (2018). *Evolutionary Deep Learning: A Genetic Programming Approach to Image Classification*. 1–6.
- Fei-Fei, L., Fergus, R., & Torralba, A. (2005). *Recognizing and Learning Object Categories*. ICCV 2005 Beijing. <http://people.csail.mit.edu/torralba/shortCourseRLOC/slides/introduction.ppt>
- Ferguson, M., Ak, R., Lee, Y.-T. T., & Law, K. H. (2017). *Automatic Localization of Casting Defects with Convolutional Neural Networks*. 1726–1735.
- Fernando, C., Banarse, D., Reynolds, M., Besse, F., Pfau, D., Jaderberg, M., Lanctot, M., & Wierstra, D. (2016). *Convolution by Evolution: Differentiable Pattern Producing Networks*. 109–116.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning* (Vol. 1, Issue 2). MIT press.
- Grauman, K., & Leibe, B. (2011). Visual Object Recognition. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(2), 1–181.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition*. 770–778.
- Iman, M., Arabnia, H. R., & Branchinst, R. M. (2021). Pathways to Artificial General Intelligence: A Brief Overview of Developments and Ethical Issues via Artificial Intelligence, Machine Learning, Deep Learning, and Data Science. *Advances in Artificial Intelligence and Applied Cognitive Computing*, 73–87.
- Ingalalli, V., Silva, S., Castelli, M., & Vanneschi, L. (2014). *A Multi-dimensional Genetic Programming Approach for Multi-class Classification Problems*. 48–60.

- Janke, J., Castelli, M., & Popovič, A. (2019). Analysis of the proficiency of fully connected neural networks in the process of classifying digital images. Benchmark of different classification algorithms on high-level image features from convolutional layers. *Expert Systems with Applications*, 135, 12–38. <https://doi.org/10.1016/j.eswa.2019.05.058>.
- Kaplanoglou, P. (2017). *Content-Based Image Retrieval using Deep Learning*. (Master's thesis, Alexander Technological Educational Institute of Thessaloniki, Sindos, Greece). Retrieved from ResearchGate. (<https://doi.org/10.13140/RG.2.2.29510.16967>).
- Konushin, A., & Artemov, A. (2021). *Convolutional features for visual recognition*. National Research University Higher School of Economics. <https://fr.coursera.org/lecture/deep-learning-in-computer-vision/recap-image-classification-VO9K7>
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (1st ed.). MIT press.
- Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs* (1st ed.). MIT press.
- Kriesel, D. (2007). *A Brief Introduction to Neural Networks*. available at <http://www.dkriesel.com>
- Krizhevsky, A., & Hinton, G. (2009). Learning Multiple Layers of Features from Tiny Images. *University of Toronto*.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25, 1097–1105.
- La Cava, W., Silva, S., Danai, K., Spector, L., Vanneschi, L., & Moore, J. H. (2019). Multidimensional genetic programming for multiclass classification. *Swarm and Evolutionary Computation*, 44, 260–272. <https://doi.org/10.1016/j.swevo.2018.03.015>.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11), 2278–2324. <https://doi.org/10.1109/5.726791>
- Li, C., Jiang, J., Zhao, Y., Li, R., Wang, E., Zhang, X., & Zhao, K. (2021). *Genetic Algorithm based hyper-parameters optimization for transfer Convolutional Neural Network*.
- Li, F.-F., Johnson, J., & Yeung, S. (2017). *Lecture 11: Detection and Segmentation*. Stanford University. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf
- McGhie, A., Xue, B., & Zhang, M. (2020). *GPCNN: Evolving Convolutional Neural Networks using Genetic Programming*. 2684–2691.
- Mitchell, T. M. (1997). *Machine Learning* (1st ed.). McGraw-Hill.
- Mohamed, H., Negm, A., Zahran, M., & Saavedra, O. (2015). *Assessment of artificial neural network for bathymetry estimation using High Resolution Satellite imagery in Shallow Lakes: Case study El Burullus Lake*. 12–14.
- Muñoz, L., Silva, S., & Trujillo, L. (2015). *M3GP – Multiclass Classification with GP*. 78–91.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., & Duchesnay, M. P. É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
- Pinos, M., Mrazek, V., & Sekanina, L. (2021). *Evolutionary Neural Architecture Search Supporting Approximate Multipliers*.
- Pretorius, K., & Pillay, N. (2020). *A Comparative Study of Classifiers for Thumbnail Selection*. 1–7.
- Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). *Regularized Evolution for Image Classifier Architecture Search*. 33(01), 4780–4789.

- Reed, R., & Marks, R. J. (1999). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press.
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386. <https://doi.org/10.1037/h0042519>
- Silva, S., Munoz, L., Trujillo, L., Ingalalli, V., Castelli, M., & Vanneschi, L. (2016). Multiclass Classification Through Multidimensional Clustering. In *Genetic Programming Theory and Practice XIII* (pp. 219–239). Springer.
- 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., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587), 484–489.
- Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*.
- Suganuma, M., Kobayashi, M., Shirakawa, S., & Nagao, T. (2020). Evolution of Deep Convolutional Neural Networks Using Cartesian Genetic Programming. *Evolutionary Computation*, 28(1), 141–163.
- Suganuma, M., Shirakawa, S., & Nagao, T. (2017). A Genetic Programming Approach to Designing Convolutional Neural Network Architectures. 497–504.
- Sun, Y., Xue, B., Zhang, M., & Yen, G. G. (2019). Evolving Deep Convolutional Neural Networks for Image Classification. *IEEE Transactions on Evolutionary Computation*, 24(2), 394–407.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). *Going Deeper with Convolutions*. 1–9.
- Szeliski, R. (2021). *Computer Vision: Algorithms and Applications* (2nd ed.). Springer Science & Business Media. <https://doi.org/10.1007/978-1-84882-935-0>
- Wendlinger, L., Stier, J., & Granitzer, M. (2021). *Evoefficient: Reproducing a Cartesian Genetic Programming Method*. 162–178.
- Willis, M.-J., Hiden, H. G., Marenbach, P., McKay, B., & Montague, G. A. (1997). *Genetic programming: An introduction and survey of applications*. 314–319. <https://doi.org/10.1049/cp:19971199>
- Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems*, 27.
- Young, S. R., Rose, D. C., Karnowski, T. P., Lim, S.-H., & Patton, R. M. (2015). *Optimizing Deep Learning Hyper-Parameters Through an Evolutionary Algorithm*. 1–5.
- Zeiler, M. D., & Fergus, R. (2014). *Visualizing and Understanding Convolutional Networks*. 818–833.
- Zhu, Y., Yao, Y., Wu, Z., Chen, Y., Li, G., Hu, H., & Xu, Y. (2018). *GP-CNAS: Convolutional Neural Network Architecture Search with Genetic Programming*.
- Zitzler, E., Laumanns, M., & Thiele, L. (2001). *SPEA2: Improving the strength Pareto evolutionary algorithm* (Vol. 103). Eidgenössische Technische Hochschule Zürich (ETH).

APPENDIX

A. VISUALIZATION OF CUSTOM CNN ARCHITECTURES

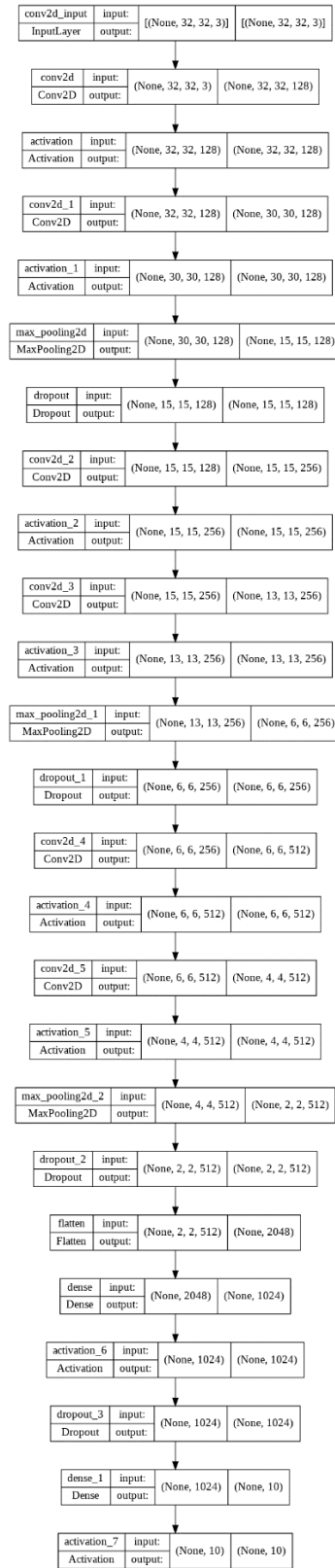


Figure A.1 – CNN 1's architecture

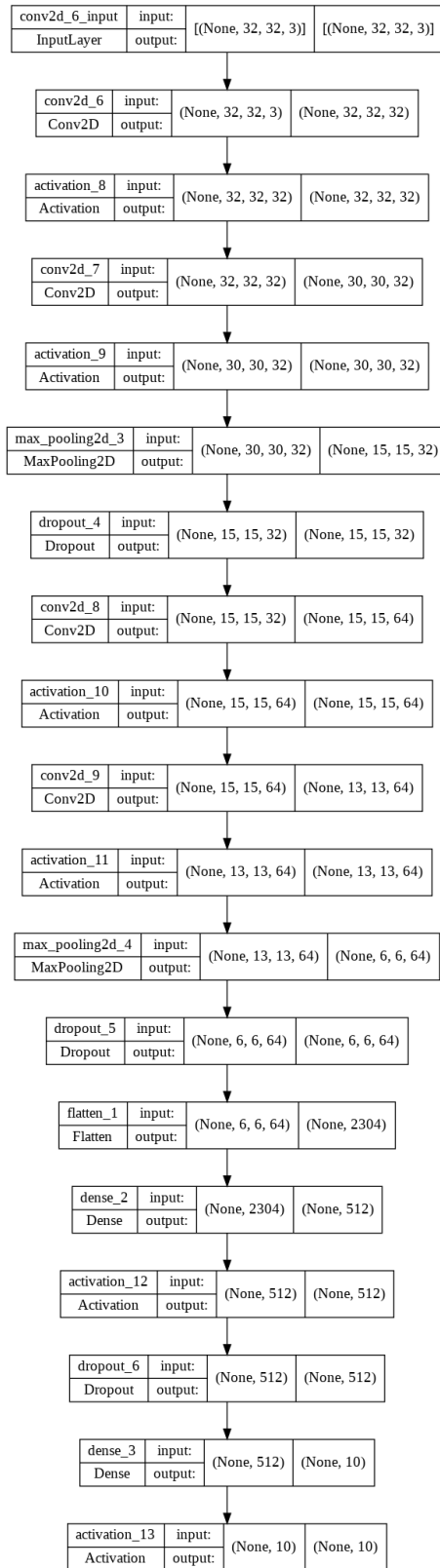


Figure A.2 – CNN 2's architecture



NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa