NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

BRUNO AFONSO COTRIM RODRIGUES

Bachelor in Computer Science

DESIGNING CONCEPT MAPPING MODELS WITH NEURAL ARCHITECTURE SEARCH

DISSERTATION

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2022

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

DESIGNING CONCEPT MAPPING MODELS WITH NEURAL ARCHITECTURE SEARCH

DISSERTATION

BRUNO AFONSO COTRIM RODRIGUES

Bachelor in Computer Science

Co-advisers: Ludwig Krippahl
Assistant Professor, NOVA University Lisbon
Matthias Knorr
Assistant Professor, NOVA University Lisbon
Ricardo Gonçalves
Assistant Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
February, 2022

Designing concept mapping models with neural architecture search

For my grandfather, Francisco Cotrim

Acknowledgements

"No man is an island", preached John Donne, delivering a sermon at St Paul's Cathedral in 1642. Even as someone who firmly believes in self-reliance as a moral imperative, I have to recognize the truth of that statement. The work presented in this dissertation would not have been possible without the help and support I received from the people around me.

Firstly, I would like to extend my sincere gratitude and appreciation to my advisers Ludwig Krippahl, Matthias Knorr, and Ricardo Gonçalves, for always being available without ever being overbearing, and for never failing to provide the kind of rigorous, pertinent criticism that pushes one to do better without discouraging him.

A big thank you is also owed to Manuel Ribeiro, whose previous work is the foundation for this one. His friendship, skill, and unrelenting eagerness to help proved absolutely instrumental in the completion of this work.

I would further like to thank my sister, Bárbara Rodrigues, who showed me the merits of the scientific pursuit. I am deeply proud of her and thankful to her for showing interest in my work, for always being available to help, and for continuously inspiring me to aim higher.

Finally, I would like to thank my family and friends for their continued interest and support. Being surrounded by good, kind people who genuinely believe in me is akin to a superpower, and I could not be more grateful. A special thank you to my mother, whom I love with all my heart, for always taking care of me.

"When you arise in the morning think of what a privilege it is to be alive, to think, to enjoy, to love ... " (Marcus Aurelius)

Abstract

Artificial neural networks are widely used in all sorts of applications, many of which directly impact the public's lives. For all of their qualities, these systems have a major flaw: their black-box nature impedes us from interpreting their behavior, which harms public trust and their overall applicability. Explainable AI is a field that focuses on developing interpretable AI systems. However, the current solutions for black-box models do not provide fully accurate or easy-to-understand explanations. Concept mapping, proposed by Sousa Ribeiro and Leite [60], promises to do both. In this method, classifiers - dubbed mapping networks - are used to map a black-box model's sub-symbolic internal representations into symbolic, human-understandable ontology concepts, opening the way to explainability. However, little investigation was done in the original work on consistently designing quality architectures for concept mapping. In this dissertation, we fill the existing knowledge gaps by conducting extensive empirical evaluation of architectures for concept mapping. We create a custom-made image classification dataset designed to facilitate observing how the black-box model's task affects concept mapping. Further, we employ a custom adaption of differentiable architecture search (DARTS [33]) to automatically find good architectures. Our adaption of DARTS for concept mapping proves capable of consistently learning exemplary architectures and shows more resilience to context changes than manual trial-and-error.

Keywords: Artificial Intelligence, Machine Learning, Auto ML, Neural Architecture Search, Explainable AI, Ontology, Description Logics

Resumo

A rede neuronal artificial tem tido vasto uso em todo o tipo de aplicações, muitas das quais têm um impacto direto na vida pública. Apesar de todas as suas qualidades, estes sistemas têm uma fraqueza crucial: a sua natureza opaca impede-nos de interpretar o seu comportamento, algo que tem um impacto negativo na sua aceitação publica e aplicabilidade. *Explainable AI* é uma área que se foca em desenvolver sistemas de inteligência artificial interpretáveis, mas muitas das soluções atuais para modelos opacos não providenciam justificações acertadas ou fáceis de entender. Mapeamento de conceitos, proposto por Sousa Ribeiro e Leite, promete ambos. Neste método, classificadores adicionais - chamados de redes mapeadoras - são criados para mapear as representações internas subsimbólicas de um modelo em conceitos pertencentes a uma ontologia: simbólicos e passíveis de compreensão humana. Todavia, pouca investigação foi feita no trabalho original sobre as arquitecturas destas peças instrumentais, as redes mapeadoras. Nesta dissertação, preenchemos as atuais brechas de conhecimento realizando extensos testes empíricos sobre arquitecturas para mapeamento de conceitos. Usamos um dataset de classificação de imagens, gerado por nós especificamente para facilitar a observação de como o mapeamento de conceitos é afetado pela tarefa do modelo original. Para além disso, usamos uma versão de procura de arquiteturas diferencial (DARTS [33]), adaptada para aprender automaticamente boas arquitecturas mapeadoras. Essa nossa adaptação prova ser capaz de consistentemente encontrar arquitecturas competentes, e demonstra uma maior resiliência a mudanças de contexto do que o método original de tentativa e erro.

Palavras-chave: Inteligência Artificial, Aprendizagem Automática, Neural Architecture Search, AI explicável, Ontologia, Lógica de descrição

Contents

Appendices

List of Figures

1

Introduction

1.1 Context and Motivation

Machine Learning (ML) systems are AI systems that learn from provided data and use it to make non-trivial inferences. The state-of-the-art of ML requires that these systems focus on highly specific tasks to achieve good results. Nonetheless, they are extremely useful. Today, ML is applied to web search results [56], image [62] and video [28] classification, virtual assistants [73], recommendation algorithms for social media websites like Youtube [11], autonomous vehicles [72], and finance [17]. Crucially, ML is used in decision support systems, which inform human decisions such as loan granting [66], job recruitment [12] and university application processing [68].

The more we rely on ML, the more considerable impact it has on our individual lives, and thus the more urgent it is that we guarantee the quality and fairness of ML-powered systems. Unfortunately, some of the most popular ML systems, namely Artificial Neural Networks (ANNs), do not justify their behavior: only the inputs and outputs are known, not the inner workings, making them opaque - black-box - models. They provide a result but no indication they produced it. The quality of such a model must be evaluated via indirect measurements, such as the accuracy of its predictions on pre-solved inputs. Amazon learned the dangers of black-box models when they discovered that the recruitment system they used showed a strong bias against women. Amazon used the system to extract the top candidates from a provided set of applications. However, it was trained on the previous ten years of the company's recruitment data, which featured a majority of males. Consequently, the system inferred that female candidates were undesirable and penalized applications for including the word "women's" and further penalized candidates from two all-women colleges [12]. It is not easy, however, to assess whether an outcome disparity is caused by system flaws or extraneous factors that would have resulted in disparities in even an entirely fair system. The black-box nature of some Machine Learning Algorithms prevents us from simply "peeking" inside a system to look for answers.

This ethics question is but one of many concerns over black-box models. Knowing

exactly why a network reached a particular conclusion is also helpful during development for diagnosing problems. In many cases, it is not enough to measure a network's accuracy, as they can produce mostly correct results by using incorrect methods. This is poignantly illustrated by how several machine learning models, including neural networks, consistently misclassify adversarial examples - inputs formed by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence [38]. In these cases, the network has extracted incorrect secondary features that happen to be statistically correlated to relevant ones in the dataset and mistakenly used them to achieve good performance without actually learning valuable representations of the data. For example, let us imagine a dataset for image classification where the vast majority of images containing dogs feature the animal laying on a green lawn, while pictures that do not include dogs rarely include a green lawn. Given this dataset, a model will likely learn to classify any image featuring a green lawn as one containing a dog. By looking at the model's accuracy, the developer would be oblivious to the problem, and would only later discover it when he asked the model to classify, for example, an image of a cat laying on a lawn and received an incorrect output.

Explainability - the ability of a system to justify its behavior - is also a big concern when deploying AI systems in a public setting. A survey of 6054 citizens from the United States, Canada, Germany, the United Kingdom, and Australia showed that only about a quarter of citizens are willing to trust AI systems [21]. In the healthcare field, when it comes to diagnosis systems, explainability does not solely increase trust but is also highly useful when deciding treatment routes [30]. Indeed, years ago, physicians ranked explainability as the most desirable feature for a clinical decision support system [30, 64].

All of these concerns gave rise to Explainable AI (XAI), a field that focuses on developing human-interpretable AI systems - i.e., systems whose behavior is capable of being understood by humans [14]. However, state-of-the-art XAI solutions for interpreting black-box models fall short of the promise of a fully interpretable ANN. Some provide a simpler, usually interpretable model that, presumably, behaves in a similar way to the original one [47]. Others are limited to outlining which parts of the input were most relevant to the model's decision [40], still failing to indicate why. Evidently, these methods merely provide a means by which a human expert can acquire, at best, a partial understanding of a model's behavior. This dependence on a human interpreter makes them unsuitable for non-expert users, which is why they have not seen much use outside ML research itself.

1.1.1 Explainability via Concept Mapping

Given the shortcomings of the current Explainable AI solutions, Sousa Ribeiro and Leite [60] proposed a novel method aimed at producing symbolic justifications for the output of an ANN.

Let us informally illustrate the logic behind this method with an example: Alice is trying to teach her son Bob to correctly identify the animals in a children's picture book. When Alice points at an animal, Bob blurts out which one he thinks it is and eagerly sees his mother's reaction to know whether his guess is correct. After some time, Bob has become rather good at this task, but because he is still young, he can not yet explain the reasoning behind his guesses. However, when he identifies them in the picture, Bob has a habit of pointing out other words he has learned - like 'ears', 'tail', or 'whiskers'. Paying attention to this, Alice can easily infer Bob's reasoning. For example, when prompted with an image of a cat, Bob says 'whiskers... ears... Cat!', and Alice is able to infer that Bob knows that any animal with whiskers and ears must be a cat (Bob has not learned the Lince yet).

The same logic can be applied to black-box image classification models. If one gets an indication of what secondary features the model found for a given example, one can infer a possible reasoning that the model may have followed. Sound as that logic may be, there were two missing pieces. Firstly, note that Alice is only able to make reasonable inferences about Bob's reasoning because she is knowledgeable about the animals - i.e., the domain of the problem. Secondly, unlike Bob, black-box models don't output any of their intermediary findings.

Sousa Ribeiro and Leite solve the first missing piece with ontologies, sets of formalized articles of knowledge (axioms) pertaining to a given domain. Ontologies describe concepts, their properties, and the relations between them. For example, an ontology could define the concept 'woman' using the concepts 'human' and 'female'. The use of an ontology is fitting for this application, as there exist highly well-made, accessible, and off-the-shelf software pieces called reasoners that can infer non-trivial knowledge from ontologies and, like Alice does, automatically produce justifications for the model's behavior. As a bonus, the explanations produced by a reasoner can easily be translated into natural language, allowing laypersons to directly understand a produced justification without an expert's help.

To understand how to solve the second missing piece (knowing what intermediary attributes the model found in an input), one needs to understand the basics of how ANNs perform classification. After receiving an input, such a model applies sequential transformations to the input until a final one results in a classification. These transformations are optimized during training to maximize the model's performance. While the final transformation outputs something humans can understand, the intermediary representations of the input are incomprehensible. That is not to mean, however, that they are random. For the model to correctly classify most of the inputs, those representations must contain information that is somehow relevant to the task. Knowing this, Sousa Ribeiro and Leite employ additional classifier models - the mapping networks - which they train to translate those incomprehensible internal representations into the relevant concepts that were previously defined in the ontology. In practice, each mapping network specializes in mapping a single concept.

Finally, by leveraging the knowledge contained in the ontology with the knowledge of what relevant concepts the main network found in an input, a reasoner can automatically generate plausible justifications for the main network's behavior. In our example scenario from before, a justification might tell us something like "The main network found that the input contained the concepts 'human' and 'female'; therefore, the main network classified the input as 'woman'. Note how the reasoning indeed depends on both the concepts found in the input and the axioms established in the ontology.

The mapping networks are the central piece of the method, extracting symbolic meaning from a subsymbolic network and opening the way to explainability. Therefore, whether their architecture is optimal heavily influences the method's usefulness. For example, it is a well-known issue that neural networks can achieve high accuracy while training by learning incorrect features that, by chance, are statistically correlated in the training set (overfitting). The result is that the network will perform much worse when faced with examples outside the training set; a worrisome prospect when deploying such a system for real-world use. One of the uses for explanatory methods is to diagnose these types of mistakes that are not evidenced by observable metrics such as accuracy. In the concept mapping method, if a mapping network is unable to learn to map a particular concept, one may conclude the main network is not using that concept in the classification task, which could be an indication of incorrect learning or incorrect assessment of what concepts are relevant to the task. However, if the architecture of the mapping network is not optimal, then that, and not one of those other factors, might be the reason for its inability to map the concept. This ambiguity severely hinders the usefulness of the explanatory method. In order to make sure that all other variables are as controlled as possible and that no observed phenomenon is of ambiguous origin, it is paramount to optimize the architecture of the mapping networks. To further emphasize this point, let us note that since the produced justifications can depend on all the identified concepts, it is enough for one of the mapping networks to produce an inaccurate output for wrong justifications to be produced. As Sousa Ribeiro and Leite [60] point out, if one were to use 21 mapping networks, each producing accurate results 90% of the time, the probability of all being simultaneously correct would be $0.9^{21} \approx 10.9\%$. Indeed, in one of the tests carried out in [60], for one of the outputs, only in 54.7% of the examples were all justifications with the highest degree of confidence correct, and for 10% of the examples, all produced justifications were incorrect.

During the work carried out in [60], two architectures were considered for the mapping networks: one with a single output neuron and one with two dense layers with ten and five neurons before the output neuron. These very simple architectures were found to be the best performing experimentally. However, that does not mean that this will always be the case, as the nature of the main network's task and its architecture can significantly affect the concept mapping task. Further, the original work's main network's task was image classification on images of train drawings over an abstract background (Fig. 1.1). The main network's task is to output which type of train is present in a given input,

Figure 1.1: Sample images from the XTRAINS dataset [60].

Each drawing is imposed on top of a random image to increase the complexity of the classification task. The used ontology describes three types of trains - Type A, Type B, and Type C. For example, in plain English, a train of type A is defined as: "trains having either a wagon with at least a circle inside and a wagon with double walls on each side or no wagons with geometric figures inside them" [60]. Types B and C are defined in similar terms.

which is defined according to the types of wagons it tows, and the geometric figures inside of them. The XTRAINS dataset is not representative of standard image classification datasets such as CIFAR10 and ImageNet. Firstly, the same features always have the same appearance: circles, triangles, and types of wagons always use the same sprite. This is not the case in other image classification datasets, which contain thousands of variants of the same class, distinguished by different shapes and colors despite their common defining attributes. Further, the train is always fully contained inside the frame. Finally, the trains do not exist in a 3D environment, making it impossible for features to appear from different angles and perspectives. Sousa Ribeiro and Leite [60] also tested the concept mapping framework on a traffic sign dataset, to which all previous criticisms equally apply. Evidently, the results obtained by the mapping networks in [60] are insufficiently generalizable, making further research imperative.

1.2 Problem Statement and Contributions

For the Concept Mapping framework to be usable and applicable to real-world scenarios, a deeper analysis must be conducted on how to build exemplary concept mapping architectures. That is what we achieve in this dissertation. Accordingly, here are the core aspects of our methodology:

- **The Main Task:** We deploy the explanatory method on a set of datasets for the same classification task, which vary only in how hard they are to classify. Doing this permits a deeper analysis of the concept mapping task: how it changes based on the complexity of the main task and how the optimal architecture of the mapping networks itself changes. This flexibility allows a better understanding of what mapping architectures to use in different scenarios. The datasets consist of synthetic data generated by us to ensure control of all variables. The images consist of snapshots of a 3D environment generated using the software Blender [18]. Similar to XTRAINS, each image is not solely labeled with its respective main class but also with other relevant concepts given by an accompanying ontology. Additionally, the dataset generator supports custom ontologies, generating images and labels in accordance with any provided valid ontology.

- **The Architecture Search:** Methodically finding what neural architecture best suits each concept mapping task involves combing through a search space that is too large to fully explore by trial and error, and limiting ourselves to a set of pre-determined architectures introduces our own biases and severely limits the exploration. This is why we propose Neural Architecture Search (NAS). NAS systems are able to autonomously discover and test novel neural architectures, adapting until they find the best performing one for a given task. Notably, concept mapping is not a typical machine learning problem. So there is not much research work and knowledge behind the optimal architecture for it, in contrast to image classification, for example. Using NAS provides a way to search more effectively and efficiently than the usual trial-and-error method used by a human researcher. NAS is also more likely to find an out-of-the-box solution, as it is not affected by the human bias of doing things in known ways. We work to test the hypothesis that NAS will prove to be useful for automatically fine-tuning the mapping networks for each particular explanation task, thus improving the applicability of Concept Mapping in real-world scenarios.

The main goal of this dissertation is to pave a road to consistently good concept mappers; computational artifacts capable of translating from a language that no human speaks. These models are our eyes into the exciting workings of the Deep Neural Network. They represent a necessary step on the path to a world where that technology is put to the betterment of our individual and public lives. We saw that Neural Architecture Search

represents a promising candidate for achieving that goal, and we set out to test that hypothesis.

Accordingly, here are the main contributions made in this dissertation:

- Empirical evaluation of concept mapping architectures, using systematic automated search methods and applied to varying scenarios.
- A NAS implementation, custom-fit for learning concept mapping architectures.

In addition, we make the following contributions:

- Variable-complexity image classification dataset, ideal for research on how the task's complexity affects computer vision.
- Ontology-independent generator for the dataset above.

1.3 Document Structure

This dissertation is divided into seven chapters. We are currently in the introductory chapter. Chapter 2, "Background", introduces the prior knowledge needed to read the rest of the document. We discuss the basics of artificial neural networks, gradient descent, and authoring ontologies with description logics. Chapter 3, "State of the art", goes into deeper detail on the current literature on the specific technologies this dissertation focuses on. We detail how concept mapping works and our chosen NAS algorithm, DARTS. Chapter 4, "The VCB dataset" centers on the dataset that was created for this dissertation. We discuss how we introduce more complexity into the classification task, the underlying ontology we used for our experimentation, and the practical details surrounding generating the images and labels. Chapter 5, "Empirical evaluation" details the methodology used in the experiments carried out in this dissertation. We discuss how we met the pre-conditions for running the experiments - adapting DARTS, generating the datasets, training the main models - and provide detailed descriptions of the experiments we ran. Chapter 6, "Results" contains a thorough discussion of our findings. We visualize the gathered data along different axes and aggregate the insights gained. Chapter 7, "Conclusion", summarizes the results and contributions made in this dissertation, and contains a discussion of possible future research avenues.

In addition, Appendix A contains the raw data from our experiments, in table form.

2

Background

This chapter presents a review of the prior knowledge relevant to the project.

2.1 Machine Learning

An Artificial Intelligence system is a system that leverages computational power in order to mimic human capabilities such as learning, decision making, and problem-solving [15]. Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) are often used interchangeably, so it is important to distinguish between them here. DL is a sub-field of ML, which is a sub-field of AI. While AI is the umbrella term for the concept of a computer mimicking human cognitive abilities, ML describes AI algorithms that improve with data and learn to perform tasks without having to be given explicit instructions. DL describes the subset of ML algorithms powered by models with multiple layers of non-linear transformations, almost always Deep Neural Networks (*DNNs*). These neural networks can extract relevant features from unstructured data, making them less reliant on prior human intervention than other ML algorithms.

Common problems in ML include Classification, Regression, Clustering, Association Rule learning, and Ranking [9]. Classification is the attribution of one (or more, in the case of *Multilabel Classfication*) out of N labels (such as dog, horse, or human) to a given input. Regression is the prediction of a numerical value, for example, *House Price*, given an input, for example, *Size*, *Location*. Clustering is the automatic grouping of inputs, for example, grouping similar retail customers by their profiles. Association rule learning is the formulation of rules that can be used to infer knowledge, for example, rules that associate a customer's purchases with products they are likely interested in. Ranking is a regression problem that returns the position of a given input on a given scale, for example, ranking search results on relevance upon a web search. When it comes to the Classification, Regression, and Ranking problems, the models require pre-solved examples to learn from. For example, in an image classification problem, each example consists of an image and its respective class. This sort of approach is called supervised learning. The usefulness of supervised learning models comes from their ability to, after

training, generalize and produce accurate results for unseen, unsolved examples. As for Clustering and Association Rule Learning, the models learn solely from the input data. This is because no particular answer is known or expected beforehand. In contrast to supervised learning models, the usefulness of these models comes from what they learn from the training set itself (Clusters, Association Rules). This is called unsupervised learning. The models focused on in this dissertation are supervised learning models; therefore, we will mainly elaborate on that particular paradigm.

2.1.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are inspired by the human brain, where hundreds of billions of interconnected neurons process information in parallel [69]. For a long time, these models have been popular solutions for classification, clustering, pattern recognition, and prediction in many disciplines. They distinguish themselves from other ML algorithms (such as support vector machines, logistic regression, decision trees, and naïve Bayes) by their ability to learn complex non-linear representations of unstructured data.

ANNs are built from smaller units called neurons (or nodes). A conventional ANN contains neurons organized in layers: Neurons in a given layer receive data from the neurons of the prior layer and output data to neurons in the following one. The first layer, which receives the data from the user, is called the input layer. The last layer, which outputs a human-understandable result (A classification, prediction, clustering, etc.), is called the output layer. The layers between the input and output layers are called hidden layers. An ANN, however, can be as simple as a single neuron.

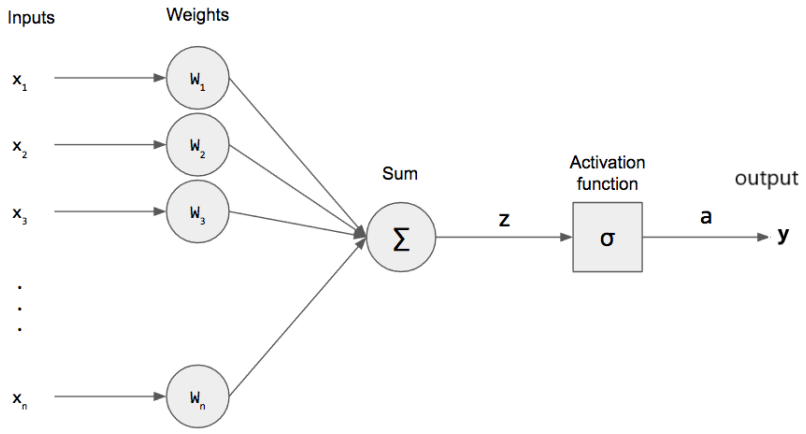


Figure 2.1: Model of a Neuron, showing the weighted sum of the inputs and the Activation Function.

Biological neurons can receive stimuli from multiple other cells, including other neurons. If a given stimulus is higher than a certain threshold, the neuron fires an impulse

that propagates to connected cells and neurons. Artificial Neurons function similarly, in a broad sense. A neuron receives several inputs, which can either be provided by the user, in the case of the neurons in the input layer, or otherwise by other neurons. Then, the neuron performs a linear combination of the input via a weighted sum, to which a bias value is added. The neuron's output is a function of that sum, which only activates above a certain threshold. This function is called the Activation Function. To illustrate, a model of a neuron is presented in Figure 2.1.

2.1.1.1 Training a Neuron with gradient descent

To start this section, let us review some definitions and notation here. We will be discussing the gradients of scalar-valued differentiable functions. Scalar-valued denotes a function with n variables that returns a scalar value for every point in n-dimensional space. Differentiable represents a function whose derivative exists at each point. So, the gradient of a scalar-valued differentiable function f with n variables is itself a function that, for any given point p in n-dimensional space, returns a vector with the partial derivatives of all of f's variables at p. The vector returned by the gradient for a point p can be relied upon to point to the direction of steepest ascent for f. For the sake of brevity, we will not prove that, but, as we will discuss briefly, this property is the basis for gradient descent. The partial derivative of a function f with respect to one of its variables is the result of treating all other variables as constants and calculating the derivative. The gradient of a function f is denoted with ∇f, and the gradient's component that corresponds to a variable x is denoted $\nabla_x f$. Notably, that component is the partial derivative of f with respect to x, which can also be written like $\frac{\partial f}{\partial x}$. The full derivative of a function f with only one variable x can be written like $\frac{df}{dx}$. The latter notation is primarily used to formalize mathematical rules. As a summary, we can write the gradient of a function f with n variables at point p as:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x1}(p) \\ \frac{\partial f}{\partial x2}(p) \\ \vdots \\ \frac{\partial f}{\partial xn}(p) \end{bmatrix} = \begin{bmatrix} \nabla_{x1} f(p) \\ \nabla_{x2} f(p) \\ \vdots \\ \nabla_{xn} f(p) \end{bmatrix} \tag{2.1}$$

A Neuron is trained by adjusting the weights of the linear combination of its inputs and the bias value in order to approximate the desired outcome [1]. Examples are presented to the neuron randomly, and the weights are iteratively adjusted to minimize a given loss function.

Since the loss is a scalar-valued differentiable function of the weights, it is possible to compute its gradient with respect to the weights. If we consider a set of weights a

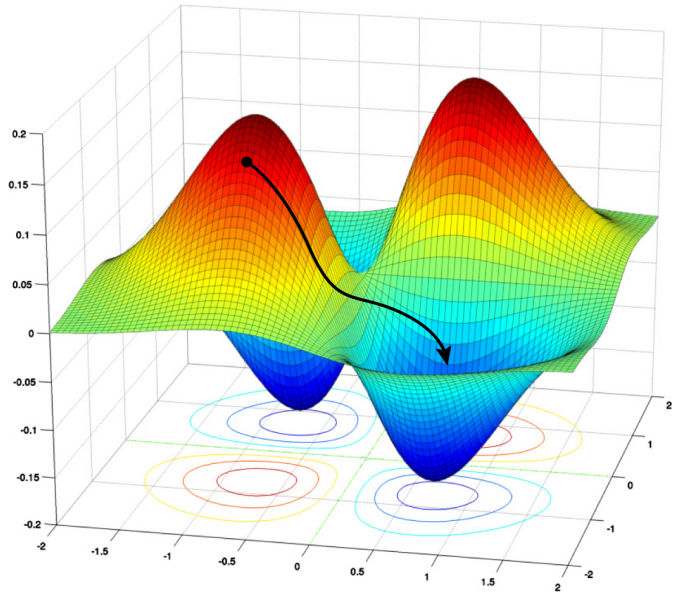


Figure 2.2: A visual representation of Gradient Descent for a two-variable function.

The two variables exist in the horizontal plane, and the graph's height represents the value of the function. The black arrow shows the possible trajectory of a Gradient Descent algorithm. Note how the algorithm stops at a local minimum and whether it finds the global minimum depends heavily on the starting point.

point in n-dimensional space, the gradient of the loss at that point is the direction of steepest ascent for that function. Thus, each weight is adjusted in order to take a step in the opposite direction, effectively minimizing the loss function. This process is called Stochastic Gradient Descent. One can intuitively visualize this process by imagining a hiker on a mountain who wants to go to its lowest point. However, it is very foggy, and the hiker can only see a few meters in every direction. The hiker might devise the following plan: look around, take a couple of steps in whichever direction has the steepest downward slope, reassess, and repeat. Notably, following this protocol, the hiker would not be guaranteed to find the absolute lowest point in the mountain. It will suffice to imagine that he follows a downward slope that leads into a vale, which, despite being surrounded by steep inclines on every side, is nonetheless quite high up in the mountain. At that point, the hiker, realizing that a step in any direction would mean an increase in altitude, would have no protocol to progress. Gradient descent, just like the hiker's strategy, does not guarantee that it finds a global minimum, but only a local one (Fig. 2.2).

The Learning Rate determines the step Gradient Descent takes, and it must be small for the process to work. After taking even a small step in some direction, the gradient changes and must be re-calculated to accurately assess the optimal direction to head in. While a large learning rate leads to faster progress in the early stages of learning, it can severely hinder convergence in the latter by 'overshooting' the local minimum (Fig. 2.3).

Since each neuron performs a linear combination of the inputs, a single Neuron network can only learn (i.e., approximate) a linear function. In other words, a single Neuron

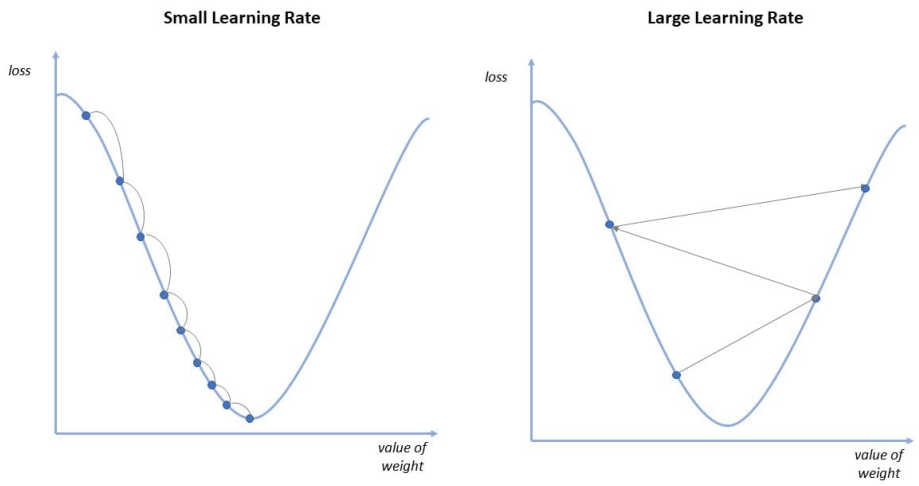


Figure 2.3: A simplified visual representation of the negative effects a large learning rate can have on gradient descent.

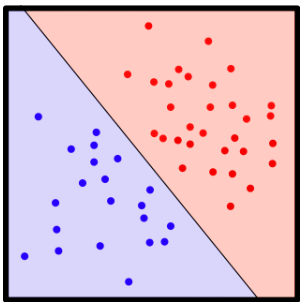


Figure 2.4: Linearly separable classes [32]

is only capable of learning to distinguish between linearly separable classes. Two classes are linearly separable if there exists at least one line (plane in 3 dimensions and hyperplane above that) which contains all elements of each class on different sides (Fig. 2.4).

2.1.1.2 Multilayer Perceptron

To create a model capable of distinguishing between classes that are not linearly separable, we need to have at least one non-linear transformation of the data before a linear classification. This way, the model learns a transformation of the input that makes the classes linearly separable, and able to be correctly classified by a linear classifier.

This nonlinearity is achieved by passing the output of a neuron through a non-linear activation function before feeding it to the next neuron. In more complex ANNs, neurons are often organized in a sequence of layers, where each layer outputs to the one in front through an activation function. When a network exhibits this unidirectional flow of information, it is called a Feed-forward Neural Network.

A commonly used Feed-forward Neural Network is the Multilayer Perceptron (*MLP*).

The *MLP* is a fully-connected network (also referred to as Dense), which means any given neuron receives, as input, the output of every neuron in the previous layer (Fig 2.5).

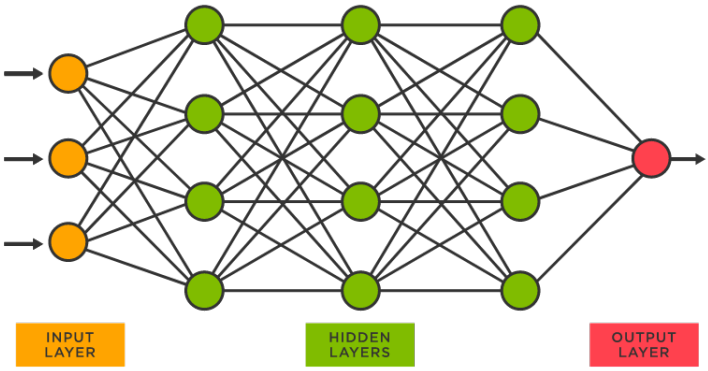


Figure 2.5: Model of an MLP, showing five fully connected layers [71]

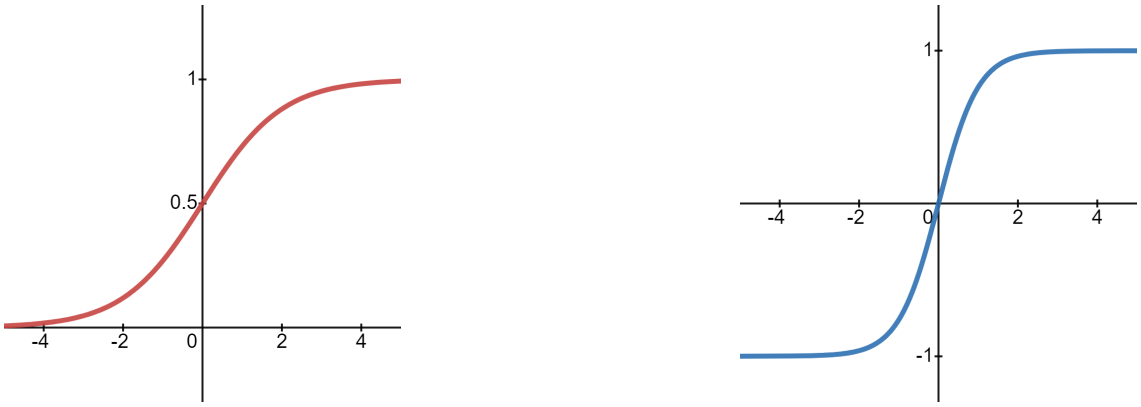
2.1.1.3 Backpropagation Algorithm

Stacking multiple layers of neurons, however, introduces the interesting question of how to train the network. We have seen that a single neuron is trained by adjusting its weights according to its output to minimize the loss function. In a multi-layer network, only the neurons in the last layer output values that can be compared to a target, so a protocol for updating the rest of the network has to be developed. The updating process starts at the output layer, and the weights of each of its neurons are updated just as before. Regarding the second to last layer, its neurons are connected to neurons in the output layer, meaning that they influence their computation. Thus, we can transfer - or propagate - the loss from each output neuron to the neurons in the previous layer which are connected to it. The loss of a neuron in a hidden layer is given by the sum of the losses of all the neurons to which it outputs. In this sum, the losses are weighted according to the coefficients of the respective neurons in the following layer (i.e the weight they attribute to the neuron's input). After computing the loss value of a neuron, its weights can be updated as usual. This method is iteratively used to update the network's weights from the neurons in the output layer all the way back to the ones in the input layer. This is called the Backpropagation Algorithm.

And so we arrive at the Feed-Forward Backpropagation Algorithm, a protocol for training a neural network. To execute this algorithm, one starts by presenting a random example to the network and passing the output of each layer to the next until reaching the output layer, which produces the desired result (Feed-Forward). Then, this result is compared to the target result for the given input, and the loss value is computed for the output layer and propagated back through the network (Backpropagation). Finally, the weights are adjusted in such a way that it minimizes the loss using Gradient Descent. This process can be repeated indefinitely.

2.1.1.4 Activation functions

Despite the popularization of the backpropagation algorithm in 1989[51], ANNs did not see widespread use for a long time, in large part due to the formerly widely used Logistic (Sigmoid) and Hyperbolic Tangent (Tanh) activation functions. Notably, these functions saturate at high and low enough values (Fig. 2.6). This means that the gradient tends towards zero outside of a small area near zero, making it hard to use gradient descent to discern which neurons contain valuable information (and adjust the weights accordingly).



(a) (b)

Figure 2.6: The Logistic (Sigmoid) a) and Hyperbolic Tangent (Tanh) b) activation functions.

The Sigmoid function tends towards zero for negative values and one for positive values, while the Tanh tends towards minus one for negative values and one for positive values. In both cases, the gradient tends towards zero at the edges.

This vanishing gradient problem, as it is called, was solved by the Rectified Linear Unit (ReLU). Despite making its first appearance back in 1975 [19], recent interest in the use of ReLU in neural networks was only spurred in 2010 when Nair and Hinton [34] used it to improve Boltzmann Machines. The ReLU function is constant at zero for negative inputs, and is an identity function for positive inputs (Fig. 2.7). Despite its simplicity, the ReLU solves the vanishing gradient problem as its gradient is constant and greater than zero for all active neurons. Further, it is cheap to compute both the function and its gradient. The Sigmoid and Tanh functions, for example, use the exponential function, while ReLu, as shown, only requires the max function. The Leaky ReLU (Also shown in Fig. 2.7) is a ReLU variant that ensures a non-zero gradient for both positive and negative values. This is meant to solve a problem with the original ReLU, where weights that turn to negative values during training stop being updated since their gradient becomes zero (the 'Dead Neuron' problem).

The Sigmoid function, despite no longer being present in the hidden layers of ANNs, is still often used in their output layer. This is due to the desirability that an ANN output a value between zero and one so that it can be interpreted as the probability that the input belongs to a given class. Thus, the Sigmoid function is commonly used for binary

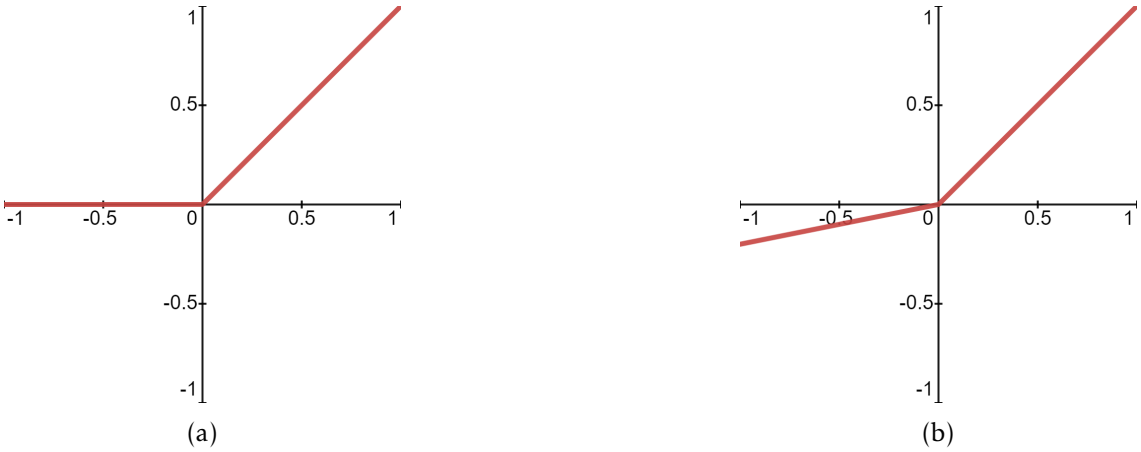


Figure 2.7: The ReLU a) and Leaky ReLU b) activation functions.

Two piece-wise linear functions. The ReLU is constant at zero for negative values, while the Leaky ReLu exhibits a slight slope. Both are an identity function for positive values. The ReLU can be written as $f(x) = max(0, x)$ and the Leaky ReLy as $f(x) = max(ax, x)$ where a is positive and lesser than 1 ($a = 0.2$ in this particular representation).

classification - i.e., a classification task with two classes - and multilabel classification - i.e., a classification task with n classes and no restriction on the number of classes that a given input may belong to. However, the Sigmoid function is not adequate for single-label classification - i.e., a classification task with n classes, each input necessarily belonging to a single one - because, in those instances, one wants a probability distribution of the membership of the input over all possible classes. So, single-label classification tasks use the Softmax function at the output layer. The Softmax function converts a vector of K real numbers into a vector of the same size with positive numbers between zero and one that add to one, i.e., a probability distribution over K possible outcomes. The following formula gives the Softmax function:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.2}$$

2.1.1.5 Convolutional Neural Networks

Other than the MLP, there are numerous types of neural networks, defined by the type of layers that compose them. Convolutional Neural Networks (CNN) are one such type worth mentioning since they excel at Computer Vision, a field of AI that focuses on extracting useful information from images. Images are unstructured data, which, summarily, means that between different images, the same pixel can not be interpreted as containing the same information. Two pictures of dogs, for example, can feature their subjects in entirely different positions in the frame. This is in contrast to structured data, such as a table of values, in which the same column in two separate rows contains values pertaining to the same feature or concept. For that reason, the MLP is inept at computer vision. It could only learn to distinguish between very uniform images where the subjects

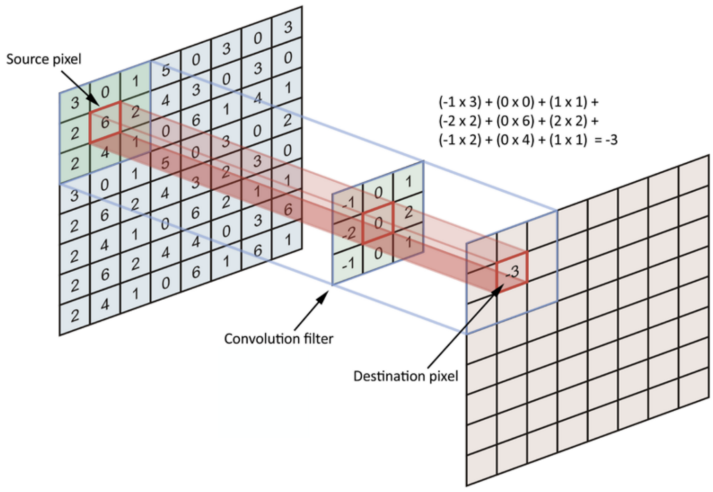


Figure 2.8: Representation of the filter operation of a Convolutional Layer [10]

are always featured in the same place. The MLP also has no sense of locality: how a pixel in a row is a neighbor of a pixel in the row below. For the MLP, the image is a one-dimensional vector, and those two pixels would be vastly separated. Convolutional Layers, on the other hand, can process two-dimensional data and preserve information about the distance between pixels. These layers apply filters to the input by sliding a kernel - a matrix of weights of a much smaller size than the image itself (e.g., three by three pixels) - across all of its pixels. Computing the value of a pixel in the output of a convolutional layer consists of centering the kernel on the same pixel on the input image, multiplying the values of the covered pixels by the corresponding kernel weight, and adding the resulting values (See Fig. 2.8). Applying the same kernel to all the input image's pixels allows for features to be extracted irrespective of where they are in the input. The weights of the kernels are the trainable parameters of convolutional layers. A typical convolutional layer comprises tens, hundreds, or even thousands of filters.

Convolutional Layers are great at extracting features from images, but it is usually still advantageous to use fully-connected layers to learn a classification from those features. Thus, a typical CNN consists of a stack of convolutional layers followed by a fully connected classifier. The first convolutional layers learn to extract basic features such as contrast, straight lines, and basic shapes. The deeper convolutional layers are able to learn more complex features by leveraging the previously extracted ones. Finally, all the information that the convolutional layers learned about the input is handed to the dense section to produce a result. Since each filter extracts a feature from the input, irrespective of its position, it is desirable to progressively reduce the resolution of the filters so as to end up with unit indicators of whether a particular feature was found anywhere in the input. To that end, Pooling layers are often used between convolutional ones. Pooling layers do not contain any trainable parameters, and their only function is to reduce the resolution of the feature maps, which they do by merging - pooling - a neighborhood of pixels into a single value. The two most common pooling methods are average pooling

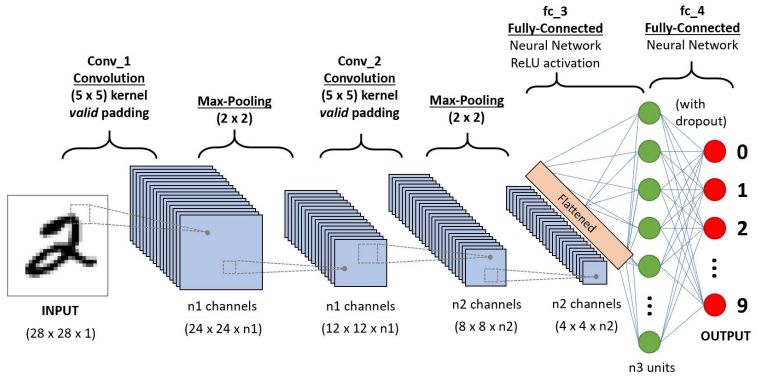


Figure 2.9: Overview of the architecture of a typical Convolutional Neural Network [10]

- replacing a neighborhood of pixels by the average of its member's values - and max pooling - replacing them with the member with the highest value. Before feeding the data to the dense section, the two-dimensional feature maps of the convolutional section of the network are flattened - i.e., transformed into a one-dimensional array of values.

2.1.2 Reinforcement Learning

In addition to supervised and unsupervised learning problems, there is a class of problems - Reinforcement Learning (RL) - that consists of optimizing the decisions taken by an agent in a given environment. A popular use case of reinforcement learning is software for self-driving vehicles. Reinforcement learning models, like supervised ones, receive feedback for their output and change accordingly. However, there is a fundamental difference: in RL, there is not a static set of data. Instead, the agent's observations of the effect of its actions in the environment are the data. Further, there is no known correct action to take in any given circumstance. Instead, after performing an action, the agent may receive a reward, which can be negative if the action produced an undesirable outcome (e.g., hit a wall) or positive if it completed, or at least advanced towards, a stipulated goal (e.g., reached destination successfully). This reward is produced by an interpreter responsible for providing feedback to the agent according to the environment's state. The training of a reinforcement learning model consists of repeating an action-response cycle, from which the agent slowly learns. This cycle can be summarized as follows:

1. The agent performs an action.
2. The action affects the environment.
3. The agent observes the environment and may receive a reward that reflects the quality of that action with respect to a goal stipulated by the user.
4. The agent adjusts its decision-making algorithm based on the received feedback.

We provide a visual representation of this framework in Figure 2.10.

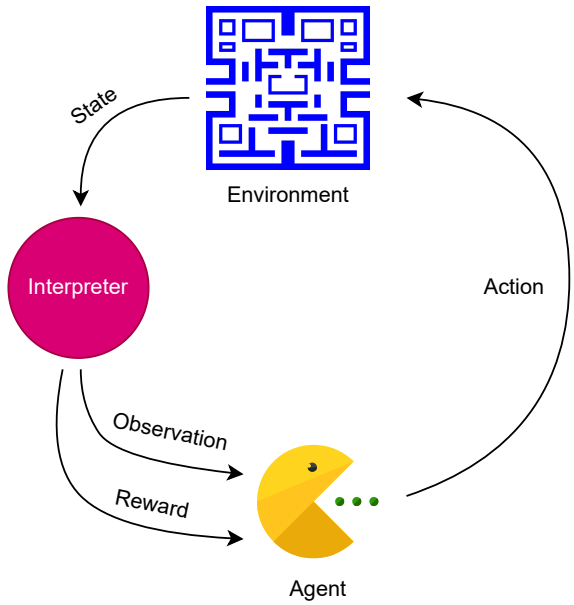


Figure 2.10: The general framework of Reinforcement Learning

Two core concepts of RL are exploration and exploitation. In the early stages of training, it is desirable that a RL algorithm experiment with different actions and strategies - Exploration - while in the latter stages, it should use what it has learned to become progressively better at the task - Exploitation. For this reason, RL algorithms often include exploration-stimulating mechanisms that gradually weaken as the training progresses. For example, some algorithms make the agent perform a random action with a certain probability, which is slowly decayed to zero.

2.2 Description Logics

Another notable area of AI is knowledge representation and reasoning (KRR) - the study of representing information in a way that a computer can use to perform complex inference and reasoning tasks. Ontologies - formal representations of a domain of knowledge - are integral parts of KRR. To conceptualize a given domain of interest, ontologies make use of individuals (singular entities), concepts (sets of individuals), and roles that individuals can occupy. Ontologies are sets of assertions describing these building blocks and how they relate to each other.

In the field of KRR, formal languages for authoring ontologies - Description Logics - are used. To represent the knowledge present in a given domain of interest, Description Logics first define the concepts of that domain (its terminology) and then use those concepts to specify the properties of individuals present in that domain [3]. These languages build on formal, logic-based semantics and ensure that implicit knowledge about concepts and individuals in the domain can be automatically inferred by reasoners from the knowledge provided explicitly. These properties are why Description Logics see wide

use in the field of Artificial Intelligence. Description Logics vary in their expressiveness, i.e., what constructors can be used to write axioms. More expressive Description Logics require more resources to reason over than simpler ones, so when choosing a Description Logic, one should avoid choosing one with more expressiveness than one needs.

Description Logics are derived from First-Order Logic (FOL), a set of formal systems commonly used in mathematics and other fields. FOL includes logical symbols such as universal , $\forall$, and existential, $\exists$, quantifiers, disjunction, $\vee$, conjunction, $\wedge$, implication, $\rightarrow$, equivalence, $\leftrightarrow$, negation, $\neg$, variables, e.g. x, y, z, and equality, $=$. Further, FOL offers predicate symbols to denote a property of a variable ($Woman(x)$ meaning that x is a woman), and function symbols ($father(x)$ returning the father of x). First-Order Logic, for how expressive it is, is not decidable, meaning that no algorithm is guaranteed to solve any true-false decision problem written in FOL in a finite number of steps. Description Logics, by supporting only fragments of FOL, are usually decidable. This characteristic is paramount if we wish to reason automatically over an ontology.

2.2.1 The $\mathcal{ALC}$ description logic

$\mathcal{ALC}$ - Attributive Concept Language with Complements - is a description Logic introduced by Schmidt-Schauß and Smolka [54], and it has served as a basis for many subsequent and more expressive description logics. For that reason, it includes all of the standard and most common constructs - and indeed all of the ones relevant to this dissertation - so let us review it here, according to the definitions provided in the "Handbook of Knowledge Representation" by Santiago [52].

A description logic is defined by its syntax (i.e., the set of legal symbols and expressions) and its semantics (i.e., the meaning of each symbol).

Let us start with the syntax: Let N_C be the set of concept names, N_R be the set of role names, N_O be the set of individual names, R be a role name ($R \in N_R$) and C and D be $\mathcal{ALC}$ concept expressions. The following are all valid $\mathcal{ALC}$ concepts:

$\top$	top (universal)
$\bot$	bottom (contradiction)
$A \in N_C$	all atomic concepts
$C \sqcap D$	the intersection of two concepts
$C \sqcup D$	the union of two concepts
$\neg C$	the complement of a concept
$\exists R.C$	the existential restriction of a concept by a role
$\forall R.C$	the universal restriction of a concept by a role

The two major elements present in all Description Logics are the *TBox* and *ABox*. The TBox introduces the domain's concepts, their properties, constraints, and how they relate to each other. The *ABox* contains assertions about individuals. Thus, the *TBox* describes a model of a world (i.e., the ground rules), and the *ABox* an instantiation of that model.

The *TBox* is a finite set of general concept inclusions (CGI), which can take the following forms:

$C \sqsubseteq D$	inclusion
$C \equiv D$	equivalence (pair of symmetrical inclusions)

For example, the *TBox* may contain the expression:

$Mother \equiv Woman \sqcap \exists hasChild.Person$

"All individuals that are women and have a child are mothers (and vice-versa)"

The *ABox* is a finite set of assertional axioms, which can take the following forms:

C(a)	concept assertion (a is a C)
R(a,b)	role assertion (a is R-related to b)

where a and b are individual's names (a,b $\in N_O$). Continuing the example, the *ABox* may contain the following expressions:

Woman(Alice)	*"The individual named Alice is a Woman"*
Person(Bob)	*"The individual named Bob is a Person"*
hasChild(Alice,Bob)	*"Bob is Alice's child"*

A knowledge base (KB) consists of a pair $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a *TBox* and $\mathcal{A}$ is an *ABox*. Intuitively, we can see that, given a KB composed of our example *TBox* and *ABox*, an apt reasoner would be able to infer that:

Mother(Alice)	*"The individual named Alice is a Mother"*

For us humans, it is relatively easy to see how this axiom follows from the previously established ones. Computers, however, do not have any knowledge but that which is given to them and so require a formalization of the meaning of each symbol in a description logic's syntax. To encode that meaning, semantics relies on pre-established set-theory. Accordingly, semantics are given by interpreting concepts as sets of individuals and roles

as sets of ordered pairs of individuals. Logical constructs called interpretations are used to encode how ontology axioms can be interpreted as set-theoretical expressions. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of domain $\Delta^{\mathcal{I}}$, which is an arbitrary non-empty set of individuals, and a interpretation function $\cdot^{\mathcal{I}}$ that maps:

- every **concept** to a subset of $\Delta^{\mathcal{I}}$
- every **role** to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

Such that, for all concepts C, D, and all role names R:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
\neg C^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | \text{ There is some } y \in \Delta^{\mathcal{I}} \text{ with } (x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | \text{ For all } y \in \Delta^{\mathcal{I}}, \text{ if } (x,y) \in R^{\mathcal{I}} \text{ then } y \in C^{\mathcal{I}}\}
\end{aligned}
$$

To say that an interpretation is a model of (denoted with $\models$) a given axiom is to say that the interpretation reflects the correct meaning of the symbols used in that axiom. For that reason, establishing rules for whether an interpretation is a model of a given axiom is a formal way of defining the meaning of those very symbols. An interpretation $\mathcal{I}$ is said to be a model of CGIs and assertional axioms according to the following rules:

$$
\begin{aligned}
\mathcal{I} \models C \sqsubseteq D \quad &\text{if and only if} \quad C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \\
\mathcal{I} \models \text{C(a)} \quad &\text{if and only if} \quad a^{\mathcal{I}} \in C^{\mathcal{I}} \\
\mathcal{I} \models \text{R(a,b)} \quad &\text{if and only if} \quad (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}
\end{aligned}
$$

An interpretation $\mathcal{I}$ is said to be a model of a *TBox* if it is a model of every CGI in it and is said to be a model of a *ABox* if it is a model of every assertional axiom in it:

$$
\begin{aligned}
\mathcal{I} \models \mathcal{T} \quad &\text{if and only if} \quad \mathcal{I} \models \Phi \text{ for every } \Phi \in \mathcal{T} \\
\mathcal{I} \models \mathcal{A} \quad &\text{if and only if} \quad \mathcal{I} \models \Phi \text{ for every } \Phi \in \mathcal{A}
\end{aligned}
$$

Finally, an interpretation $\mathcal{I}$ is said to be a model of a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ if it is a model of both its *TBox* and *ABox*:

$$\mathcal{I} \models \mathcal{K} \quad \text{if and only if} \quad \mathcal{I} \models \mathcal{T} \text{ and } \mathcal{I} \models \mathcal{A}$$

One knows that one's perception of a given axiom or set of axioms is congruent with their actual meaning if the interpretation corresponding to that perception is a model of the axiom or set of axioms. Given this formalization of a description logic's semantics, reasoners use set-theory principles to infer an individual's membership to a concept (i.e., a set of individuals) as a consequence of its membership to other concepts.

Another critical notion to introduce here is entailment. Entailment means that an axiom Φ follows from a set of axioms KB. Entailment is also denoted with $\models$. Formally, we say that KB $\models \Phi$ if and only if Φ is true in all worlds where KB is true. In other words, it is impossible to instantiate a model that entails all axioms in KB but not Φ.

2.2.2 Current Description Logics and Reasoners

Web Ontology Language (OWL) is a family of knowledge representation languages built on the World Wide Web Consortium's (W3C) XML standard RDF [37]. The W3C currently endorses multiple profiles of OWL, which vary in their expressiveness. OWL 2 DL offers the highest possible expressiveness while remaining decidable. It features all OWL constructs but has constraints on how they can be used. OWL 2 Full is OWL 2 without any limitations; however, it is undecidable. Thus no reasoner exists for it that can always return a yes/no answer. OWL 2 EL, OWL 2 QL, and OWL 2 RL are all less expressive than OWL 2 DL and aimed at specific applications - large bio-health ontologies, integration with database management systems, and scalable reasoning, respectively [36].

Popular reasoners for OWL include Fact++ [16] by the University of Manchester, Hermit [24] by Oxford University Computing Laboratory, Pellet [39] by Clark & Parsia, LLC, and RacerPro [43] by Racer Systems. However, some of these reasoners have not, at the time of writing, fully implemented OWL 2 conformance.

A notable extension of OWL-DL is DISPONTE [50] semantics, which introduces the ability to represent uncertainty in an ontology's axioms by annotating each axiom with the probability that it is true. DISPONTE creators, Riguzzi et al., also presented the BUNDLE reasoner, which performs inference over probabilistic ontologies, and produces explanations with an attached probability that can be interpreted as a confidence value.

3

State of the art

In this chapter, we review the current literature on the areas relevant to this dissertation. We include a general overview of two fields - Explainable AI and Neural Architecture Search - and go into deeper detail on the technologies that will be directly involved in our work - Concept Mapping and DARTS, respectively.

3.1 Existing Explainable AI Solutions for black-box models

Multiple solutions to the Explainable AI problem have been proposed. One category of such solutions is proxy-based methods, in which, given a complex uninterpretable model, a simpler model is created and trained to behave similarly to the first. This simpler model can then presumably be interpreted [47]. However, these methods do not directly provide any explanation for the original model's behavior, only a means of finding one. Often, proxy-based methods produce either over-simplified versions of the original model or ones too complex to be easily interpretable. Additionally, since some of these methods only use the input-to-output relation of the original model to build the proxy one, it is possible to build unfaithful proxies.

Other Explainable AI solutions focus on outlining which parts of a given input were most relevant to a model's decision. This is done via the creation of Attribution maps, which assign a contribution value to each input, corresponding to how important that input was to the model's decision. Attribution maps can be generated by using the internal parameters of the model to compute relevancy metrics [4, 57], or by selectively occluding parts of the input and measuring how that affects the produced output [74] [40]. Figure 3.1 shows an overview of how one of these methods (RISE [40]) functions, and Figure 3.2 shows examples of the attribution maps obtained using that same method. Attribution maps, while providing information about what parts of the input were most relevant to a produced output, provide no information as to why that is the case [47]. The task of leveraging the knowledge into an understanding of the model's inner workings still falls on the user.

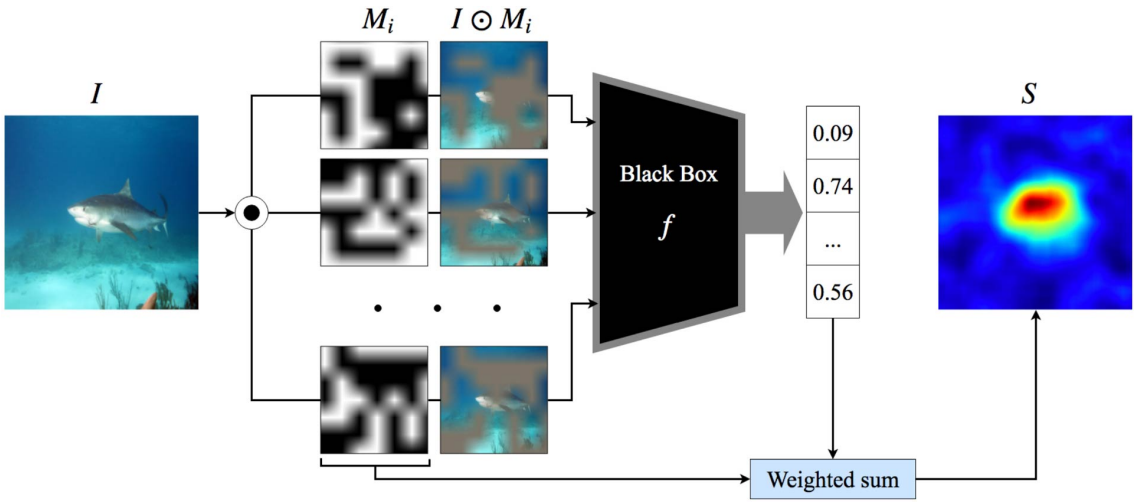
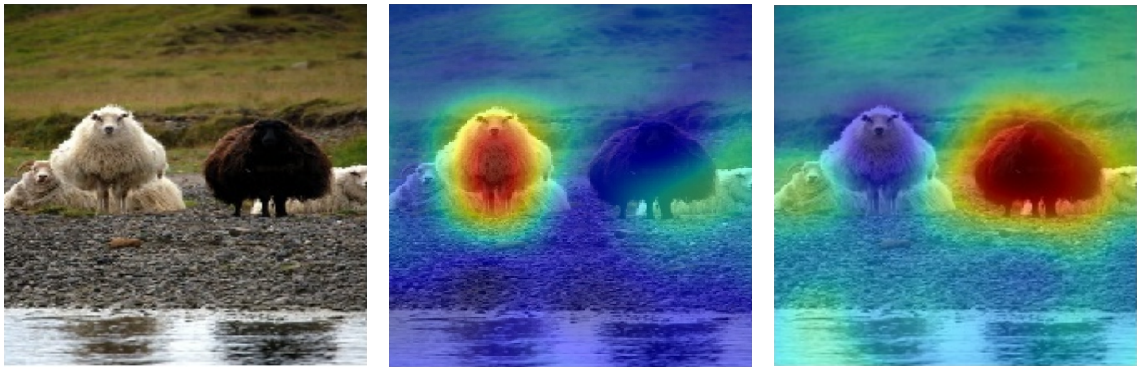


Figure 3.1: Overview of RISE [40].

Parts of an image are occluded using randomized masks. The outputs of a black-box model are recorded for each masked input. A final saliency map is generated using a weighted function on those outputs.



(a) Sheep - 26%, Cow - 17% (b) Importance map of 'sheep' (c) Importance map of 'cow'

(d) Bird - 100%, Person - 39% (e) Importance map of 'bird' (f) Importance map of 'person'

Figure 3.2: Pixel importance maps generated using RISE [40] for a ResNET50 (black-box) model.

(b) and (c) show that the model only recognized the white sheep while confusing the black one for a cow. (e) shows that the model mainly recognized the bird on the right-hand side, and (f) shows that the model misinterpreted parts of the birds as a person.

3.2 Concept Mapping

Sousa Ribeiro and Leite [60] developed a method to produce justifications for a neural network's output. This method, concept mapping, leverages an ontology that provides the necessary knowledge about the task's domain to produce justifications for a given output. This ontology includes concepts equivalent to the ones extracted by the main network (i.e., the classes in the classification task) and others relevant to the definition of the main concepts. In order to leverage that knowledge, a mapping between the network's internal representations of the input and each relevant concept is established. In other words, a mapping is established between each relevant concept and what internal behavior it triggers in the main network when it appears in the input.

3.2.1 The Mapping Networks

In order to establish this mapping, Sousa Ribeiro and Leite [60] proposed the use of classifiers, dubbed mapping networks, each trained to detect a single concept in the network's latent activations. These mapping networks are trained via a supervised learning approach, meaning labeled examples must be produced beforehand to train them. To do this, examples from the dataset of the main network are additionally labeled with the selected ontology concepts present in them. For example, an image of a turtle may contain the concepts describing "individuals having four legs" and "individuals having a shell". Since this is a manual and expensive process, it is only done for a subset of examples of the size needed to train the mapping networks. After acquiring the labels, the mapping networks are fed the activations of the hidden layers in the main network, and given whether their respective ontology concept was present in the input that produced those activations as targets, .

The reader might wonder whether all of the main network's activations need be fed to the mapping network. After all, conventionally sized CNNs can have tens of millions of trainable parameters, and given that all the data is fed forward through the network, we should expect the same information to be found at multiple levels, albeit not always in the same format. We previously saw that Convolutional Neural Networks are apt at extracting complex features from unstructured image data. This is due to how the stacking of multiple layers allows for simple patterns (e.g., lines and basic shapes) to be detected in the first layers and more complex ones (e.g., ears, tail, wheels) to be extracted in the latter ones. So, if a concept in the ontology describes "All animals with a tail" and that concept is relevant to classification task, it should be easily extracted from the activations of one of the latter layers. As was detailed in Section 2.1.1.5, at the end of the convolutional part of a CNN, the features extracted from the image have been identified and largely abstracted from their position in the input. Therefore, the representations of the dense part of the network are viable candidates for input for the mapping task.

3.2.2 Producing justifications

A justification for a given fact Φ, given a set of axioms KB is the minimal subset of KB that entails Φ. It is a minimal subset in the sense that removing any one of the facts will result in the entailment no longer holding. So a justification for Φ, given a KB is $\mathcal{J} \subseteq \text{KB}$, such that $\mathcal{J} \models \Phi$ and for all $\mathcal{J}' \subset \mathcal{J}$, $\mathcal{J} \not\models \Phi$. This ensures that no unnecessary axioms are presented in the justification. Having trained the mapping networks, we can assess what concepts the main network found in a given input and subsequently produce a justification for its output by leveraging those concepts along with the ontology. In essence, we form a knowledge base in which the *TBox* is the knowledge contained in the ontology, and the *ABox* contains the observations of the mapping networks. Then, we produce a justification for an axiom that represents the output of the main model, given that KB. For example, consider a KB = $(\mathcal{T}, \mathcal{A})$ where:

$$\mathcal{T} = \{Woman \equiv Human \sqcap Female, Mother \equiv Woman \sqcap \exists hasChild.Person, (...)\},$$
$$\mathcal{A} = \{\}$$

Now, let us imagine that the main network classified a given input as Woman and that the concept mappers found the concepts Human and Female in its internal representations. First, we add the concept mappers' findings to the *ABox* in the KB:

$$\mathcal{T} = \{Woman \equiv Human \sqcap Female, Mother \equiv Woman \sqcap \exists hasChild.Person, (...)\},$$
$$\mathcal{A} = \{Human(input), Female(input)\}$$

Then, producing a justification for the axiom $Woman(input)$, which represents the output of the main network, we would get the axioms:

$$Human(input)$$
$$Female(input)$$
$$Woman \equiv Human \sqcap Female$$

From which by necessity follows that $Woman(input)$, giving us the likely reasoning the main network took.

A further problem arises when producing these justifications in a real - more complex - scenario: often, large numbers of plausible justifications are generated for a single output. This is not helpful to the human user, who likely wants to be given only a few, if not a single explanation. For this reason, Sousa Ribeiro and Leite[60] interpreted the values output by the mapping networks as a degree of belief in their prediction. Then, they used DISPONTE [50] - a semantics for representing probabilistic knowledge bases - to represent that degree of belief and the BUNDLE reasoner [49] - a Pellet-based [59] reasoner that supports DISPONTE - to calculate the degree of belief for each justification. Then, the justifications with the highest degree of belief are selected to be shown to the user.

When interpreting the output of the mapping networks as their degree of confidence, calibration becomes a consideration. Calibration is a notion that describes how close the probability given by the main network for the membership of an input to a class is to the actual probability of that input belonging to that class. In other words, for a perfectly calibrated model, predictions with probability $p \in [0,1]$ are correct $100p$ percent of the time. Plotting the model's degree of confidence against its accuracy, then, should yield the identity function.

Calibration can be visually evaluated with reliability diagrams. Because there are finite samples, predictions are partitioned into equally-shaped bins, and accuracy is computed jointly for each bin (Figure 3.3). Deviation from the identity represents miscalibration.

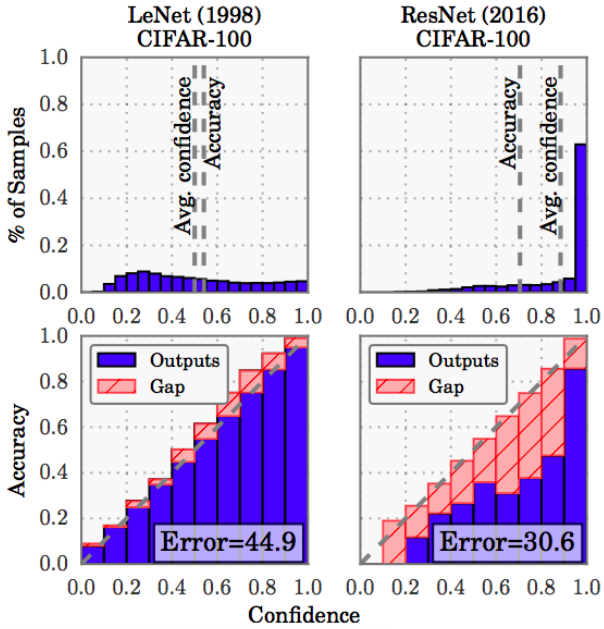


Figure 3.3: Confidence histograms (top) and reliability diagrams (bottom) for a 5-layer LeNet (left) and a 110-layer ResNet (right) on CIFAR-100. Taken from [22].

Further, calibration can also be summarized in a single scalar value. Expected Calibration Error (ECE) is one of several ways of doing so. ECE approximates calibration error by taking the weighted average of the accuracy/confidence difference of M bins, containing a total of n samples, like this:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)| \tag{3.1}$$

where B_m is the mth bin. An ECE value of zero means perfect calibration (accuracy equals confidence).

3.2.3 The protocol

Let us summarize the protocol for employing the Concept Mapping explanatory method. Firstly, meet the following pre-requisites:

- Have a dataset and a classification task for that dataset.
- Have a deep neural model (the main network) that was trained for that task and whose behavior you want to justify.

Then, acquire the tools required for the method:

- Write an ontology - in a decidable description logic - that describes concepts analogous to the classes in the classification task and their relations to other, more specific concepts that you deem relevant.
- Label a subset of examples in the dataset with respect to the relevant concepts.
- Build and train classifiers (the mapping networks) to map the network's internal representations to each relevant concept.
- Acquire a reasoner that is compatible with your chosen description logic.

Finally, it is possible to produce justifications for the main network's output for a given example by:

- Providing the main network with the example and having it produce a classification.
- Using the mapping networks to identify the concepts that the main network found in that example.
- Using the reasoner to produce a justification for the network's output.

Optionally, use a probabilistic reasoner to rank the produced justifications. Fig. 3.4 shows an overview of the Concept Mapping method.

3.2.4 Discussion

This method relies on the hypothesis that if the selected concepts are indeed relevant to the task, they will be extractable from the trained network's internal representations of the input at a high level of accuracy. This hypothesis was tested and confirmed. Naturally, a situation can happen when the human-defined and thought-to-be-relevant concepts can not be effectively extracted from the main network's internal representations. This would be an early sign that the network is extracting the wrong features. This could happen if the dataset contained irrelevant features that were strongly statistically correlated to the relevant ones (e.g., all the pictures of dogs also feature green grass). If the irrelevant features are easier to learn than the relevant ones, the network will likely learn incorrectly and overfit. Similarly, the selected ontology concepts should also be as statistically

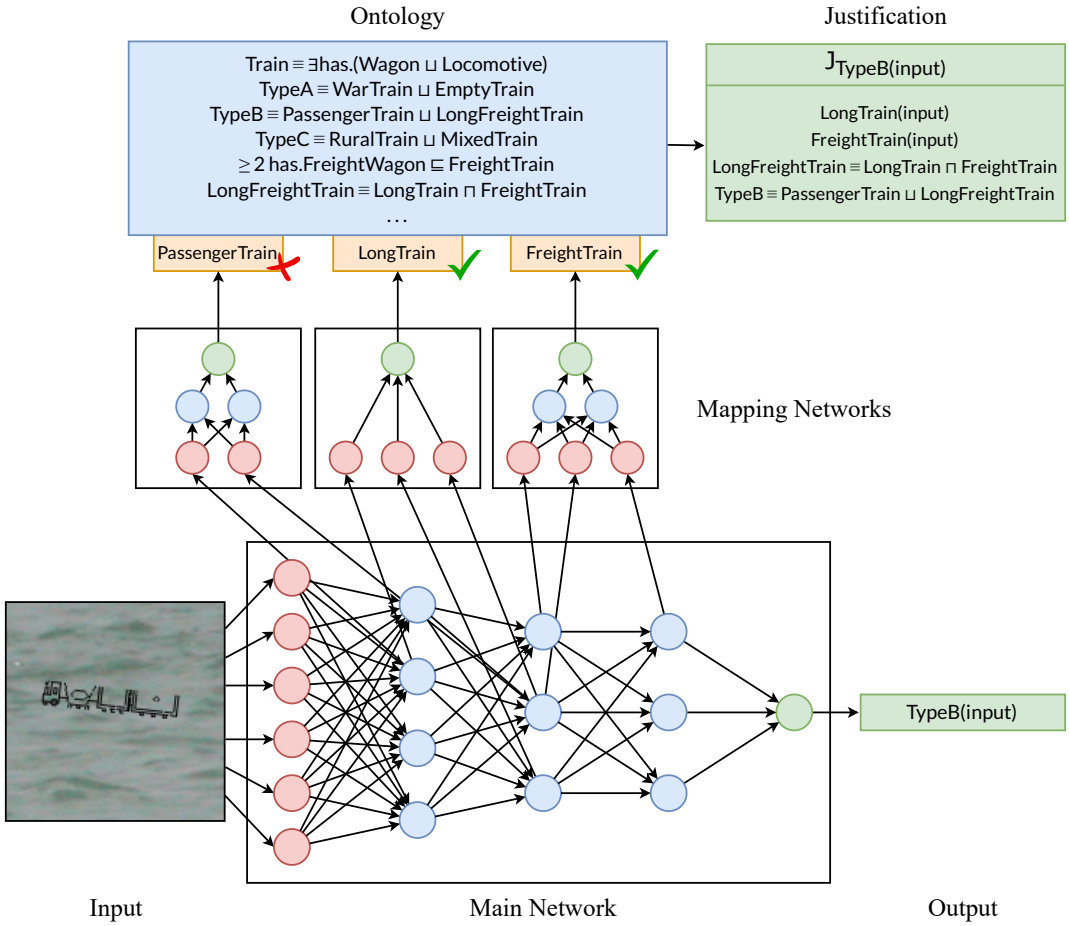


Figure 3.4: Overview of the Concept Mapping Method, sourced from [60]

independent of each other as possible. Otherwise, a mapping network could erroneously learn to detect a concept by "looking for" representations produced by a different one. If a strong correlation between two concepts is present in the actual domain and not just a consequence of biased sampling, then it should be considered that only one need be extracted.

Another consideration for this method is the input received by the mapping networks. Providing them with the whole set of activations can be considered inefficient, and it may be thought that filtering the input may result in performance gains. Sousa Ribeiro and Leite [60] experiment with using a procedure called input Reduce, which by selectively hiding some of the features from the mapping network, training it with those features, and comparing results with different configurations, manages to reduce the input size of the mapping networks while maintaining similar accuracy. While InputReduce did manage to provide similar to slightly better performance with fewer parameters, the computational cost of running the procedure is very high. This, coupled with the fact that ANNs are already quite apt at extracting the relevant sections from a larger input makes using InputReduce unnecessarily expensive.

3.3 Neural Architecture Search

Due to its ability to perform feature extraction, deep learning has led to large strides in many fields of research. However, creating a Neural Network for a particular problem is far from straightforward, as it heavily relies on the researcher's prior experience, making it hard for beginners to make valuable contributions. In fact, discovering state-of-the-art neural network architectures requires substantial effort from even human experts.

The field of Neural Architecture Search (NAS) stems from the interest in streamlining and automating the process of neural architecture design. NAS-RL [76] and META-QNN [5], two early NAS systems that were based on Reinforcement Learning (Section 2.1.2), can create architectures capable of state-of-the-art performance in image classification tasks [45]. While these systems were important proofs-of-concept for NAS, they require huge amounts of computational power by the standards of most ML researchers, effectively locking them from using NAS in a meaningful way. Since we are optimizing the architecture of relatively simple models - the mapping networks - this does not outright disqualify it as a viable tool. Nevertheless, in recent years, the NAS research community has placed significant emphasis on finding approaches for speeding up the process.

As a relatively young field, NAS research is rather exploratory in nature, meaning that different NAS systems diverge significantly in their methodology. So, to better understand and compare the different alternatives, it is important to know the main defining attributes of a NAS system:

- **Search Space:** The domain of architectures that are considered as candidates. This attribute poses a trade-off between convergence time and the quality of the final model. Having a more extensive search space may lead to finding higher-quality architectures, but it will also increase the time it takes to find them.
- **Search Strategy:** The method by which the Search Space is traversed. Common search strategies employed by early NAS systems are random search (RS) [55], reinforcement learning (RL) [5, 76], evolutionary algorithms (EA) [44], bayesian optimization (BO) [27], monte carlo tree search (MCTS) [35], and sequential model-based optimization (SMBO) [25].
- **Evaluation Strategy:** The method used to benchmark the candidate architectures. In order to find an optimal architecture, we need a protocol to obtain at least a comparative ranking between different candidates.

In sum, a NAS system traverses its search space using its search strategy while using its evaluation strategy to decide the direction in which to head. Fig. 3.5 shows a diagram of this general framework.

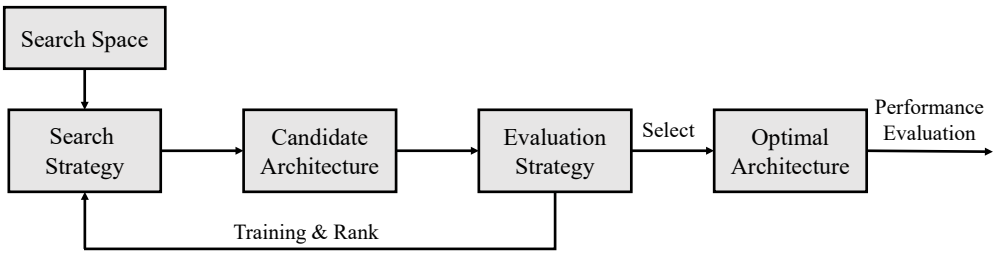


Figure 3.5: The general framework of NAS [45]

Here we summarize the main ways in which these core aspects have changed from the very first implementations of NAS to modern and much more efficient NAS solutions:

- **Modular Search Space:** Early NAS work utilized a global search space, which means the system had to optimize the entire architecture at once. Recently, a popular approach has been to adopt a Modular search space. This commonly entails transforming the search space into a few small computational structures, often called cells or blocks. A final architecture is built by repeatedly stacking these cells. This method effectively reduces the complexity of the NAS search because optimizing a single cell is a much simpler problem than optimizing a whole architecture. Further, this approach offers versatility since the same learned cells can be arranged in different ways to allow for good performance in different dataset tasks, which is much less feasible with a global search space. This concept is further supported by the fact that state-of-the-art human-made architectures often consist of repeating architectural patterns, such as the VGG [58], and ResNet [23] architectures.

- **Continuous Search Strategy:** The initial approaches to NAS, based on Reinforcement Learning, Evolutionary algorithms, and such, treat NAS as a discrete optimization problem. This is the intuitive approach, given the modular nature of deep neural networks. However, adopting a continuous search strategy, and thus allowing for the use of gradient optimization techniques, can provide significant efficiency benefits. Earlier works adopting a continuous search strategy focused on fine-tuning existing architectures (e.g., optimizing the hyperparameters of convolutional layers [48]). DARTS [33], a later work, formulates the complete NAS problem in a differentiable manner, allowing for efficient search of the architecture using gradient descent.

- **Neural Architecture Recycling** Searching a Neural Architecture from scratch has its advantages. The higher degree of freedom makes it more likely for an "out-of-the-box" network structure to be found. However, this comes with a significant efficiency cost. To make NAS algorithms more time-efficient, we can leverage prior knowledge and high-performance networks as a starting point for further optimization. To this effect, multiple NAS works have focused on making incremental transformations to an existing man-made network [6, 8].

- **Incomplete Training** At the core of the NAS problem is the benchmarking and comparison of multiple candidate networks. The value being optimized is the loss of the fully trained architecture on a validation set. So, the intuitive way to compare the networks is to simply train each one and measure their performance. Although this is the most accurate way to compare candidate networks, it is also the slowest, once again introducing a trade-off between performance and efficiency. Furthermore, the learned parameters of each candidate network are simply discarded after training, which is an under-utilization of that learning. To target this issue, ENAS [41] pioneered the concept of parameter sharing by regarding each candidate network as a computing subgraph of a supercomputing graph - the search space. With this paradigm, multiple candidate networks containing the same section of the supercomputing graph can share the optimized weights for that section (Fig. 3.6). This removes the need to train each candidate network from scratch, significantly reducing time complexity. Another popular method to reduce time spent evaluating candidate networks is early stopping. This simply entails training each candidate network for only a few epochs and extrapolating from there their relative performance. In fact, Zheng et al. [75] showed that the relative performance rankings of candidate networks largely remain the same between early and late training. Hutter, Hoos, and Leyton-Brown [25] propose a new ranking framework: Sequential Model-Based Optimization (SMBO), which tackles the problem differently by continuously training a performance predictor network (called the surrogate model). Only the candidate cells that are most highly rated by the surrogate model are actually trained, after which the surrogate model's weights are optimized to reflect the observed performance. As the search progresses, the surrogate gets progressively better at selecting the most promising architectures.

The main factors in making a choice of NAS framework for use in concept mapping are efficiency and performance. These two attributes are often inversely proportional, thus offering a trade-off. However, due to the extensive work done in optimizing NAS methods, it is today possible to obtain state-of-the-art results without investing thousands of GPU days. In the extensive performance comparison presented in [45] of current NAS methods on the CIFAR-10 dataset, it was found that NAS-RL, requiring 22400 GPU days to converge, found a network with a 3.65 error rate, while DARTS, requiring 4 GPU days to converge, found one with 2.76 error rate. This is partly due to how DARTS uses a modular search space, continuous search strategy, and incomplete training, while NAS-RL features none of the above. Given its state-of-the-art results and diminished computational requirements, DARTS was selected as the basis for a NAS algorithm to use for this dissertations' experiments and concept mapping as a whole. Thus, we will discuss it in more detail here.

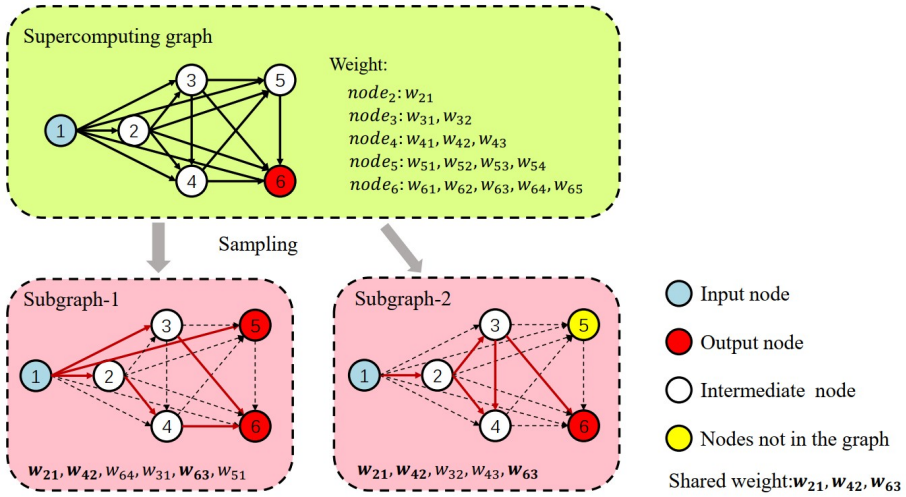


Figure 3.6: Diagram of the parameter sharing mechanism in ENAS [41]

The nodes in the graphs represent data transformations and the edges represent data flow. Different candidate networks sample different subgraphs of the supercomputing graph, but the weights are shared for intersection regions. Sourced from [45].

3.3.1 DARTS

3.3.1.1 Search Space

DARTS uses a modular search space, which means that, instead of designing a whole architecture, it designs a cell that can be duplicated and stacked to build a final architecture. In the original DARTS work, copies of the learned cell can be arranged to form convolutional or recursive neural networks, depending on the configuration. A cell is regarded as a Directed Acyclic Graph (DAG), in which each node $x^{(i)}$ is an internal representation of the input and each edge (i, j) is an operation $o^{(i,j)}$ that further transforms $x^{(i)}$. The content of each node $x^{(i)}$ is computed based on all of its incoming edges:

$$x^{(i)} = \sum_{i<j} o^{(i,j)}(x^{(i)}) \tag{3.2}$$

Each cell is set to have two inputs and one output. The inputs represent the outputs from its two closest preceding cells. In the case of the first cell, its two inputs are the same and equal to the input of the network. The second cell's inputs are the input of the network and the output of the first one, and so forth. The output of a cell is given by a concatenation of the output of each of its nodes.

Learning the optimal cell consists of learning the optimal operation to place on each of its edges. In addition to other candidate operations, a *Zero* operation is used to indicate a non-existing connection between two nodes.

During training, to continuously relax the search space, each edge of the cell, rather than a particular operation, represents a Softmax over all candidate operations:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_o^{(i,j)})} o(x) \tag{3.3}$$

where $\mathcal{O}$ is the set of all candidate operations and α a set of vectors $\alpha^{(i,j)}$ containing mixing weights associated to each operation of each edge between two nodes. Essentially, rather than an edge representing a single transformation to the data, it represents, during the architecture search, a weighted sum of all candidate transformations, weighted by the likelihood that each is the optimal one. This likelihood is given by α, the trainable parameters via which DARTS learns an architecture. Going further, we will refer to the set of these mixing weights α as the arch weights - since they encode the architecture - and to the trainable parameters of the candidate operations as the model weights.

Once the optimal cell has been learned, the discrete architecture can be obtained by replacing the mixed operations with the most likely operation:

$$o^{(i,j)} = argmax_{o \in \mathcal{O}} \quad \alpha_o^{(i,j)} \tag{3.4}$$

The validation-set loss value depends on both the arch weights and model weights, so, in order to minimize it, both are jointly optimized. This implies a bilevel optimization problem, a kind of hierarchical optimization where an optimization task contains another optimization task as a constraint. We refer to the former as the upper-level task and the latter as the lower-level task. Informally, the upper level's task outcome is a function of its own decision (the upper-level variable) and the lower-level task's decision (the lower-level variable). For any given decision made by the upper-level task, there is a lower-level optimization problem that provides the optimal "response" to that decision. In our case, we want to optimize the arch weights α, such that when the weights w of the resulting architecture are optimized to minimize the training-set loss (denoted with $w^*(\alpha)$), the validation-set loss is minimized. The resulting bilevel optimization problem is one with α as the upper-level variable and w as the lower-level variable:

$$min_\alpha \quad \mathcal{L}_{val}(w^*(\alpha), \alpha)) \tag{3.5}$$

$$s.t.^{[1]} \quad w^*(\alpha) = \text{argmin}_w \quad \mathcal{L}_{train}(w, \alpha) \tag{3.6}$$

where $\mathcal{L}_{val}$ is the validation loss and $\mathcal{L}_{train}$ is the training loss. To solve this problem, the arch weights α are first optimized by descending the validation loss. Then the model weights are optimized by descending the training loss of the architecture found in the upper-level optimization. This process is repeated until convergence, at which point the final architecture is derived. Figure 3.7 contains a summary of the DARTS process for Neural Architecture Search. It is important to note that this diagram is slightly simplified.

[1] a short-hand for "subject to"

In practice, there are two additional incoming edges to each of the nodes, one for each of the cell's inputs (recall that a cell has two inputs from its two closest preceding cells). These additional edges act just like the others: they represent a set of candidate data transformations that are optimized during training.

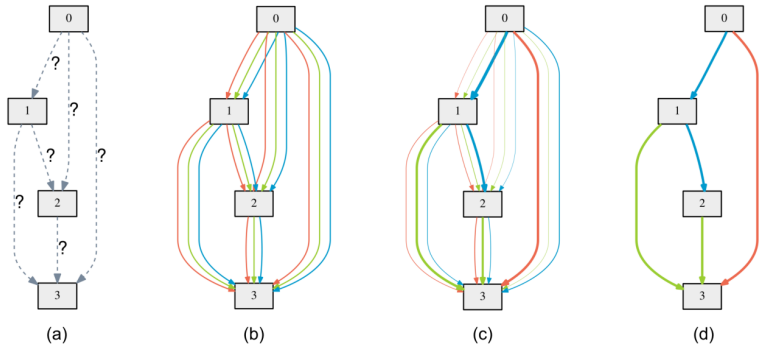


Figure 3.7: An overview of DARTS [33].

(a) The problem, find the optimal cell. (b) Continuous relaxation of the problem by replacing each edge with a set of mixed operations. (c) Joint optimization of architecture and network weights. (d) Derivation of final architecture by replacing mixed operations with the learned most likely operation

3.3.2 Approximating the Architecture Gradient [2]

Optimizing the upper-level task (equation 3.5) with gradient descent entails computing the gradient of the validation loss for a model with the current arch weights and fully trained model weights ($\mathcal{L}_{val}(w^*(\alpha), \alpha)$). However, that would involve training the model to convergence at each step of the architecture search, and that would be prohibitively expensive. This computation is at the core of DARTS, and indeed was implemented in the course of this dissertation, so we will discuss it in detail here. DARTS proposes the following approximation:

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \tag{3.7}$$

$$\approx \nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha) \tag{3.8}$$

where $w^*(\alpha)$ denotes the fully optimized model weights for the current arch weights, w denotes the current model weights, and ξ is the learning rate for the model weights optimization. This avoids having to compute $w^*(\alpha)$ (i.e., optimize the model weights until convergence) by approximating it using a single training step on w. Now, we must simply compute the approximate gradient in expression 3.8.

[2] Note that the background on the notation surrounding gradients and partials derivatives from Section 2.1.1.1 is relevant to this section.

To begin solving the expression let us use w' as a short-hand for $w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha)$. Using this abbreviation, we get the following expression:

$$\nabla_\alpha \mathcal{L}_{val}(w', \alpha) \tag{3.9}$$

Here, essentially, we have to compute the gradient of a multivariate function $\mathcal{L}_{val}$ whose variables are both functions of α. The first term w' is a function $x(\alpha) = w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha)$ and the second term α can be viewed as the identity function $y(\alpha) = \alpha$. To solve such a gradient, we use the multivariate chain rule, which states that given two functions x(t) and y(t), both differentiable at t, then a function z = f(x(t), y(t)) is differentiable at t and:

$$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt} \tag{3.10}$$

In our case:

- t is α
- x(t) is $x(\alpha) = w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha) = w'$
- y(t) is $y(\alpha) = \alpha$
- z = $\mathcal{L}_{val}(w', \alpha)$

Thus, following equation 3.10 and starting with the first term, we need to compute:

$$\frac{\partial z}{\partial x}\frac{dy}{dt} \widehat{=}^{3} \frac{\partial \mathcal{L}_{val}}{\partial w'}\frac{dw'}{d\alpha} \tag{3.11}$$

The left-hand side of the product is the partial derivative of $\mathcal{L}_{val}$ with respect to w' (i.e. $\nabla_{w'} \mathcal{L}_{val}(w', \alpha)$) and the right-hand side is the derivative of w' (i.e. $\nabla_\alpha(w')$). So, we can now write the first term of equation 3.10 for our case:

$$\frac{\partial \mathcal{L}_{val}}{\partial w'}\frac{dw'}{d\alpha} = \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \nabla_\alpha(w') \tag{3.12}$$

Given that w' is a known function, we can further solve $\nabla_\alpha(w')$. Since $w' = w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha)$ and $\nabla_w \mathcal{L}_{train}(w, \alpha)$ is a function of α, this is a derivative of a function composition, so we can use the chain rule, which states that, given a function g(x) differentiable at x, then a function f(g(x)) is differentiable at x and:

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx} \tag{3.13}$$

In our case:

- f is w'
- g is $g(\alpha) = \nabla_w \mathcal{L}_{train}(w, \alpha)$

[3] The $\widehat{=}$ symbol simply means "corresponds to"

- x is α

And thus, we need to compute:

$$\frac{df}{dg}\frac{dg}{dx} \triangleq \frac{dw'}{d\nabla_w \mathcal{L}_{train}}\frac{d\nabla_w \mathcal{L}_{train}}{d\alpha} \tag{3.14}$$

The left-hand side of the product is the derivative of w' with respect to the function $\nabla_w \mathcal{L}_{train}$, so, we treat $\nabla_w \mathcal{L}_{train}$ as a variable and solve the derivative, which yields $-\xi$. The right-hand side is the derivative of $\nabla_w \mathcal{L}_{train}$ with respect to α, i.e $\nabla_\alpha(\nabla_w \mathcal{L}_{train}(w,\alpha))$, which can be written as $\nabla^2_{\alpha,w}(\mathcal{L}_{train}(w,\alpha))$. So we solve $\nabla_\alpha(w')$ as such:

$$\frac{dw'}{d\nabla_w \mathcal{L}_{train}}\frac{d\nabla_w \mathcal{L}_{train}}{d\alpha} = -\xi\nabla^2_{\alpha,w}(\mathcal{L}_{train}(w,\alpha)) \tag{3.15}$$

Replacing $\nabla_\alpha(w')$ in expression 3.12, we get:

$$-\xi\nabla^2_{\alpha,w}(\mathcal{L}_{train}(w,\alpha))\nabla_{w'}\mathcal{L}_{val}(w',\alpha) \tag{3.16}$$

And so we arrive at the first term in equation 3.10. Turning our attention to the second term, we need to compute:

$$\frac{\partial z}{\partial y}\frac{dy}{dt} \triangleq \frac{\partial \mathcal{L}_{val}}{\partial y}\frac{dy}{d\alpha} \tag{3.17}$$

Since y is the identity function of α, the right-hand side of the product equals 1, and we get the following:

$$\frac{\partial \mathcal{L}_{val}}{\partial y}\frac{dy}{d\alpha} = \frac{\partial \mathcal{L}_{val}}{\partial \alpha} = \nabla_\alpha \mathcal{L}_{val}(w',\alpha) \tag{3.18}$$

Finally, plugging the solved terms into expression 3.10, we get the following approximate gradient for the arch weights:

$$\nabla_\alpha \mathcal{L}_{val}(w',\alpha) - \xi\nabla^2_{\alpha,w}(\mathcal{L}_{train}(w,\alpha))\nabla_{w'}\mathcal{L}_{val}(w',\alpha) \tag{3.19}$$

This expression contains an expensive matrix-vector product in the second term, so DARTS reduces the complexity significantly by using the finite difference approximation. To understand this principle, let us write the formal definition of a partial derivative:

$$\frac{\partial f(x,z)}{\partial x} = \lim_{e\to 0}\frac{f(x+\epsilon,z) - f(x-\epsilon,z)}{2\epsilon} \tag{3.20}$$

The finite difference method approximates this limit by choosing a fixed and very small value for ϵ:

$$\frac{\partial f(x,z)}{\partial x} \approx \frac{f(x+\epsilon,z) - f(x-\epsilon,z)}{2\epsilon} \tag{3.21}$$

Since ϵ is an arbitrarily small number, we can multiply it by a constant k, which yields:

$$\frac{\partial f(x,z)}{\partial x} \approx \frac{f(x+k\epsilon,z)-f(x-k\epsilon,z)}{2k\epsilon} \tag{3.22}$$

Then, multiplying both sides of the equation by k we get:

$$\frac{\partial f(x,z)}{\partial x}k \approx \frac{f(x+k\epsilon,z)-f(x+k\epsilon,z)}{2\epsilon} \tag{3.23}$$

We can use this equation to approximate the product of a partial derivative by a constant k. Notably, that is what we have in equation's 3.19 second term:

$$\nabla^2_{\alpha,w}(\mathcal{L}_{train}(w,\alpha))\nabla_{w'}\mathcal{L}_{val}(w',\alpha) \tag{3.24}$$

$$= \nabla_\alpha(\nabla_w(\mathcal{L}_{train}(w,\alpha)))\nabla_{w'}\mathcal{L}_{val}(w',\alpha) \tag{3.25}$$

Given the principle of the equality of mixed partials (Schwarz's theorem), which states that it is possible to interchange the order of taking partial derivatives of a function without changing the result, we get:

$$\nabla_w(\nabla_\alpha(\mathcal{L}_{train}(w,\alpha)))\nabla_{w'}\mathcal{L}_{val}(w',\alpha) \tag{3.26}$$

Writing this is in the notation of equation 3.23, we get:

$$\frac{\partial \nabla_\alpha \mathcal{L}_{train}(w,\alpha)}{\partial w}\nabla_{w'}\mathcal{L}_{val}(w',\alpha) \tag{3.27}$$

And so we can approximate this expression. Let ϵ be a small scalar and $w^\pm = w \pm \epsilon\nabla_{w'}\mathcal{L}_{val}(w',\alpha)$, we consider $\nabla_{w'}\mathcal{L}_{val}(w',\alpha)$ to be the constant k and get:

$$\nabla^2_{\alpha,w}(\mathcal{L}_{train}(w,\alpha))\nabla_{w'}\mathcal{L}_{val}(w',\alpha) \approx \frac{\nabla_\alpha \mathcal{L}_{train}(w^+,\alpha)-\nabla_\alpha \mathcal{L}_{train}(w^-,\alpha)}{2\epsilon} \tag{3.28}$$

With this approximation, the expression no longer contains a matrix-vector product, requiring only two forward passes for the weights and two backward passes for α, which reduces the complexity from $O(|\alpha||w|)$ to $O(|\alpha|+|w|)$ [33]. DARTS uses $\epsilon = 0.01/\|\nabla_{w'}\mathcal{L}_{val}(w',\alpha)\|$. Using $\xi = 0$, the approximate gradient (equation 3.8) reduces to:

$$\nabla_\alpha \mathcal{L}_{val}(w) \tag{3.29}$$

Which simply entails computing the gradient of the validation loss with the current model weights. This leads to a speedup, but worse performance [33]. Liu, Simonyan, and Yang call this simplification 'First-Order approximation' and 'Second-order approximation' to the original version (with $\xi > 0$).

3.3.2.1 Discrete Architecture Derivation

DARTS derives a node in the discreet architecture by retaining the top-k strongest operations (from distinct nodes), excluding zero operations. If a *Zero* operation is given the strongest in a given edge, DARTS considers the second-strongest one instead. The strength of an operation is given by a softmax along all candidate operations:

$$\frac{exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} exp(\alpha_{o'}^{(i,j)})} \tag{3.30}$$

DARTS uses k=2 for convolutional cells and k=1 for recurrent cells. As previously mentioned, an arbitrary number of cells can be stacked to form the final architecture. In practice, the final architecture further performs data transformations before feeding its input to the first cell and after receiving the output from the last. In the case of the algorithm for finding convolutional architectures, the input is passed through two separate Convolutional - ReLu - Batch Normalization stacks (one for each input of the first cell), and the output of the last cell is passed through a final Convolutional layer before exiting the network (Fig. 3.8).

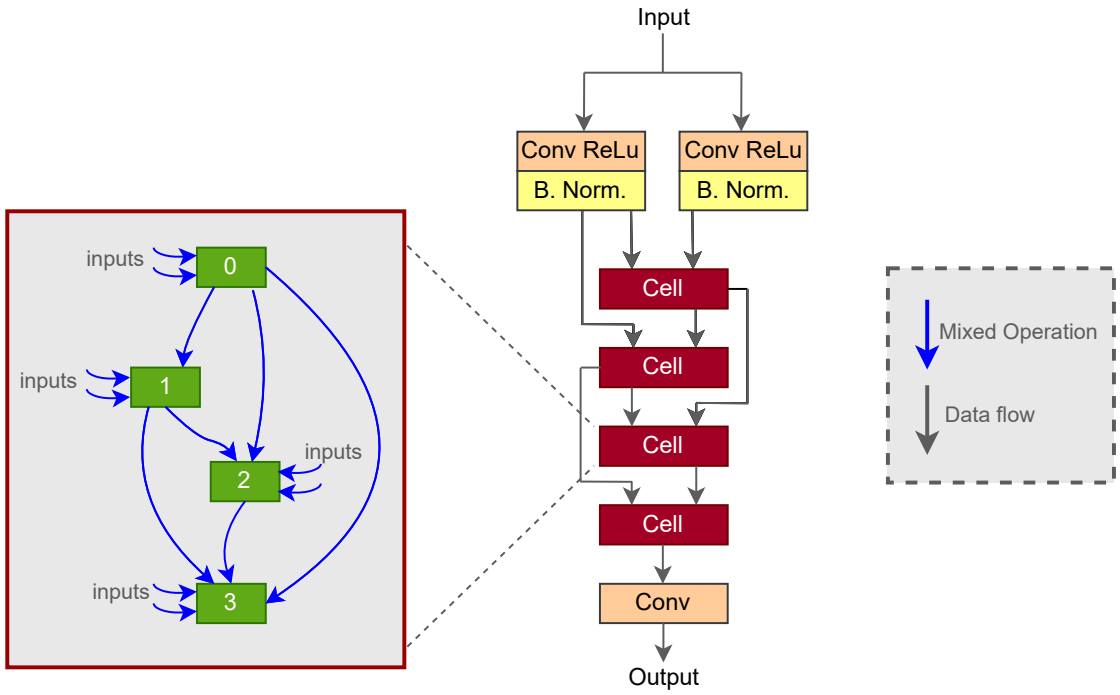


Figure 3.8: A diagram of a DARTS network composed of four cells.

The diagram shows the transformations applied to the input before feeding it to the first cell (top) and the final transformation applied to the output of the last cell (bottom). Each cell receives the output of the two preceding ones. Each node in the cell receives the two cell inputs and the output of all preceding nodes. Blue edges denote a mix of candidate operations (or, in the case of the discrete network, the learned operation), and gray edges simply denote data flow.

4

The VCB dataset

The variable-complexity buildings (VCB) dataset is a set of image classification datasets with an identical underlying ontology and labels that differ only in how difficult each one is to classify. This configuration is ideal for experiments that aim to understand how dataset complexity affects image classification, since that is the only changing variable. Additionally, each image of every dataset is labeled with 77 labels containing detailed information about its low-level features, the position of the camera, and present ontology concepts in addition to its respective class.

VCB is made up of synthetic data. Apart from the significant resources required for compiling and labeling real image data into an image classification dataset, that approach would not give us the level of control we needed for our purposes. Locking all variables other than complexity with real image data would be impossible.

The images contain depictions of buildings that vary in their surroundings and visual attributes. On the building itself, there may be a door, windows, awnings, billboards, porches, wall signs, roof statues, tiled roofs, chimneys, and pipes. Surrounding the building, there may be eating areas, pole signs, vending machines, construction machinery, trucks, and cars. Excluding the tiled roof and roof statues, the features can appear in one of multiple locations, and windows and awnings can also appear in varying quantities. Different combinations of these features are mapped to one of three main classes - Residential, Commercial, and Industrial - by an accompanying ontology.

4.1 Varying complexity

The term "Complexity" denotes how "hard to learn" a given dataset is. All other factors being the same, a classifier should converge faster and perform better on lower-complexity datasets (excluding overfitting).

Multiple methods are used to increase the complexity of the images: Alternating the appearance of each feature by using a model from a set of alternatives, varying the textures of surfaces such as the ground and walls, varying the camera's position, adding varying background elements and randomizing the contrast (the difference in brightness

between dark and light parts of the image) and overall brightness of each image. Here we create six levels of complexity by progressively introducing more of these complexity-adding methods:

	1	2	3	4	5	6
Models	✓	✓	✓	✓	✓	✓
Textures		✓	✓	✓	✓	✓
Camera - Small variance			✓	✓	✓	✓
Background elements				✓	✓	✓
Camera - Large variance					✓	✓
Contrast						✓
Brightness						✓

4.1.1 Models

There are five alternative models of each feature (e.g., Window, Door). While all variant models contain the representative traits of their respective types, they vary in shape, size, and color. This forces the classifier to learn the more nuanced defining attributes of the features rather than simply "memorizing" the appearance of an arbitrary instantiation of that feature.

4.1.2 Textures

Similarly to the models, textures are varied by sampling from a set of textures with contrasting shapes and colors. There are three sets of textures, each used for one of three - floor, overhang, and wall - sections and each containing five, three, and ten different textures, respectively. Fig 4.1 shows each object color-coded according to their section.

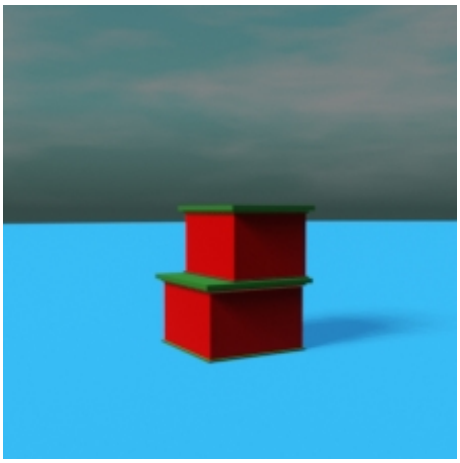


Figure 4.1: Texture sections in the VCB 3D environment.

The floor section is shown in blue, the overhang section is shown in green, and the wall section is shown in red.

4.1.3 Camera

Note that we further divide the camera category into two sub-categories: small and large variance, as this allows for a smoother increase in complexity through the levels. The camera variance entails varying the distance at which the camera is placed from the subject, the coordinates of the camera on the sphere with a radius equal to that distance, and the point to which the camera is pointed. By default, the camera points to (x,y,z) = (0,0,0) and is placed at the Cartesian coordinates (x,y,z) = (40.6, 71.2, 17.7), deemed to be fitting for a centered view of all features. The parameters that vary between the small and large variance categories are the range of allowed distances, the section of the sphere that the camera is allowed to be placed in, and the maximum deviation from (x,y,z) = (0,0,0) for the coordinates that the camera points to.

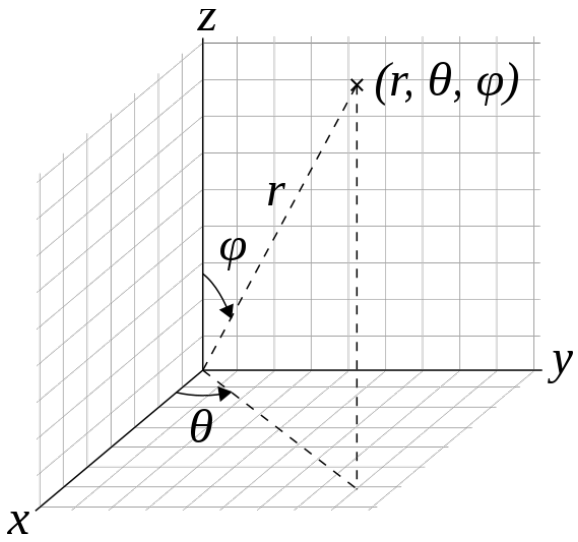


Figure 4.2: Spherical coordinates

With the axes shown in the image, the point with coordinates (0,0,0) is where the camera is pointed. R is the length of the vector that originates at that point and ends at the camera's location (radius vector). ϕ is the angle between the radius vector and the positive z-axis. θ is the angle between the x-axis and the vertical plane defined by the radius vector.

In the following, we use spherical coordinates to denote the valid section of the sphere, according to the notation in Fig. 4.2:

- Small variance:
  - 60 < Distance (or r) < 80
  - Point of focus = $(\pm 5, \pm 5, \pm 5)$
  - $\theta = 45 \pm 10$
  - $40 < \phi < 80$
- Large variance:

- $60 <$ Distance (or r) < 120
- Point of focus = $(\pm 15, \pm 15, \pm 15)$
- $\theta = 45 \pm 35$
- $15 < \phi < 90$

4.1.4 Background elements

Background elements add additional noise (i.e., a random signal; irrelevant to the task) to the data. Learning to separate the signal from the noise adds complexity to the classification task. Five alternative elements can be placed randomly in the background, including mountains, forests, and urban skylines.

4.1.5 Contrast and Brightness

Contrast and brightness are randomly altered in each image, adding further noise and complexity. This is achieved after rendering by applying a filter to the image, using the *ImageEnhance* module from the *PIL* library. The values are set randomly between 0.5 and 1.5, given that 1 corresponds to the original image.

4.2 The ontology

4.2.1 Being ontology-independent

The VCB dataset was built specifically for running experiments, and thus, being able to make tweaks and alterations quickly and easily is a key requirement. For that reason, the generator for VCB was made independent of the ontology used to generate the images. The user need only provide a valid ontology (saved in the .owl format) that encodes what combination of features describes each class, and the generator will output an equal amount of examples for each class, according to the provided "recipes".

The ontology may also contain "ground rules" - restrictions applicable to all buildings, irrespective of class. These rules can define combinations of features that are not allowed or even some that are obligatory. For example, in the ontology used during this dissertation, no building of any class is allowed to feature awnings if it does not feature either a door or a window. In cases where a new ontology contains the same ground rules as the one used to generate an existing set of images, one could skip regenerating the images and simply re-label the existing ones. Indeed, that functionality was also implemented.

4.2.2 Ontology Expressions

In the following, we show the ontology used in the experiments carried out in this dissertation.

Firstly, the axioms that every ontology must have to be valid:

$$\text{owl\_thing} \equiv Building \sqcup Feature$$

"All individuals are either a Building or a Feature."

$$\text{Building} \equiv Industrial \sqcup Residential \sqcup Commercial$$

"Buildings are at least one of these: (...), and all of them are Buildings."

$$\text{Feature} \equiv Awning \sqcup Billboard \sqcup Car \sqcup Chimney \sqcup Door \sqcup Machine \sqcup Pipe \sqcup Porch \sqcup Sign \sqcup Statue \sqcup Table \sqcup TiledRoof \sqcup Truck \sqcup VendingMachine \sqcup WallSign \sqcup Window$$

"Features are at least one of these: (...), and all of them are Features."

Secondly, axioms that, if changed, can alter what combinations of features are "allowed" to happen:

$$\text{Building} \sqsubseteq \neg((\exists has.Car \sqcup \exists has.Truck) \sqcap (\exists has.Machine))$$

"A Building can not have both a vehicle and a machine."

$$\text{Building} \sqcap \neg(\exists has.Door \sqcup \exists has.Window) \sqsubseteq \neg\exists has.Awning$$

"A Building with no doors or windows can not have awnings."

$$\text{Building} \sqsubseteq \neg(\exists has.Car \sqcap \exists has.Truck)$$

"A Building can not have both a car and a truck"

$$\text{Building} \sqsubseteq \neg(\exists has.Chimney \sqcap \exists has.Statue)$$

"A Building can not have both a car and a truck"

Thirdly, the remaining axioms: changing these would merely result in some changes in higher-level labels:

$$\text{Commercial} \equiv Cafe \sqcup Hotel \sqcup MiscCommercial \sqcup Restaurant \sqcup Store$$

"Commercial Buildings are at leas one of these: (...), and all of them are commercial buildings"

$$\text{Industrial} \equiv ConstructionSite \sqcup MiscIndustrial \sqcup PowerPlant \sqcup WaterTreatment$$

"Industrial Buildings are at leas one of these: (...), and all of them are industrial buildings"

$$\text{Residential} \equiv CountryHouse \sqcup MiscResidential \sqcup Suburban$$

"Residential Buildings are at leas one of these: (...), and all of them are residential buildings"

$$\text{Cafe} \equiv \exists has.Statue \sqcup \exists has.VendingMachine)$$

"Any Building that has a statue and a vending machine is a Cafe (and vice-versa)"

$$\text{Hotel} \equiv \exists has.WallSign$$

"Any Building that has a wall-sign is a Hotel (and vice-versa)"

$$\text{MiscCommercial} \equiv \exists has.Awning \sqcap \exists has.Table$$

"Any Building that has Awnings and a table is a MiscCommercial (and vice-versa)"

$$\text{Restaurant} \equiv (\exists has.Car \sqcup \exists has.Truck) \sqcap \exists has.Sign$$

"Any Building that has a vehicle (Car or Truck) and a Sign is a Restaurant (and vice-versa)"

$$\text{Store} \equiv \exists has.Billboard$$

"Any Building that has a Billboard is a Store (and vice-versa)"

$$\text{Industrial} \sqsubseteq \neg \exists has.Table$$

"No Industrial Building has a table (and vice-versa)"

$$\text{ConstructionSite} \equiv \exists has.Machine$$

"Any Building that has a Machine is a ConstructionSite (and vice-versa)"

$$\text{MiscIndustrial} \equiv \neg \exists has.Awning \sqcap \exists has.Truck$$

"Any Building that has no Awnings and a Truck is a MiscIndustrial (and vice-versa)"

$$\text{PowerPlant} \equiv \exists has.Chimney \sqcap \exists has.Pipe$$

"Any Building that has a Chimney and a Pipe is a PowerPlant (and vice-versa)"

$$\text{WaterTreatment} \equiv \exists has.Pipe \sqcap \exists has.Truck$$

"Any Building that has a Pipe and a Truck is a WaterTreatment (and vice-versa)"

$$\text{Residential} \sqsubseteq \neg(\exists has.Chimney \sqcup \exists has.Pipe)$$

"Residential Buildings do not have Chimneys nor Pipes"

$$\text{CountryHouse} \equiv \exists has.Car \sqcap \exists has.TiledRoof$$

"Any Building that has a Car and a TiledRoof is a CountryHouse (and vice-versa)"

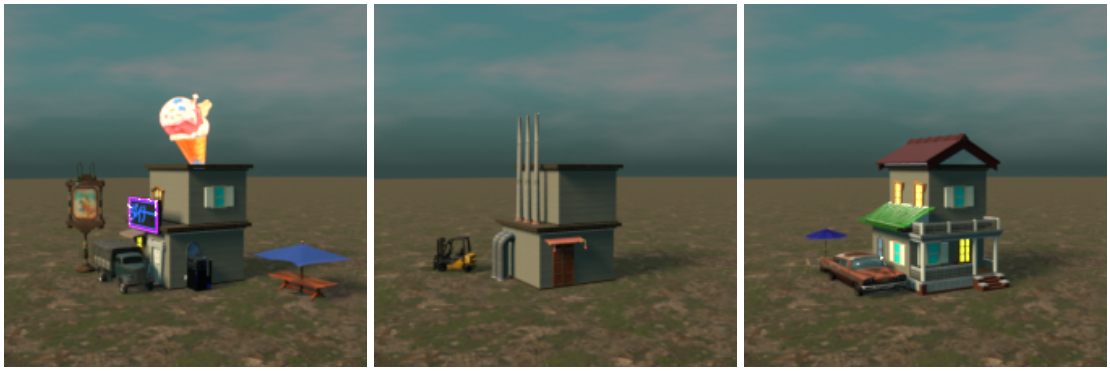
$$\text{MiscResidential} \equiv \neg(\exists has.Awning \sqcap \exists has.Table) \sqcap \exists has.TiledRoof$$

"Any Building that has a TiledRoof and does not have both an Awning and a Table is a MiscResidential (and vice-versa)"

$$\text{Suburban} \equiv \exists has.Porch$$

"Any Building that has a Porch is a Suburban"

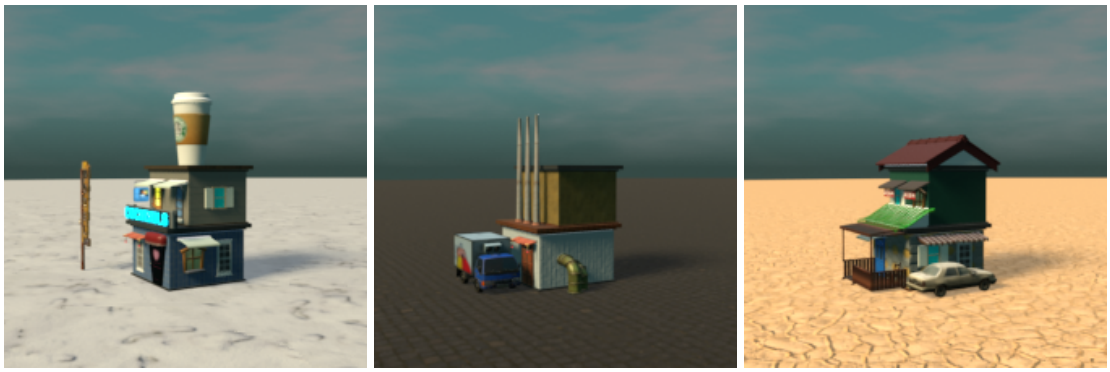
Figures 4.3, 4.4, 4.5 4.6, 4.7 and 4.8 show examples of each main class (Commercial, Industrial and Residential) in the VCB dataset, for each level of complexity.



(a) Commercial (b) Industrial (c) Residential

Figure 4.3: Examples from VCB - Complexity 1.

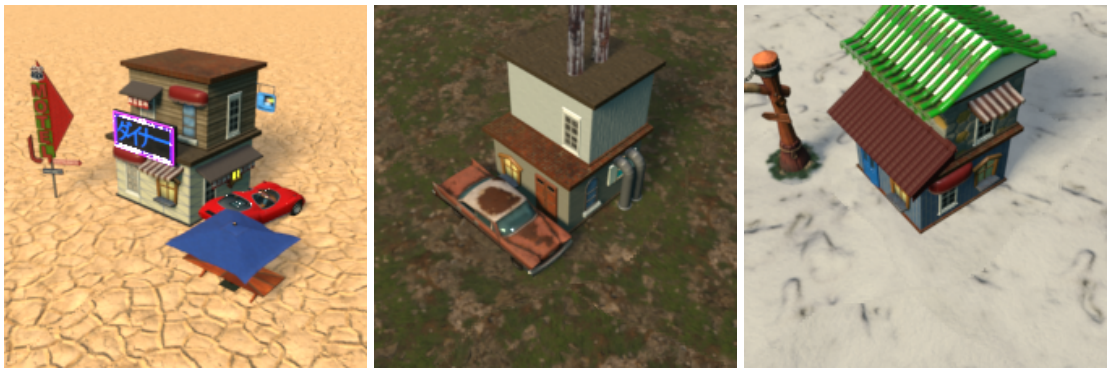
The textures and camera position are static; only the models are varied.



(a) Commercial (b) Industrial (c) Residential

Figure 4.4: Examples from VCB - Complexity 2

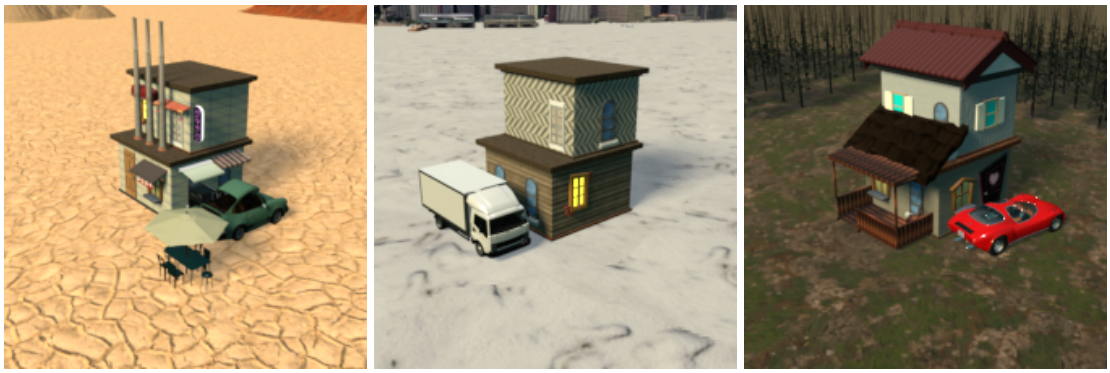
Introduces texture variation, the camera remains static.



(a) Commercial (b) Industrial (c) Residential

Figure 4.5: Examples from VCB - Complexity 3

Introduces small camera variations.



(a) Commercial (b) Industrial (c) Residential

Figure 4.6: Examples from VCB - Complexity 4

Introduces background elements.

(a) Commercial (b) Industrial (c) Residential

Figure 4.7: Examples from VCB - Complexity 5

Introduces large camera variance.

(a) Commercial (b) Industrial (c) Residential

Figure 4.8: Examples from VCB - Complexity 6

Introduces contrast and brightness variance.

4.3 How the datasets were generated

The VCB dataset was generated using the open source 3D creation suite Blender [18]. Blender includes an API that offers most of the operations available via its GUI in a Python [67] context. We created a script that uses that API to render images and store their respective labels. While it is possible to run python scripts on Blender from the built-in text editor, we ran the script from the console by providing a python script to the blender command because that allows Blender to run in the background, without the GUI.

The python script has several command-line arguments, namely the number of examples to generate (-n), the complexity-level to use (–complexity or -c), the (path to the) ontology to be followed (–ontology), the desired resolution of the pictures (–resolution) and whether to keep all, none or a specific number of the images in the .png format. All the generated images are stored in a .npz file for practicality and faster imports into other applications.

The script starts by loading the blender project containing all relevant assets. Then, after setting the render options, the script stores pointers to the data blocks containing the camera object, all of the various models (windows, signs, doors, walls, floor, background, etc.), and all of the various textures. The models, in particular, are wrapped in a custom 'Model' class that abstracts the most common model-related operations behind its various methods. Objects can be shown or hidden from the render, moved, rotated, or snapped back to their original location.

Then the script instantiates a custom 'OwlClass' object for each of the child concepts of the main classes - Residential, Commercial, Industrial - in the ontology. When instantiated, an object of this class collects all of the information about this class present in the ontology and simplifies it into the disjunctive normal form (DNF), a disjunction of conjunctions of literals representing low-level ontology concepts (i.e., $\exists has.[someFeature]$). This expression acts as a set (disjunction) of different combinations of features (conjunctions) that can be used to create examples of the respective class. After being instantiated, an 'OwlClass' can be asked to generate a random feature-set that describes an individual of this class. Additionally, this class can be asked to evaluate whether an existing feature-set describes an individual of its class.

Then the script starts generating the images. For each of the images, 77 labels are saved. Some contain ontology-related data, while others contain details about the generation of the images. There are three for the three main classes, twelve for the subclasses in the ontology (e.g. 'Cafe'), eighteen for the low-level features (TiledRoof is divided into TiledRoof, TiledRoofBottom, and TiledRoofTop, as those are independent), seventeen for the position of the features (TiledRoof is not included in the position, only the top, and bottom variants), seventeen for the model variants used for each feature, two for the number of windows and awnings, three for storing the wall, overhang and floor textures, two for the camera's position and orientation, one for the background model and two for

the contrast and brightness values. The labels of an image could be used to generate an exact copy of that image, as they describe it completely.

To populate the labels for a given image, the script starts by selecting a subclass of the three main classes for the current example and asking its respective 'OwlClass' object for a random feature-set that describes that class. Then, the script checks whether the chosen feature-set describes any of the other high-level classes (using the above described 'OwlClass' method) and adjusts the labels accordingly. Notably, the script generates an equal amount of examples of each class. The script then randomizes the textures, camera, and background object according to the chosen complexity level.

Then, the script starts placing the actual models on the building. It is important to note here that the script includes collections of pre-set positions that each feature adheres to. There are two main classes of objects in this phase: those that share that collection with other objects and those that don't. Objects that don't share space (e.g., Billboards, wall signs, chimneys, etc.) can be placed in a random position from their respective collection. Models that do share space are placed sequentially, and each one has to select from the positions left free by the previous ones. The windows and awnings are special cases, as they can also vary in quantity. A random amount of windows is selected from the available wall spaces (A door may be occupying one of the spaces), and the selected amount is placed at random positions. The awnings can only be placed on top of either a door or a window, so that sets the upper limit for their amount. At this stage, the needed models, which are normally not visible in the render, are made visible and moved to position according to the decisions made.

Finally, the image is rendered and saved, and a new row of labels is added to the table containing the data from all previously generated examples. We used Blender's built-in image rendering engine *Eeve* for the generation because it is very optimized and provides the fastest render times. After this, all objects are hidden from the render, and the process starts again for the next picture.

5

Empirical evaluation

This section contains details about the empirical tests conducted in this dissertation. We detail the used datasets, the implementation of DARTS for use in concept mapping, how the main networks were built and trained, and the methodology used in the experiments.

5.1 Specifications of the hardware used

All the computation carried out in the course of this dissertation - generating the VCB dataset, training the main networks, running the modified DARTS algorithm, etc. - was done on a single desktop computer with the following specifications:

- **Operating System:** Windows 11 Pro
- **Processor:** Intel(R) Core(TM) i9-9900K CPU
- **Graphics Card:** NVIDIA GeForce RTX 2080
- **Memory:** 48.0 GB

5.2 Specifications of the used datasets

Rendering thousands of images, each possibly containing dozens of complex models, requires significant computational resources. Indeed, using the hardware described in Section 5.1, while the logic of the generator script takes about 50ms per image, the rendering takes five to upwards of ten seconds, depending on the complexity level. So, a dataset with twenty thousand examples takes around one-and-a-half days to generate for lower complexities and two-and-a-half days for higher ones, which is a considerable amount of time. Further emphasizing this point is the fact that we aimed to generate six separate datasets. This motivates us to choose the number of examples per dataset carefully; so as not to waste time and resources. In the course of this dissertation, the datasets are required for two tasks: training the main networks and training the mapping networks. Regarding the mapping networks, we use four thousand examples across the training,

validation, and test sets. That may seem like a low number, but note that the examples used for concept mapping have to be additionally labeled with the desired secondary labels, which is an expensive and time-consuming process. Further, if the concepts are chosen adequately, and the main network has learned useful representations of the data, we should expect that not many examples be needed to learn a mapping of said concepts. Indeed, Sousa Ribeiro and Leite[60] only used two thousand total examples in their experiments. The lower bound of needed examples then falls on the number of images required for the main network to converge and generalize well, which we define by an accuracy of around ninety-five percent on the test set. In the end, the datasets of complexities one, two, and three contain twenty thousand images each, and the datasets of complexities four, five, and six contain forty thousand images each.

5.3 Adapting DARTS for concept mapping architecture search

DARTS is mainly geared toward developing Convolutional Architectures for image classification tasks.

For DARTS to be a viable tool for our purposes, some modifications must be made. We want to use DARTS to develop networks that interpret the feature maps of another model. Since the feature maps of a trained model are structured data in which the proximity of data points is irrelevant, using a convolutional neural network to interpret them would be inadequate. To tackle this, we modify the set of candidate operations in the original DARTS algorithm to contain more adequate ones. This does not compromise the principles and methodology of DARTS. After all, the concept of optimizing an architecture with respect to its validation set loss does not mention, and thus, does not depend on what set of candidate operations are used [33].

Additionally, DARTS has the advantage that it does not assume the size of the final model; the same learned cell can be copied any number of times to build a final model, which is desirable both because we do not have to conform to sizes chosen for different types of tasks, but also because this allows us to conduct experiments with models of varying depths, for the same learned architecture. Moving forward, we will refer to our adaption of DARTS for learning concept mapping architectures as DARTS-CM.

5.3.1 Including linear architectures in the search space

The original DARTS implementation does not allow linear networks to be found. Indeed, the transformations applied to the data before it is even fed to the first cell include non-linear activation functions, as seen in Fig. 3.8. Additionally, all candidate operations excluding the *skip_connect*, *zero*, and the pooling operations also use non-linear activation functions. However, the search space of the NAS algorithm we use should include linear networks, so as to increase confidence in the fact that, when the algorithm converges to a non-linear architecture, that is truly because there are performance advantages in

doing so. After all, we should expect the algorithm to "prefer" simpler architectures since those with fewer parameters converge faster, and so, if they are sufficient for the task, should be expected to provide better early performance than more complex alternatives. To this end, we remove the non-linear activation functions from the pre-processing and post-processing operations of the network, leaving only Dense layers that resize the data. Additionally, our set of candidate operations includes a Dense operation without an activation function. In fact, most of the candidate operations used do not include a non-linear activation function.

5.3.2 Set of candidate operations

The set of candidate operations defines the data transformations available to our algorithm for "coming up" with an architecture. They define what the search space actually is. In the experiments carried out in this dissertation, the set of candidate operations is as follows:

- The **Skip Connect** operation encodes the identity operation. This means that no transformation is applied to incoming data and that it is simply forwarded to the subsequent nodes.
- The **Batch Normalization** [26] operation makes normalization a part of the architecture by normalizing each input mini-batch during training. This allows for the use of much higher learning rates and makes the model less responsive to changes in the initial weights.
- The **Layer Normalization** [2] is similar to Batch Normalization, but it normalizes all neurons in each layer equally, regardless of the current mini-batch.
- The **Dropout** [61] is a regularizing operation that randomly drops certain units during training. This prevents units from forming complex co-adaptations - i.e., extensively learning very complex patterns that are only applicable to the training set and do not generalize well - which prevents overfitting and improves performance.
- The **Gaussian Dropout** [46] achieves the regularizing effect of Dropout by applying Gaussian noise to the inputs. This was proposed as a way to mitigate the original Dropout's slow-down of training.
- The **ReLU** [34] is a non-linear function $f(x) = max(0, x)$, which returns the input if it is positive and zero otherwise. This is the default activation function for many deep learning applications because it is easy to train (i.e., cheap to compute its derivative), and it achieves better performance than alternatives.
- The **LeakyReLU** [65] is a variant of the ReLU function that has a slight slope for negative values, instead of being constant at zero, which avoids the 'Dead Neuron' problem (Section 2.1.1.4).

- The **Dense** operation encodes a fully connected layer without an activation function. This operation is included to continue ensuring that linear architectures are included in the algorithm's search space. Two additional operations are used with activation functions (one with ReLU and the other with LeakyReLU) after the dense layer.

The attentive reader might notice that this list does not include the *Zero* operation (mentioned in Section 3.3.1) used in the original DARTS paper. This is due to the fact that DARTS never selects the *Zero* operation in the discretization step, removing any advantage that the inclusion of this operation might pose. This decision further came as a result of the observation that the *Zero* operation dominates most of the edges during training, often leaving the other operations to share a small fraction of the "attention", and thus receive less optimization. This hinders the method's stability, as the model may simply ignore the optimal operation. Indeed, some subsequent DARTS-based works [31, 53, 70] have similarly noticed how the *Zero* operation disrupts the architecture search and removed it.

5.3.3 The cell structure

Similarly to the candidate operations, DARTS [33] is not specific to one type of cell structure. In the original work, Liu, Simonyan, and Yang employed two different cell structures for convolutional and recurrent architectures, according to what is optimal for either, as well as what would allow for comparison with existing NAS systems. Given that our goals are markedly different, we must analyze and choose the cell structure that best fits our purposes; optimizing architectures for concept mapping. Modern convolutional architectures [23, 63] use skip connections and branching layers, so the convolutional cell in DARTS accommodates those types of connections. However, given that the concept mapping task is much simpler than image classification tasks, a sequential architecture - one where the output of each node is simply fed to the next - is most likely sufficient. To emphasize this point, note that Sousa Ribeiro and Leite [60] managed to get satisfactory results from a single neuron, for some of the concepts. Allowing non-sequential architectures would simply hinder architecture optimization by over-expanding the search space. We adopt a cell with four nodes (i.e., four learned operations per architecture). Fig. 5.1 shows a diagram of a whole network in DARTS-CM, different from the original DARTS (Fig. 3.8) in the core aspects described above. Given that concept mapping is a binary classification task, a sigmoid operation can be applied to the network's output to obtain a probability value that the input contained a given concept.

5.3.4 Implementation details

While DARTS-CM was virtually all implemented from scratch, the original code accompanying the paper [42] (Implemented in PyTorch) and a Tensorflow 1.0 version adapted

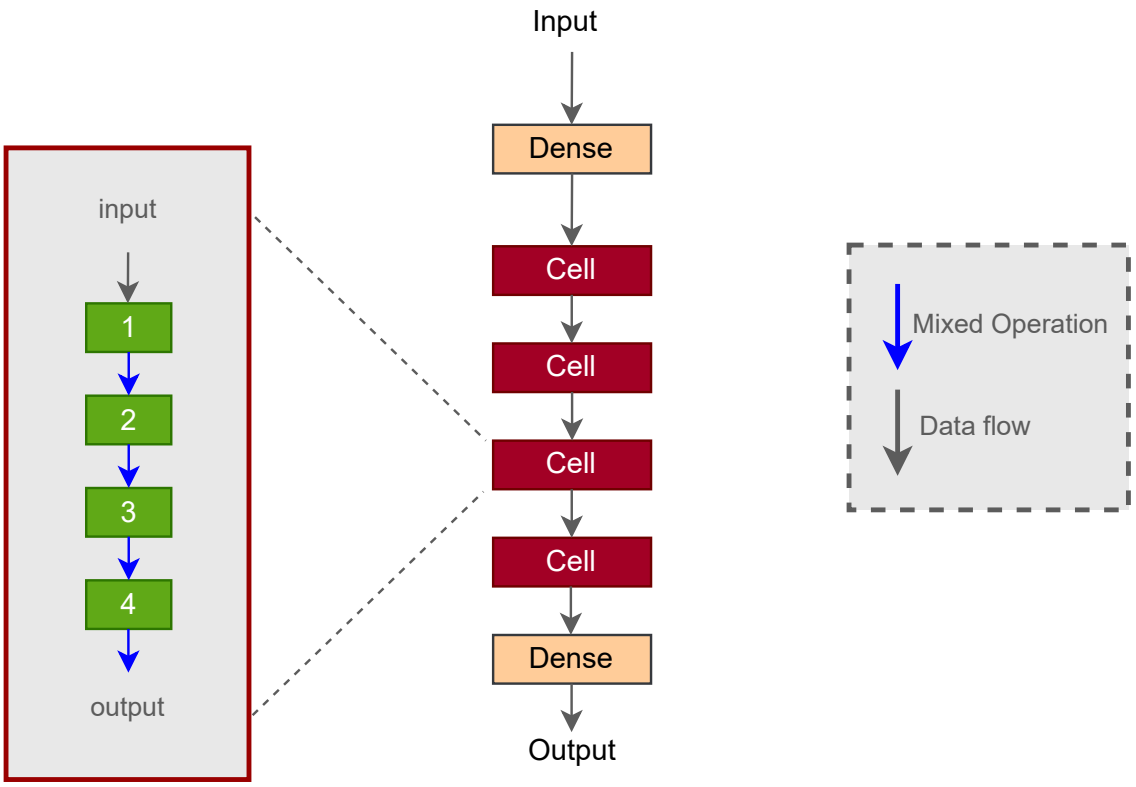


Figure 5.1: A diagram of a DARTS-CM network composed of four cells.

The diagram shows the changed linear transformations on the top and bottom of the network. Each cell receives the output of the preceding one. Each of the four nodes in the cell receives the output of the preceding node. Blue edges denote a mix of candidate operations (or, in the case of the discrete network, the learned operation), and gray edges simply denote data flow.)

from the original code [13] were used as a basis for the structure and other implementation details. Our version is implemented in Tensorflow 2, which discarded Tensorflow 1's focus on computational graphs and lazy execution for an eager execution mode.

The models used were implemented using the Keras subclassing API. The continuous and discrete model classes are subclasses of the *tensorflow.keras.Model* class, and the cells and mixed edges are subclasses of the *tensorflow.keras.layer.Layer* class. With this API, one can overwrite the methods that build the models, initialize the weights, compute a forward pass of the model, and create custom training loops, while Keras' existing implementation handles the rest. The *GradientTape* class from TensorFlow was used to perform automatic differentiation (*GradientTape* records all operations made to TensorFlow variables inside a certain context to allow for the computation of their gradient). When built, the continuous model initializes the arch weights and feeds them to each of the cells, which in turn feed them to each of their mixed edges, which use them to weight the outputs of their operations. Of note is also the fact that DARTS-CM implements the Second-order approximation version of the gradient for the arch weights, as described in

Section 3.3.2.

5.3.5 Use of feature extractors

The mapping networks receive internal activations of a main network as input. One might think that the activations must be extracted and stored in a dataset beforehand to be used for training the concept mappers. However, these internal activations are of relatively high dimensionality, meaning that it would be inefficient to store them on disk. Indeed, each mapping network may receive a different combination of the internal activations, and thus multiple of these "internal activation datasets" would have to be created and stored. To avoid doing that, we extract the activations online, as they are needed during training. This is done with code implemented by Sousa Ribeiro and Leite [60], used with permission. This code provides objects, called Feature Extractors, that can be used to - provided an image - return the main network's internal activations that result from being presented with that image. Thus, no additional data need be stored on disk; the algorithm's training data consists of the images, which are converted to their respective activations as needed.

5.4 The Main Networks

In the following experiments, we use DARTS to search mapping architectures for a set of Main Networks. These networks were trained on the VCB dataset, and there is one for each of the three main classes, for each of the six complexity levels, totaling eighteen networks. The template shown in Fig. 5.2 represents the architectures that were used, where c and d are variables that determine the number of times the respective block is stacked. Convolutional layers in the same block share the same amount of filters, which increases from block to block with the network's depth. The first Convolutional block always has thirty-two filters, and that number doubles for each subsequent one. In the Dense section, the number of neurons per layer progressively decreases into a single binary classification. The layer that directly precedes the output layer is set to have sixteen neurons, and every preceding layer has double the amount of its successor. A rate of 0.3 was used in the Dropout layers. Each model was trained for a maximum of fifty epochs, stopping earlier if no improvement to the validation loss is observed for twenty epochs, and the weights from the epoch with the lowest validation-set loss were reloaded. The weights were optimized using a Binary Cross-entropy loss function and the Adam [29] Optimizer with a learning rate 1e−3, momentum $\beta = (0.9, 0.999)$. The models for complexities one to three were trained with eight thousand examples in the training set and around twenty-seventwenty-seven hundred in each of the validation and test sets. The models for complexities four to six were trained with twelve thousand examples in the training set and four thousand examples in each of the validation and test sets. Table 5.1 shows, for each main network, the values for c and d as well as the obtained loss and

accuracy on the test-set. Ideally, the main networks for complexities five and six would have received further optimization. However, time constraints impeded that possibility.

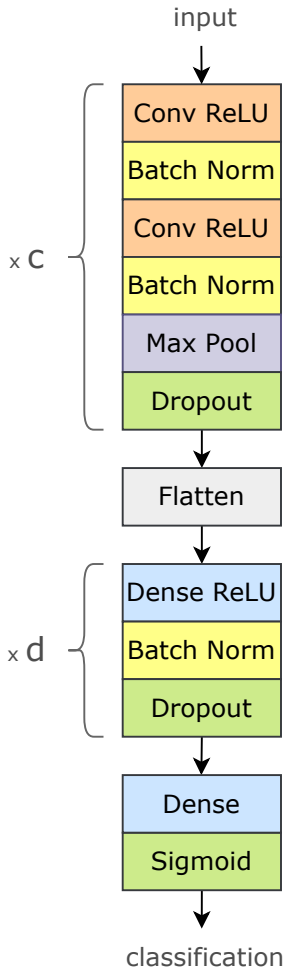


Figure 5.2: The architecture template used for the main networks.

5.5 Preliminary testing of the system on the XTRAINS dataset

This initial experiment was done as a test run of the system, and it is presented here as a proof of concept for the idea that NAS can be used to find exemplary mapping architectures. The experiment entails learning mapping architectures for the XTRAINS dataset. We can then quickly compare the results with those obtained by Sousa Ribeiro and Leite [60] to assess whether our methodology appears sound.

5.5.1 Implementation

Some aspects of the algorithm's implementation were different in this experiment than described above. Namely, the changes outlined in section 5.3.1 had not yet been implemented. Additionally, the *Zero* was still included in the set of candidate operations.

Complexity	Class	c	d	Loss	Acc
1	Commercial	3	2	0.036	0.9904
	Industrial	3	2	0.026	0.992
	Residential	3	2	0.051	0.9865
2	Commercial	3	2	0.0526	0.9814
	Industrial	3	2	0.055	0.9823
	Residential	3	2	0.0362	0.9883
3	Commercial	6	2	0.0721	0.9805
	Industrial	3	2	0.0973	0.9761
	Residential	3	2	0.0879	0.9817
4	Commercial	2	3	0.143	0.958
	Industrial	2	3	0.0743	0.9773
	Residential	2	3	0.0778	0.9827
5	Commercial	6	2	0.1545	0.9453
	Industrial	6	2	0.1325	0.955
	Residential	6	2	0.1209	0.9562
6	Commercial	6	2	0.1487	0.9456
	Industrial	6	2	0.0953	0.9741
	Residential	6	2	0.1193	0.9591

Table 5.1: Test-set performance for each main network.

5.5.2 Experiment

In this experiment, we had our DARTS algorithm learn mapping architectures for mapping a single concept from the activations of one of the three main networks. Each of the main networks has been trained to map one of the three main classes in the XTRAIN dataset: typeA, typeB, and typeC. These models are the same as those used in [60], so we can draw a direct comparison between the performance of mapping architectures for the same main network and concept.

The same amount of examples contained in the training, validation, and test sets is eight hundred, two hundred, and a thousand, respectively. These values were used because they match the ones used in [60], further facilitating comparisons.

5.5.3 Results

This experiment provided promising results. After training the automatically-found mapping networks, we find that they are, at worst, equivalent to the linear models used in [60], with some cases of a significant gain in performance. Tables 5.2, 5.3 and 5.4 show the comparisons of the test-set performance between the mapping networks that our DARTS algorithm found and the linear ones used in [60]. We can see that the best improvement we got was in mapping FreightWagon from the typeA network, where there was a 3.6 percentage point increase in accuracy. The worst result was a decrease of 0.4 percentage points when mapping MixedTrain from the typeC network. This was the only case where the DARTS algorithm did not perform better than the linear networks. This

could be a situation where the linear model was indeed enough to map the concept fully, and introducing more complexity just slowed convergence, albeit not by much. Across all concepts, there was an average of about one percentage point increase in accuracy.

Type A		
	DARTS-CM	Sousa Ribeiro and Leite
Empty Train	99.8	99.4
Freight Wagon	96.2	92.6
Reinforced Car	99.8	99.2
War Train	99.5	99.3

Table 5.2: Comparison of accuracy on the test set between mapping architectures for Type-A's relevant concepts.

Type B		
	DARTS-CM	Sousa Ribeiro and Leite
Freight Train	96.6	95.2
Freight Wagon	97.6	96.2
Long Train	98.4	97
Passenger Train	96.1	94.8

Table 5.3: Comparison of accuracy on the test set between mapping architectures for Type-B's relevant concepts.

Type C		
	DARTS-CM	Sousa Ribeiro and Leite
Freight Wagon	95.3	95
Long Wagon	98.9	98.1
Mixed Train	97.1	97.5
Rural Train	99.4	99.1

Table 5.4: Comparison of accuracy on the test set between mapping architectures for Type-C's relevant concepts.

These preliminary results confirm that our DARTS algorithm is succeeding at optimizing the mapping architectures. Further, it indicates that non-linear architectures can indeed provide a performance gain for the concept mapping task.

5.6 Extensive architecture comparison

In this section, we describe the extensive comparison of concept mapping architectures that was carried out. To allow for direct comparison, the same set of concepts was mapped for each main class of examples. Specifically, three concepts were selected for each main class - Residential, Commercial, and Industrial - to be mapped from the activations of

the respective main networks of all six complexities. The three concepts were chosen so that, for each main class, there is:

- A **High-level** concept, which are the complex, intermediary concepts that are defined in terms of other, lower-level ones but that, in turn, are used to define the main classes. For example, the concept 'Restaurant' is defined as a building having either a Car or a Truck and a sign.
- A **Trivial low-level** concept, which is a concept of the form $\exists has.[someFeature]$ (low-level) that can be mapped directly to a main class (trivial). For example, the concept '$\exists has.Billboard$' is equivalent to the concept 'Store', which, in turn, is included by the main class 'Commercial'.
- A **Non-trivial low-level** concept, which is a low-level concept that can not be directly mapped to a main class. For example, the concept $\exists has.TiledRoof$ does not offer any certainty of an individual belonging to a particular main class.

In the following, for the sake of readability, we use $someFeature$ as a short-hand for the trivial concept $\exists has.[someFeature]$. The concepts selected for use in the learning and testing of concept mapping architectures in the experiments carried out in the course of this dissertation are the following:

Main Class	High-level	Trivial low-level	Non-trivial low-level
Commercial	Restaurant	Billboard	Statue
Residential	MiscResidential	Porch	TiledRoof
Industrial	MiscIndustrial	Machine	Chimney

Testing concept mapping networks for each of the nine selected concepts for each of the six complexities yields fifty-four models per architecture.

All the architectures received as input the same set of activations from the main networks: the outputs of the flatten layers and of all subsequent batch normalization layers. The flatten layer directly follows the convolutional part of the main network, where relevant features in the image have been extracted and abstracted from their positions in the input. Similarly, the output of the batch normalization layers may contain other useful representations of those features, produced by the preceding dense layers and respective activation functions. We use the output of the batch normalization layers instead of directly using the output of the dense layers because normalized data is generally better for machine learning algorithms.

Regarding the size of the data used, sixteen hundred examples were used in each of the training and validation sets, and eight hundred were used in the test set. We use the same amount of examples for training and validation because neural architecture search uses the validation data to learn optimal architectures, and having enough examples is required for it to do so well. Further, we ensured that an even amount of positive and negative examples were used for each concept being mapped.

5.6.1 Learning architectures with DARTS-CM

This experiment entails running DARTS-CM to learn architectures for the fifty-four previously described configurations. For each configuration, the model and architecture weights of the continuous model are jointly optimized for a maximum of fifty epochs, stopping earlier if no improvement to the validation-set loss is observed for twenty successive epochs. Both the model and architecture weights from the epoch with the lowest validation-set loss value are reloaded.

The continuous model is discretized by picking the strongest operation for each edge in the cell according to the corresponding architecture weights. Following this, discrete models composed of one, two, three, and four cells are built and trained from scratch on the same data for a maximum of fifty epochs, stopping earlier if no improvement to the validation-set loss is observed for fifteen epochs with similar reloading of the best weights. This process is repeated five times for each of the four discrete models, and results are averaged to make them less dependent on weight initialization. After training, each model is evaluated on the test set, and the observed loss and accuracy values are recorded.

The model weights, in both the architecture search phase and the training of the discrete model, are optimized using a Binary Cross-entropy loss function and the Adam [29] Optimizer with a learning rate 1e−3, momentum $\beta = (0.9, 0.999)$. The architecture weights are optimized using the Adam Optimizer, with initial learning rate of 3e−4, momentum $\beta = (0.5, 0.999)$. These were used as they were the values used in the original DARTS work.

As a reminder, DARTS finds an optimal cell architecture and builds the final network by vertically stacking a configurable amount of these cells. Note that while the cells share the arch weights, the particular parameters of their operations - namely, the number of neurons in the dense layers - are subject to variation. We use a continuous model with two cells with 64 neurons in the first cell and thirty-two neurons in the second. For the discrete models, the last cell in the model is set to have an output size of sixty-four, with each previous cell having two times the size of its successor.

The continuous model is trained with a batch size of sixty-four, while a batch size of thirty-two is used for the discrete models.

5.6.2 Manually created architectures

To measure the usefulness of DARTS-CM, we tested a suite of manually created architectures on the previously described fifty-four configurations. Firstly, as a baseline, we include a model with a single output neuron (a fully connected layer with one neuron, followed by a sigmoid activation function), as is used in the current literature [60]. We then add a set of architectures that a user might try to the suite.

For the sake of brevity and readability, we use the following abbreviations:

- **Dense(n, a)**: Dense layer with **n** neurons and **a** activation function.
- **BN**: A Batch Normalization layer
- **DO**: A Dropout layer with a rate of 0.3

Further, layers are shown from left to right and separated by hyphens. The following is a list of the manual architectures that were tested:

1. Dense(1, sigmoid) *(Baseline)*
2. Dense(128, ReLU) - Dense(1, sigmoid)
3. Dense(128, ReLU) - BN - DO - Dense(1, sigmoid)
4. BN - Dense(1, sigmoid)
5. DO - Dense(1, sigmoid)
6. BN - DO - Dense(1, sigmoid)
7. BN - Dense(128, ReLU) - BN - Dense(1, sigmoid)
8. DO - Dense(128, ReLU) - DO - Dense(1, sigmoid)
9. BN - DO - Dense(128, ReLU) - BN - DO - Dense(1, sigmoid)

Notably, DARTS-CM always has a Dense layer before the first cell. Therefore, the architectures that feature either a Batch Normalization or Dropout layer before the first Dense layer (4-9) are not included in DARTS-CM's search space.

The chosen architectures were inspired by the work of Chen et al. [7] on independent component layers. Summarily, this work proposes combining the Batch Normalization and Dropout - which they call an independent component (IC) layer - before the first weight layer. The IC layer was proposed as a computationally cheap way of whitening the data; to transform the input into a set of uncorrelated variables with unit variances. This process was experimentally shown to result in a more stable training process, faster convergence speed, and better convergence limit on CIFAR10/100 and ILSVRC2012 datasets [7].

Each model is trained for a maximum of fifty epochs, stopping earlier if there is no improvement to the validation loss for fifteen epochs. The model uses a Binary Cross-entropy loss function and the Adam [29] Optimizer with a learning rate 1e−3, momentum $\beta = (0.9, 0.999)$.

6

Results and discussion

In this chapter, we show and discuss the results of the empirical evaluation described in Chapter 5. In total, four thousand and fifty individual networks were trained, five trials for each of the eight hundred and ten different (architecture, complexity, concept) combinations. We will compare the loss value obtained by these architectures in the test set after convergence. To aggregate the data, we separate the models into four categories:

- Architectures found by DARTS-CM
- Manual architectures that are included in DARTS-CM's search space (*Inc.*)
- Manual architectures that are not included in DARTS-CM's search space (*Not Inc.*)
- 1 Neuron

Doing a simple tallying of the times that each category obtains the best results out of the four, we get the following scoreboard:

Category	Times best performing
DARTS-CM	21
Not Inc.	17
Inc.	15
1 Neuron	1

Table 6.1: Number of times each category obtained the best performing architecture

DARTS-CM obtained the best performing architecture the most times out of any category. *Not Inc.* and *Inc.* share a similar amount of victories, and the 1 Neuron wins only once. Table 6.2 shows these scores divided per complexity level. We can see that DARTS-CM has the biggest advantage on the lower complexities. However, it is also the only category that does not have zero wins on any complexity, demonstrating a higher resilience to change. Immediately, the results present a case for DARTS-CM, even if not an outright disqualification of the trial-and-error method. Notably, we can see that in the higher complexities the *Inc.* proves insufficient, with only the *Not Inc.* category, made up of models more complex than any that DARTS-CM can learn, getting better results than

DARTS-CM. This indicates that DARTS-CM is the right approach and could obtain even better results with some improvement. In the following sections, we will go into deeper detail to help weigh each solution's advantages and disadvantages for different scenarios.

Complexity	Category	Times best peforming
1	DARTS-CM	6
	Not Inc.	2
	Inc.	1
	1 Neuron	0
2	DARTS-CM	4
	Not Inc.	3
	Inc.	2
	1 Neuron	0
3	DARTS-CM	3
	Not Inc.	2
	Inc.	4
	1 Neuron	0
4	DARTS-CM	2
	Not Inc.	0
	Inc.	7
	1 Neuron	0
5	DARTS-CM	3
	Not Inc.	5
	Inc.	1
	1 Neuron	0
6	DARTS-CM	3
	Not Inc.	5
	Inc.	0
	1 Neuron	1

Table 6.2: Number of times each category obtained the best performing architecture per complexity

6.1 Comparison without selection

Figure 6.1 shows the distribution of the test-set loss value obtained by the models in each category. The density of the values in a given range is given by the width of the graph in that range. The values shown include the results obtained by all of the architectures in each category prior to selecting the best performing model for each one. Note that the scale differs from graphic to graphic. The significant changes in the ranges of values between complexities make a fixed scale impractical. The same applies to the other sets of graphics in this section.

The first thing of note is that, while there are significant differences and visible patterns along the multiple categories for complexities one to four, complexities five and six show much more homogeneous results. If this was simply a consequence of the increase

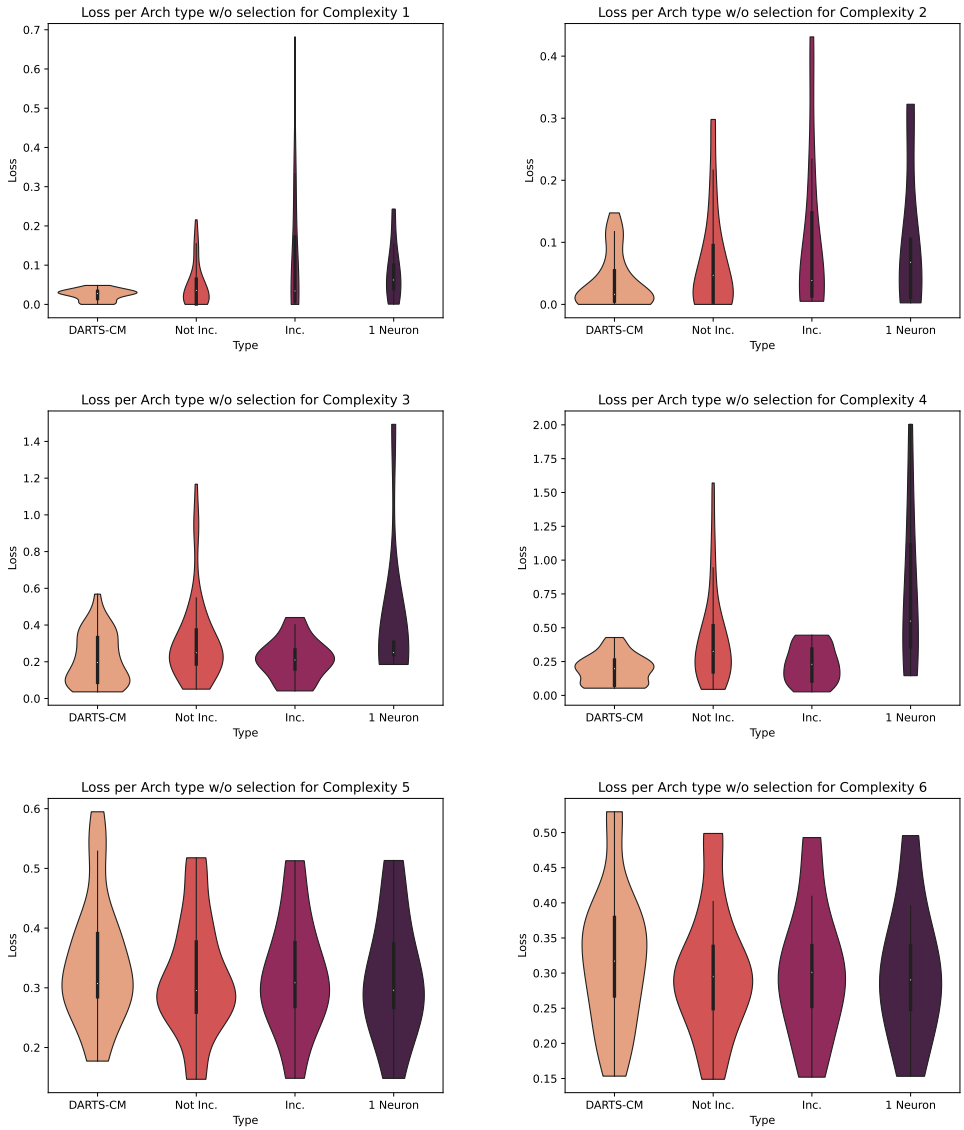


Figure 6.1: The distribution of the loss values obtained by the tested architectures, grouped by category, and with no selection of the best out of each category.

in complexity, we should expect to see a smooth transition, not a sudden change at complexity five. We believe this phenomenon owes itself to the fact that the main networks for complexities five and six did not learn the main task to the same level as the ones for lower complexities (Table 5.1), effectively setting a bottleneck on concept-mapping performance, and eliminating the advantage that more apt architectures have over lesser ones. After all, note that the labels used for concept mapping are not ground truth. They represent assumptions that if a certain concept is present in the input, the main network will have identified it. If the main network only correctly identifies a certain concept 80% of the time, then a perfect mapping network - which always correctly predicts whether the main network found the concept - would obtain 80% accuracy. For the 20% of examples for which the main network does not correctly extract the concept, the mapping

network's correct predictions would not correspond to the labels - which simply describe whether a concept is present in the input. This illustrates an interesting point: Concept Mapping, as a methodology for explainable AI, is only ever as good as the model being interpreted, and thus, it is most useful when applied to a well-trained high-performing main network.

Another pattern that begs attention is that the loss values obtained across the board for complexities three and four are significantly higher than for complexities one and two. In this case, however, the respective main networks obtained similar performance. This seems to support the hypothesis that the complexity of the main classification task affects concept mapping. Further, there is a significant jump in loss from complexity two to three, whereas the increase from one to two and three to four is much less pronounced. This is likely a consequence of camera variation being first introduced in complexity three. This affects a change in the filters learned by the convolutional layers, as they now have to accommodate for the same features appearing from different angles. Our results show that this change significantly impacts the concept mapping task, further supporting the argument for a case-by-case evaluation of architectures for concept mapping in lieu of a one-size-fits-all solution.

Let us now consider the relative performance of the candidate architectures on complexities one to four. DARTS-CM exhibits lower variance in the loss values, providing more consistent and predictably good performance. As a matter of fact, it exhibits the lowest upper bound for three out of the four complexities, having a slightly larger range than the *Inc.* category for complexity 3. The architectures in the *Not Inc.* and *Inc.* show relatively similar distributions, with the highest density at low loss values. Both these categories, tough, display, seemingly at random, outliers with much higher loss values than the other candidates, which evidence a high instability of performance across concepts. The *1 Neuron* category exhibits a more so even distribution of values and a relatively large variance. This is to be expected: the 1-neuron architecture assumes that the presence of the concepts in the input is a linear function of the main network's activations. When that assumption holds, the 1-neuron model achieves equal performance to the other candidates, but when it does not, the same model does not manage to compete and achieves much worse results. Nevertheless, the 1-neuron model's loss distribution shows a slightly higher density at lower values, indicating that a significant portion of concepts can be linearly mapped to at least some capacity.

In Figure 6.2 we plot the distribution of the difference between the loss value of each architecture and the one from the best performing model for the same (complexity, concept) combination. The difference is zero for the best performing architecture, so a high density near or at zero means that that category is often the best or close to the best. The lines are cut-off at the minimum and maximum observed values, to allow for accurate visualization of their range.

DARTS-CM is the clear winner in complexities one to four, exhibiting a much higher density of values at or near zero and a smaller range of values. For complexities five and

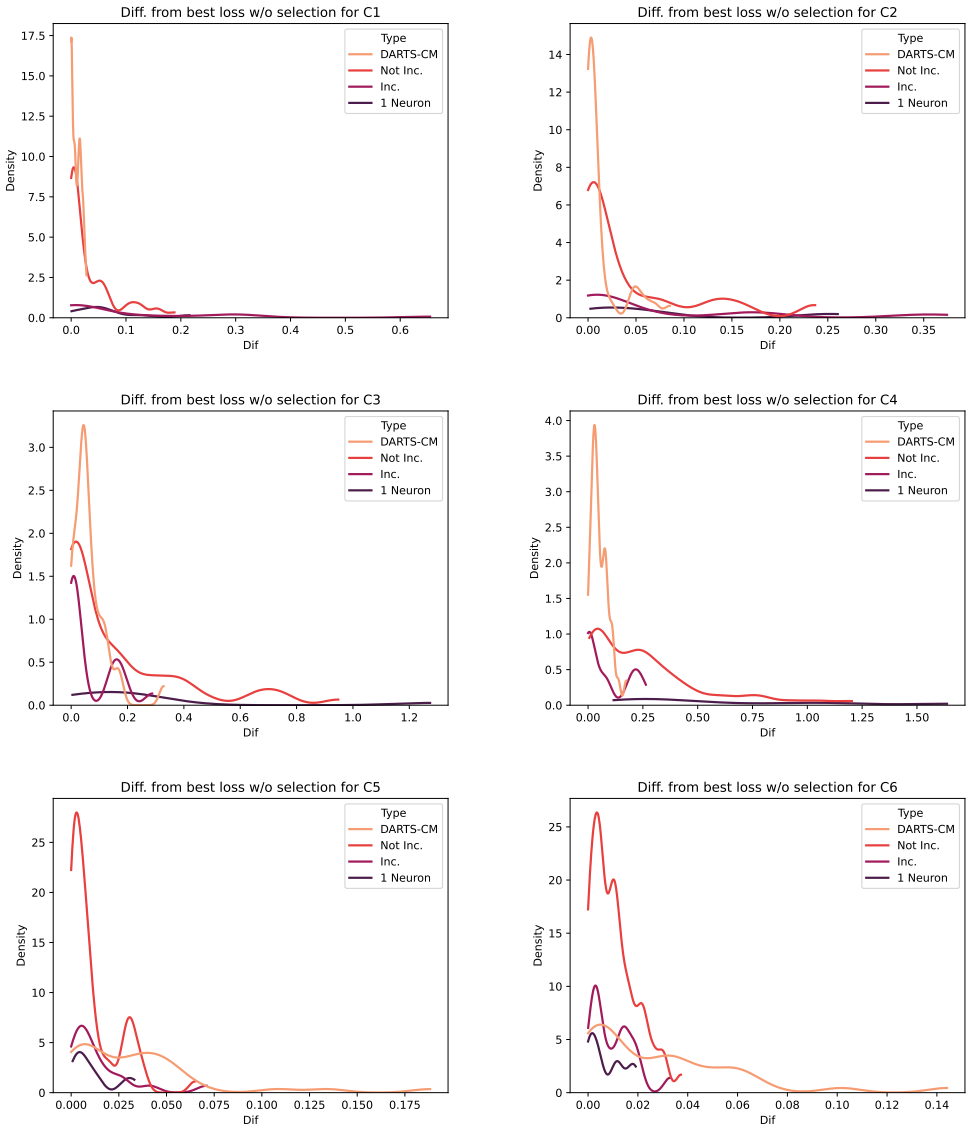


Figure 6.2: The distribution of difference of loss to the best performing architecture, before selection of the best performing for architecture.

six, we see a sharp reversal, with the *Not Inc.* category showing the better distribution. DARTS-CM seems to suffer significantly with the sub-optimally trained main networks of complexities five and six, just as the concept mappers do. With complexities five and six, DARTS-CM seems hampered in its previously demonstrated ability to optimize the architecture, producing sub-par results. This provides a critical insight. Automated search methods for architectures do not conform to architecture design standards, which have evolved through extensive empirical experimentation and evaluation of what generally works well. That added freedom has the advantage of removing upper limits on performance that the standards can place, but it also removes the lower ones. Free from pre-established standards, DARTS-CM relies instead on the information provided by the main network: how its internal activations map to relevant concepts in the input and how

those mappings can best be learned. The quintessential information-technologies maxim neatly summarizes this point: "Garbage in, garbage out". One can imagine the following edge case: If asked to learn an architecture for a truly random function, DARTS-CM, computing the gradient of the validation loss with respect to the arch weights, would get random results that depend solely on whatever train batch it is processing at that point. The gradient would contain *no* useful information whatsoever, and the resulting architectures would contain similarly random assortments of operations, only by chance resembling a sensible design.

6.2 Comparison with selection

For this comparison, for each category and each of the fifty-four (complexity, concept) combinations, we kept the model with the lowest loss value obtained by any model in that category. This, essentially, represents doing manually, for the *Inc.* and *Not Inc.* categories, what DARTS-CM does automatically by discarding non-optimal architectures. Besides the fact that one is done automatically and the other manually, the difference is that DARTS-CM does not have to train its candidate networks fully to evaluate them and that DARTS-CM's search space is much larger.

Fig. 6.3 shows the distribution of the best test-set loss value for each category and complexity. As expected, DART-CM'S distribution suffered the most minor change compared to Fig. 6.1. In contrast, the *Inc.* and *Not Inc.* categories observed a sizeable reduction in the range of values after pruning the sub-optimal architectures. While DARTS-CM still shows clearly better results for complexity one, the other candidates now have lower upper bounds in complexities two and three. When it comes to the distribution of the difference to the best performing architecture (Fig. 6.5), we see the manual pruning of sub-optimal architectures being more reliably low than DARTS-CM for higher complexities.

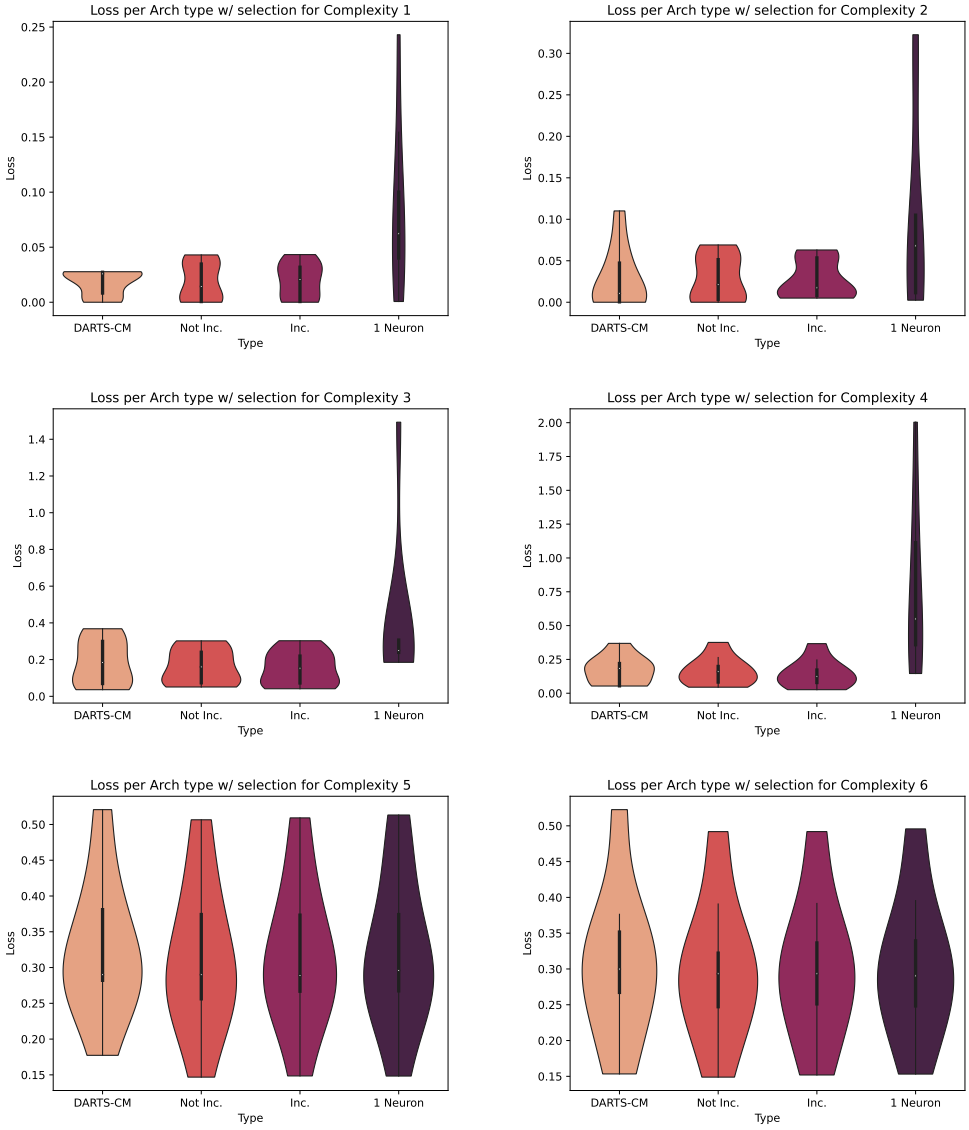


Figure 6.3: The distribution of the lowest loss values obtained by the best performing architectures, grouped by category.

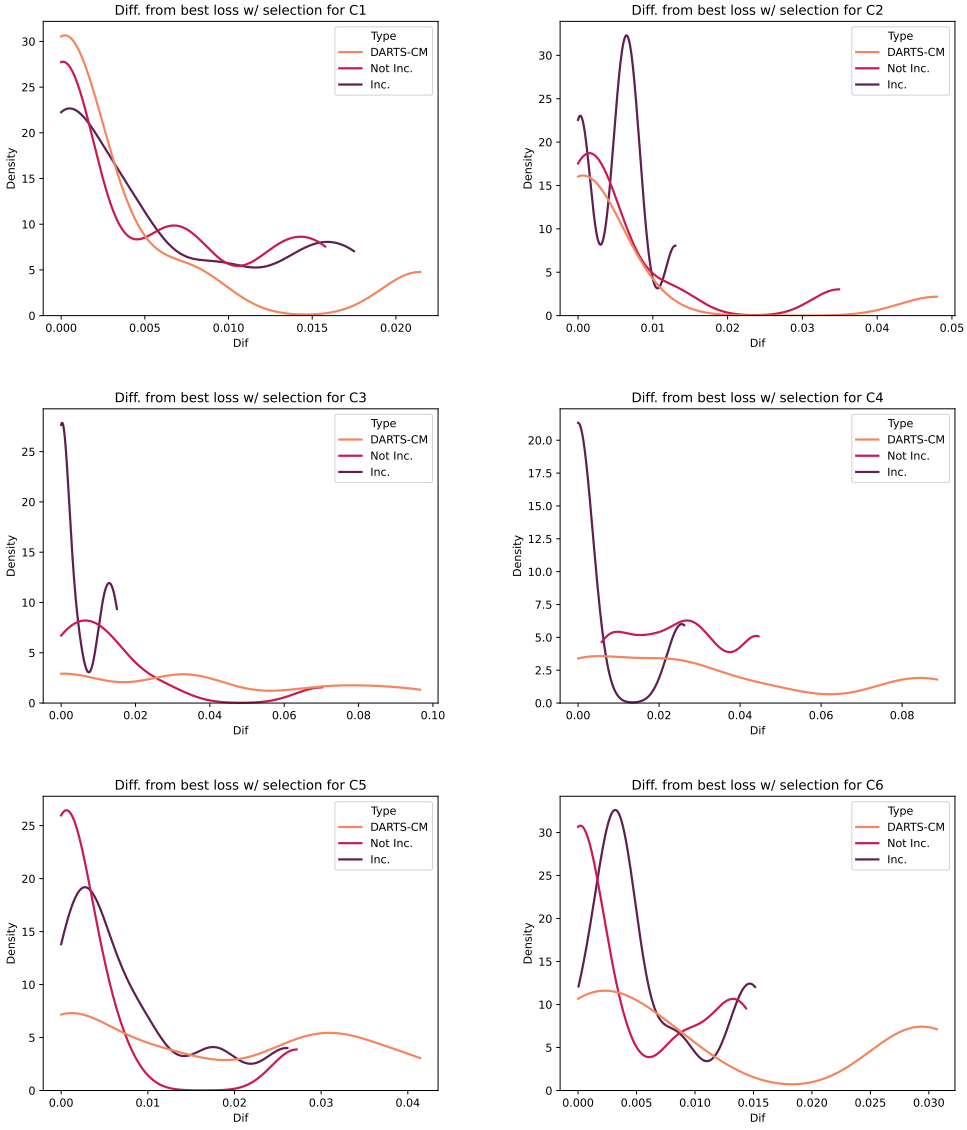


Figure 6.4: The distribution of the lowest loss values obtained by the best performing architectures, grouped by label type and category.

6.3 Comparison between concept types

One of the things we wanted to assess with the empirical tests was the degree to which it is possible to predict the optimal architecture from the concept being mapped. Even if no one architecture is optimal for all concepts, there might be one for each type of concept. Thus, for each main class, we selected a concept from each of three types: high-level, low-level trivial, and low-level non-trivial. The meaning of each type is described in detail in Section 5.6, but here is a reminder: high-level concepts describe combinations of lower-level ones, like the high-level concept *Hotel* is defined by the lower-level one $\exists has.WallSign$. The trivial category describes low-level concepts that, if true, are enough to infer that a main class is also True, like $\exists has.WallSign$, if true, means *Hotel*, which means *Commercial*. Non-trivial, in contrast, are concepts that, even if true, are not enough to infer that any main class is true. This is not the only way one could subdivide the concepts, but it represents precisely the sort of intuitive categorization that a possible user might draw.

Figure 6.4 shows, for each complexity, the loss distribution per category and label type. It is hard to discern any patterns in these results. The *1 Neuron* category, for example, seems to perform better on non-trivial concepts for complexities one and two. However, in complexity three, non-trivial concepts are where the same architecture gets the worst results. DARTS-CM shows the smallest variance on high-level concepts for complexity one, but high-levels have the highest variance in complexity two. All categories show higher or lower-bounds for high-level concepts in complexities one to four, but in five and six, the opposite is true. These results show that it is not feasible to predict what type of architecture will be best for a given concept, further emphasizing the advantage of automated search methods such as DARTS-CM.

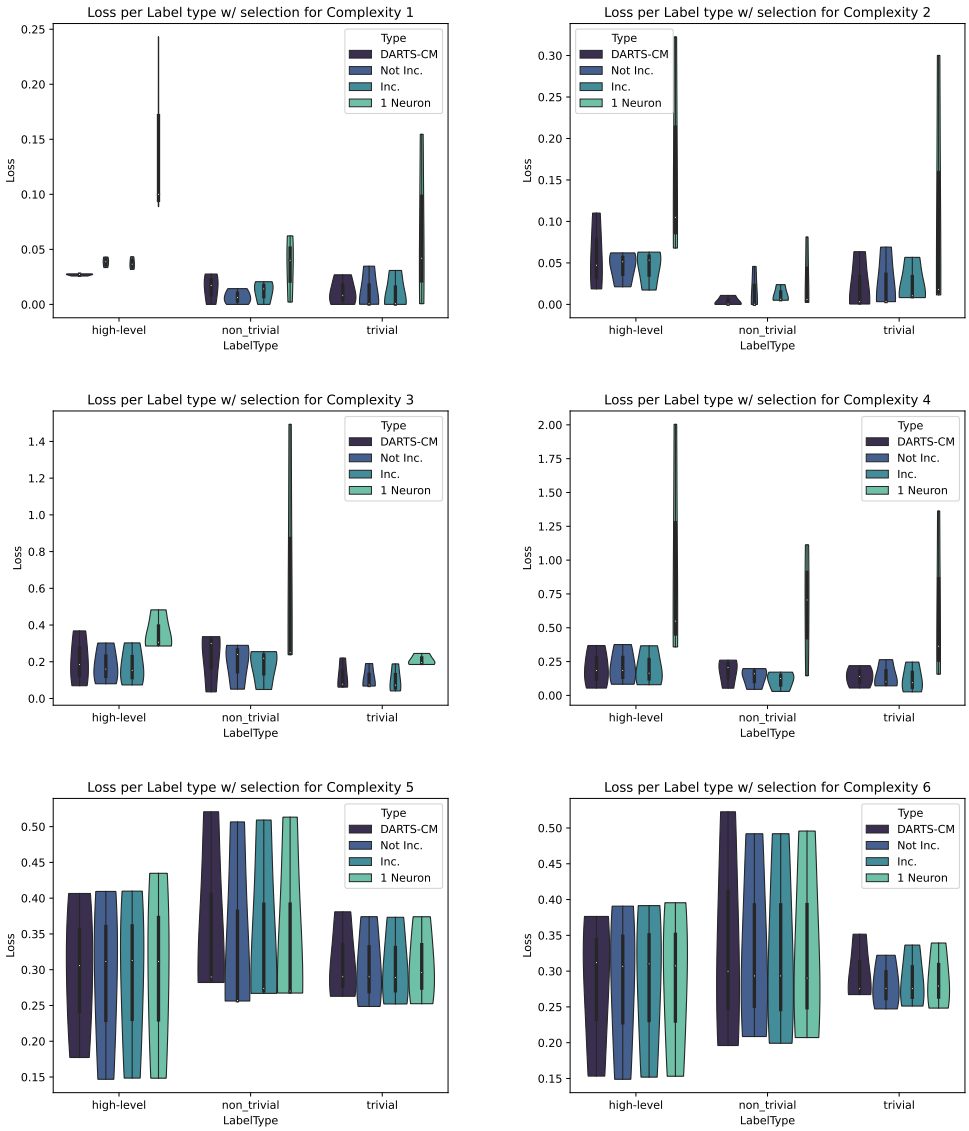


Figure 6.5: The distribution of the lowest loss values obtained by the best performing architectures, grouped by category.

6.4 The learned architectures

While we have placed more focus on the performance of the architectures learned by DARTS-CM, it is also worthwhile to take a brief look at their particular composition. Because the algorithm learns a four-node computational cell, the resulting architecture encoding consists of a sequence of four neural operations, belonging to the list of candidate operations used. Figures 6.6, 6.7, and 6.8 contain graphical representations of the learned cell for particular configurations. The first node, labeled "C(k-1)", denotes the previous cell's output, and the last node, labeled "C(k)", denotes the current cell's output.

The most commonly chosen operation overall is the layer normalization operation, seemingly used by DARTS-CM as a default in instances where parametric operations are

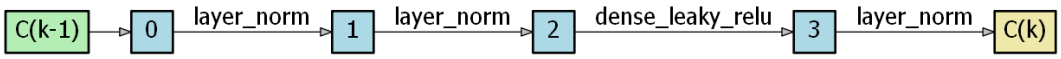


Figure 6.6: Learned cell for complexity 1, main class Residential, concept MiscResidential

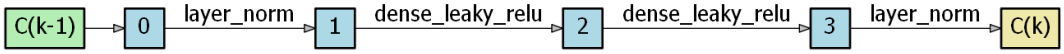


Figure 6.7: Learned cell for complexity 2, main class Industrial, concept MiscIndustrial

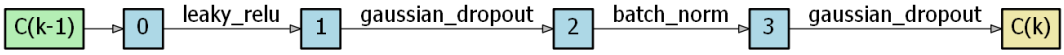


Figure 6.8: Learned cell for complexity 4, main class Industrial, concept MiscIndustrial

not required. However, the use of this operation is more common on low complexities, whereas the higher ones see an increased use of the dense operation with non-linear activation functions. This indicates that DARTS-CM reacted to the increased difficulty of concept mapping for higher complexities. Nonetheless, operations without trainable parameters form the majority of selected operations for all complexities. Dropout was the least used class of operations, and only the Gaussian Dropout alternative was selected.

6.5 Discussion

DARTS-CM showed an especially large advantage over the manual models for complexity one. After selection, DARTS-CM's advantage slowly diminishes for each progressive complexity as it becomes more similar to the other categories. At first glance, this may seem unintuitive. If DARTS-CM can create custom-tailored architectures that beat the pre-set ones for one dataset, why can it not do it for the others to the same degree? We believe the answer is that DARTS-CM requires further optimization to find the optimal architecture unfailingly. We saw that the loss increased with the complexity across the board, meaning that concepts from more complex classification tasks are also more complex to map. However, DARTS-CM was trained on a maximum of fifty epochs with a relatively strict early stopping policy with a patience of twenty epochs. DARTS-CM optimizes the arch weights at the same time as the model weights, meaning that operations with

fewer parameters are favored in the beginning. That has the advantage that DARTS-CM does not tend to over-complicate; if a simple architecture is good enough, it is likely to choose it over more complex, even if similarly performing, ones. However, this initial bias towards operations with fewer parameters also means that optimal operations with high parameter counts require more time to surface. The fact that the majority of the candidate operations used were non-parametric further reduces the attention received by the others, reducing the probability that they are 'noticed' by the algorithm. For low complexities - and better trained main networks - DARTS-CM manages to fully optimize the architecture, distinguishing itself from the pre-set architectures, but at high ones, it is not able to learn the optimal architecture fast enough. However, this was the first time Neural Architecture Search was applied to the concept mapping problem. Optimizing the search space further would likely enable DARTS-CM to more reliably get an advantage over trial-and-error. We have learned from these results that, while blindly using the same architecture for every concept leads to sub-optimal results, testing a set of ten architectures and picking the best one out of the bunch is a viable solution. This means that the search space of a Concept Mapping NAS system need not be that large. Our results indicate that optimizing DARTS-CM's search space to become faster at learning the optimal cell while retaining the ability to emulate the individual testing of thousands of architectures is a highly promising route.

As a matter of fact, we tested this intuition by running DARTS-CM for a maximum of two hundred epochs and with an early stopping policy with a patience of thirty epochs. The concept $\exists has.Porch$ was mapped from the main network for the complexity four dataset and main class *Residential*. Additionally, we shortened the search space by leaving only the skip_connect, dropout, batch normalization, and the dense (with and without an activation function) operations. Previously DARTS-CM had achieved a test-set loss value of 0.0546 for this same configuration, being surpassed by a manual architecture that achieved a loss value of 0.0262. After these changes, DARTS-CM managed to find an architecture that obtained a loss value of 0.0243, not only obtaining a better result than before but a better one than any of the manual architectures. Unfortunately, time did not allow for the whole test suite to be re-run with the new implementation.

Let us evaluate, though, the different options as they exist now. If trying to perform concept mapping on a well-trained main network with a task of low-to-medium complexity, DARTS-CM is an obvious choice. In that scenario, it can reliably achieve better results than trial-and-error. Interestingly, that is also the scenario in which we find concept mapping as a whole to be most applicable. In a scenario where the main network has been trained on a more complex task to a satisfactory degree, the question becomes about priorities. In this scenario, DARTS-CM's advantage over trial-and-error reduces, reaching mostly equivalent but, at times, inferior results. If the one-and-only consideration is performance, then manual trial-and-error will likely beat DARTS-CM as it currently stands, given enough tries. The question then becomes, though, what set of architectures to try. As we saw in the charts with no selection, trial-and-error performance is unpredictable

and has a high variance. In this scenario, deciding on the architectures to use would fall on a human expert, which takes time and other resources. As a matter of fact, how much time is needed is also somewhat unpredictable. If using a set of pre-defined architectures, the size of that set would dictate the time that the trial-and-error process takes. If one uses too large a set, the time to explore it could become much larger than the time DARTS-CM takes to explore countless more options. If one does not use a pre-defined set, then there is no indication of how long it will be before an architecture considered good enough is found. In contrast, DARTS-CM takes a more predictable and even amount of time, given that the same amount of examples are used. Further, the work done by that expert would only be usable for the specific task: as we saw, concept mapping is sensitive to the main task, and the set of architectures needed to arrive at good results would also likely change. With its ample search space, DARTS-CM is more resilient to those types of changes. This is to say that DARTS-CM is a more resource-efficient method, despite the relatively large initial investment of implementing it. The time and resources saved by using automated search scale with the number of concepts being mapped and the number of different models being interpreted. If other than performance, efficiency is also a consideration, DARTS-CM presents a way to get above-average concept mappers for a large number of applications, with little human intervention.

Let us here summarize the main insights gained in the course of this dissertation. Firstly, we find that the one-size-fits-all solution offered by the 1-neuron model is categorically insufficient. There is no reliable way to predict whether a given concept is a linear function of the inputs. Using non-linear models with regularization techniques offers better results in the vast majority of the cases. When it comes to the other candidates, while results show that the concept mapping task is not altogether random (it is possible to arrive at good results through manual experimentation of a set of candidate architectures), we find our hypothesis confirmed in the fact that it is extremely sensitive to the main task and the degree to which the main network has learned it. This fact brings into question how applicable even a fixed set of architectures is to other models one may want to interpret. Indeed, we saw the performance of the manual architectures vary significantly from dataset to dataset and main network to main network, even in our relatively controlled setting. In contrast, we find that DARTS-CM is able to use its larger search space efficiently to produce consistently good, and in fact, the best results more often than any of the other categories.

7

Conclusion

In this dissertation, we empirically demonstrate the merits of employing neural architecture search in the design of concept mapping architectures. We conduct extensive empirical tests, training thousands of models to analyze how different axes - the main task's complexity, the main network's quality, the concept, and the type of architecture - affect the concept mapping task. We show that our proposed system, DARTS-CM, can reliably obtain comparable to better results than trial-and-error while requiring less human intervention and remaining more resilient to context changes. We also find that DARTS-CM offers the biggest advantage for scenarios where concept mapping is most applicable. This represents a significant step in making concept mapping applicable in real use cases at a low cost to the user. To further support this point, the isolated test we conducted - with a more prolonged architecture search and a reduced search space - shows that DARTS-CM can be significantly improved with minor changes to the methodology. Adding to that the fact that in higher-complexity datasets, only the architectures not included in DARTS-CM's search space - i.e., more complex than any DARTS-CM could learn - manage to out-perform it, it is clear that the idea of using DARTS-CM is the right one, and with slight modifications to its search-space, it can even further distinguish itself as the optimal way to find exemplary concept mapping architectures.

As another contribution of this dissertation, we deliver the VCB dataset: a set of image classification datasets for the same task, but that differ in how difficult they are to classify. As opposed to simply adding noise to an existing dataset, different VCB complexities change in fundamental ways (such as whether the subjects are always seen from the same angle), which fundamentally changes the classification task. Furthermore, the generator for the VCB dataset is ontology-independent, allowing the user to customize the underlying logic of the images and their labels. The VCB dataset is unique in its versatility and it is a powerful tool for research and pedagogic environments alike.

7.1 Future work

This work done in this dissertation is a first step in introducing neural architecture search into a potential concept-mapping pipeline. As we successfully demonstrate the merits of that idea, avenues for further improvement come to mind.

7.1.1 Optimizing DARTS-CM's search-space

With the current implementation of DARTS-CM, it is straightforward to change the set of candidate operations, as well as the basic structure of the cell. So, work can be done on optimizing the search space of DARTS-CM to get faster convergence while remaining flexible enough to adapt to different use cases. DARTS shows an early bias for operations with fewer trainable parameters, so introducing a majority of those types of operations into the search space can be dangerous. One possible solution is to have composite candidate operations, with multiple sub-operations each. For example, one candidate operation could be a batch normalization layer followed by a dense layer followed by a ReLU activation function. By keeping the skip-connect operation, the search algorithm would remain able to find simple designs, but the operations with high parameter counts would be awarded more attention by the algorithm.

Another route would be to experiment with different cell structures. In this work, we used a linear cell structure with four nodes (Fig. 5.1). Experimentation can be done by increasing the number of nodes, or making the cell non-linear, with each node receiving multiple inputs from preceding structures. In the discretization phase, one could experiment with keeping differing amounts of edges per node.

7.1.2 Optimizing the concept mapper's input

One variable that remained static in our experiments was the set of activations from the main network that were fed to the concept mappers. Experiments with extracting different sets of the main network's activation could yield interesting results. Different types of concepts can likely be best extracted at different stages of the network, so optimizing the inputs could increase the upper bound on mapping performance. Sousa Ribeiro and Leite proposed the *InputReduce* procedure, which already does this optimization, but it is prohibitively expensive to run it for multiple concepts. Instead, more general guidelines may exist to get better performing and smaller mapping models. In our experiments, we fed the mapping networks the output of the flatten layer and all subsequent batch normalization layers. The batch normalization layers, however, simply contain different representations of the flatten layer's output, which inherently makes the input data highly correlated, something that is not generally desirable. Having established that, the output of the batch normalization layers does benefit from the learning performed by the main network's dense layers. Using the feature maps of the convolutional layers may also prove

helpful to map lower-level concepts that are identified early in the network. Regarding this, we believe a few questions are worthy of investigation:

- Is it possible to map every concept at the highest possible performance from the flatten layer alone?
- If not, is there a benefit in using the feature maps of the convolutional layers?
- What is the best way to mitigate the downsides of highly correlated data?
- Is it possible to accurately predict the optimal input to map a concept, given that concept's role in the ontology?

7.1.3 Optimizing the mapping network's labels

As we previously touched upon, training concept mapping models with a supervised learning approach is only possible using likely labels. We do not have access to the ground truth about what the black-box model did or did not identify in an input. The target labels given to the concept mapper represent whether the concept is in the input, not if it was identified. One simple thing that could help in this regard would be to only feed the mapping networks the activations from the inputs that the main network classified correctly. That way, there is a higher chance that the main network has indeed extracted the relevant concept, which would increase the likelihood that the target label given to the mapping network matches reality. While this may increase the performance of the mapping models for those types of examples, the mappings for an example that the main network misclassified could become even less reliable.

Another option could entail using an unsupervised learning approach. A protocol can be imagined that starts by learning a clustering of the activations, where presumably the activations corresponding to the same concepts would be in the same cluster (perhaps crops of the original image could be used to isolate different concepts better). Then, the original images that produced the activations could be shown to a user who is asked to label each cluster with the common concept among the pictures in that cluster. At this point, the user could build an ontology, using those concepts, that describes the domain of the task. This way, the user would not have to guess what concepts are relevant to the main network. Possibly, association rule learning [20] could be used to suggest likely ontology axioms to the user. After the writing of the ontology, the clustering could be used for concept mapping, by seeing what cluster a new example is placed in. This is only an example and would require significant work to assess its feasibility. Nevertheless, if possible, this would provide two key advantages: a more informed choice of concepts to use in the ontology and a removal of the need to manually label examples with secondary labels.

Bibliography

[1] O. I. Abiodun et al. "State-of-the-art in artificial neural network applications: A survey". In: *Heliyon* 4.11 (2018), e00938. ISSN: 2405-8440. DOI: https://doi.org/10.1016/j.heliyon.2018.e00938. URL: https://www.sciencedirect.com/science/article/pii/S2405844018332067 (cit. on p. 10).

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. *Layer Normalization*. 2016. DOI: 10.48550/ARXIV.1607.06450. URL: https://arxiv.org/abs/1607.06450 (cit. on p. 54).

[3] F. Baader, ed. *The description logic handbook: theory, implementation, and applications*. Cambridge, UK ; New York: Cambridge University Press, 2003. ISBN: 9780521781763 (cit. on p. 18).

[4] S. Bach et al. "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation". en. In: *PLOS ONE* 10.7 (Oct. 2015), e0130140. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0130140. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0130140 (visited on 02/15/2022) (cit. on p. 23).

[5] B. Baker et al. "Designing Neural Network Architectures using Reinforcement Learning". In: (2016). URL: https://arxiv.org/abs/1611.02167 (cit. on p. 30).

[6] H. Cai et al. "Efficient Architecture Search by Network Transformation". In: *arXiv:1707.04873 [cs]* (Nov. 2017). arXiv: 1707.04873. URL: http://arxiv.org/abs/1707.04873 (visited on 01/13/2022) (cit. on p. 31).

[7] G. Chen et al. *Rethinking the Usage of Batch Normalization and Dropout in the Training of Deep Neural Networks*. arXiv:1905.05928 [cs, stat]. May 2019. URL: http://arxiv.org/abs/1905.05928 (visited on 09/10/2022) (cit. on p. 63).

[8] T. Chen, I. Goodfellow, and J. Shlens. "Net2Net: Accelerating Learning via Knowledge Transfer". In: *arXiv:1511.05641 [cs]* (Apr. 2016). arXiv: 1511.05641. URL: http://arxiv.org/abs/1511.05641 (visited on 01/13/2022) (cit. on p. 31).

[9] *Common ML Problems | Introduction to Machine Learning Problem Framing*. URL: https://developers.google.com/machine-learning/problem-framing/cases (cit. on p. 8).

[10] *Convolutional Neural Network Architecture CNN Architecture*. en. Oct. 2020. URL: https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/ (visited on 02/09/2022) (cit. on pp. 16, 17).

[11] P. Covington, J. Adams, and E. Sargin. "Deep Neural Networks for YouTube Recommendations". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys '16. Boston, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 191–198. ISBN: 9781450340359. DOI: 10.1145/2959100.2959190. URL: https://doi.org/10.1145/2959100.2959190 (cit. on p. 1).

[12] J. Dastin. *Amazon scraps secret AI recruiting tool that showed bias against women*. Oct. 2018. URL: https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G (cit. on p. 1).

[13] Deathn0t. *DEATHN0T/darts-tensorflow: Tensorflow code for differentiable architecture search*. URL: https://github.com/Deathn0t/darts-tensorflow (cit. on p. 56).

[14] D. Doran, S. Schulz, and T. R. Besold. "What Does Explainable AI Really Mean? A New Conceptualization of Perspectives". In: *arXiv:1710.00794 [cs]* (Oct. 2017). arXiv: 1710.00794. URL: http://arxiv.org/abs/1710.00794 (visited on 02/10/2022) (cit. on p. 2).

[15] I. C. Education. *What is Artificial Intelligence (AI)?* URL: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence (cit. on p. 8).

[16] *FaCT++ reasoner | OWL research at the University of Manchester*. en-US. URL: http://owl.cs.manchester.ac.uk/tools/fact/ (visited on 02/17/2022) (cit. on p. 22).

[17] A. Fadlalla and C.-H. Lin. "An Analysis of the Applications of Neural Networks in Finance". In: *Interfaces* 31.4 (2001), pp. 112–122. DOI: 10.1287/inte.31.4.112.9662 (cit. on p. 1).

[18] B. Foundation. *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. en. URL: https://www.blender.org/ (visited on 09/10/2022) (cit. on pp. 6, 50).

[19] K. Fukushima. "Cognitron: A self-organizing multilayered neural network". en. In: *Biological Cybernetics* 20.3-4 (1975), pp. 121–136. ISSN: 0340-1200, 1432-0770. DOI: 10.1007/BF00342633. URL: http://link.springer.com/10.1007/BF00342633 (visited on 09/09/2022) (cit. on p. 14).

[20] J. Fürnkranz and T. Kliegr. "A Brief Overview of Rule Learning". In: *Rule Technologies: Foundations, Tools, and Applications*. Ed. by N. Bassiliades et al. Cham: Springer International Publishing, 2015, pp. 54–69. ISBN: 978-3-319-21542-6 (cit. on p. 79).

[21] N. Gillespie, S. Lockey, and C. Curtis. "Trust in artificial Intelligence: a five country study". In: (2021). DOI: 10.14264/e34bfa3 (cit. on p. 2).

[22] C. Guo et al. "On Calibration of Modern Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, June 2017, pp. 1321–1330. URL: https://proceedings.mlr.press/v70/guo17a.html (cit. on p. 27).

[23] K. He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. ISBN: 9781467388511. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459/ (visited on 09/06/2022) (cit. on pp. 31, 55).

[24] *HermiT Reasoner: Home*. URL: http://www.hermit-reasoner.com/ (visited on 02/17/2022) (cit. on p. 22).

[25] F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". en. In: *Learning and Intelligent Optimization*. Ed. by C. A. C. Coello. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 507–523. ISBN: 9783642255663. DOI: 10.1007/978-3-642-25566-3_40 (cit. on pp. 30, 32).

[26] S. Ioffe and C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv:1502.03167 [cs]. Mar. 2015. URL: http://arxiv.org/abs/1502.03167 (visited on 08/25/2022) (cit. on p. 54).

[27] K. Kandasamy et al. "Neural Architecture Search with Bayesian Optimisation and Optimal Transport". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/f33ba15effa5c10e873bf3842afb46a6-Paper.pdf (cit. on p. 30).

[28] A. Karpathy et al. "Large-scale Video Classification with Convolutional Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014 (cit. on p. 1).

[29] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980 [cs]. Jan. 2017. DOI: 10.48550/arXiv.1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 09/09/2022) (cit. on pp. 57, 62, 63).

[30] J.-B. Lamy et al. "Explainable artificial intelligence for breast cancer: A visual case-based reasoning approach". In: *Artificial Intelligence in Medicine* 94 (2019), pp. 42–53. ISSN: 0933-3657. DOI: https://doi.org/10.1016/j.artmed.2019.01.001. URL: https://www.sciencedirect.com/science/article/pii/S0933365718304846 (cit. on p. 2).

[31] C. Li et al. *Enhancing the Robustness, Efficiency, and Diversity of Differentiable Architecture Search*. arXiv:2204.04681 [cs]. Apr. 2022. URL: http://arxiv.org/abs/2204.04681 (visited on 09/08/2022) (cit. on p. 55).

[32] *Linear separability (Image)*. https://en.wikipedia.org/wiki/Linear_separability. Sept. 2021 (cit. on p. 12).

[33] H. Liu, K. Simonyan, and Y. Yang. "DARTS: Differentiable Architecture Search". In: *arXiv:1806.09055 [cs, stat]* (June 2018). arXiv: 1806.09055 version: 1. URL: http://arxiv.org/abs/1806.09055 (visited on 01/13/2022) (cit. on pp. vii, viii, 31, 35, 38, 53, 55).

[34] V. Nair and G. E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *ICML*. 2010, pp. 807–814. URL: https://icml.cc/Conferences/2010/papers/432.pdf (cit. on pp. 14, 54).

[35] R. Negrinho and G. Gordon. *DeepArchitect: Automatically Designing and Training Deep Architectures*. 2017. DOI: 10.48550/ARXIV.1704.08792. URL: https://arxiv.org/abs/1704.08792 (cit. on p. 30).

[36] *OWL 2 Web Ontology Language Primer (Second Edition)*. URL: https://www.w3.org/TR/2012/REC-owl2-primer-20121211/ (visited on 02/17/2022) (cit. on p. 22).

[37] *OWL Web Ontology Language Overview*. URL: https://www.w3.org/TR/owl-features/ (visited on 02/16/2022) (cit. on p. 22).

[38] U. Pawar et al. "Explainable AI in Healthcare". In: *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*. 2020, pp. 1–2. DOI: 10.1109/CyberSA49311.2020.9139655 (cit. on p. 2).

[39] *Pellet - Semantic Web Standards*. URL: https://www.w3.org/2001/sw/wiki/Pellet (visited on 02/17/2022) (cit. on p. 22).

[40] V. Petsiuk, A. Das, and K. Saenko. *RISE: Randomized Input Sampling for Explanation of Black-box Models*. arXiv:1806.07421 [cs]. Sept. 2018. URL: http://arxiv.org/abs/1806.07421 (visited on 09/04/2022) (cit. on pp. 2, 23, 24).

[41] H. Pham et al. "Efficient Neural Architecture Search via Parameter Sharing". In: *arXiv:1802.03268 [cs, stat]* (Feb. 2018). arXiv: 1802.03268 version: 1. URL: http://arxiv.org/abs/1802.03268 (visited on 01/13/2022) (cit. on pp. 32, 33).

[42] quark0. *QUARK0/darts: Differentiable architecture search for convolutional and recurrent networks*. URL: https://github.com/quark0/darts (cit. on p. 55).

[43] *RACER - Home*. URL: https://www.ifis.uni-luebeck.de/~moeller/racer/ (visited on 02/17/2022) (cit. on p. 22).

[44] E. Real et al. *Large-scale evolution of image classifiers*. June 2017. URL: https://arxiv.org/abs/1703.01041 (cit. on p. 30).

[45] P. Ren et al. "A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions". In: *arXiv.org* (2021). URL: https://arxiv.org/abs/2006.02903v3 (cit. on pp. 30–33).

[46] M. Rey and A. Mnih. "Gaussian dropout as an information bottleneck layer". In: () (cit. on p. 54).

[47] M. T. Ribeiro, S. Singh, and C. Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. New York, NY, USA: Association for Computing Machinery, Aug. 2016, pp. 1135–1144. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778. URL: https://doi.org/10.1145/2939672.2939778 (visited on 02/16/2022) (cit. on pp. 2, 23).

[48] S. Richard, P. Charles, and S. Dawn. "DIFFERENTIABLE NEURAL NETWORK ARCHITECTURE SEARCH". In: () (cit. on p. 31).

[49] F. Riguzzi et al. "BUNDLE: A Reasoner for Probabilistic Ontologies". en. In: *Web Reasoning and Rule Systems*. Ed. by W. Faber and D. Lembo. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 183–197. ISBN: 9783642396663. DOI: 10.1007/978-3-642-39666-3_14 (cit. on p. 26).

[50] F. Riguzzi et al. "Reasoning with probabilistic ontologies". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015 (cit. on pp. 22, 26).

[51] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". en. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0 (visited on 09/12/2022) (cit. on p. 14).

[52] D. G. S. Santiago. "Handbook of Knowledge Representation". In: (). URL: https://www.academia.edu/17590697/Handbook_of_Knowledge_Representation (visited on 09/05/2022) (cit. on p. 19).

[53] R. Sato, J. Sakuma, and Y. Akimoto. "AdvantageNAS: Efficient Neural Architecture Search with Credit Assignment". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.11 (May 2021), pp. 9489–9496. URL: https://ojs.aaai.org/index.php/AAAI/article/view/17143 (cit. on p. 55).

[54] M. Schmidt-Schauß and G. Smolka. "Attributive concept descriptions with complements". en. In: *Artificial Intelligence* 48.1 (Feb. 1991), pp. 1–26. ISSN: 00043702. DOI: 10.1016/0004-3702(91)90078-X. URL: https://linkinghub.elsevier.com/retrieve/pii/000437029190078X (visited on 09/05/2022) (cit. on p. 19).

[55] C. Sciuto et al. "Evaluating the Search Phase of Neural Architecture Search". In: *CoRR* abs/1902.08142 (2019). arXiv: 1902.08142. URL: http://arxiv.org/abs/1902.08142 (cit. on p. 30).

[56] Y. Shen et al. "Learning Semantic Representations Using Convolutional Neural Networks for Web Search". In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW '14 Companion. Seoul, Korea: Association for Computing Machinery, 2014, pp. 373–374. ISBN: 9781450327459. DOI: 10.1145/2567948.2577348. URL: https://doi.org/10.1145/2567948.2577348 (cit. on p. 1).

[57] A. Shrikumar et al. "Not Just a Black Box: Learning Important Features Through Propagating Activation Differences". In: *arXiv:1605.01713 [cs]* (Apr. 2017). arXiv: 1605.01713. URL: http://arxiv.org/abs/1605.01713 (visited on 02/15/2022) (cit. on p. 23).

[58] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv:1409.1556 [cs]. Apr. 2015. DOI: 10.48550/arXiv.1409.1556. URL: http://arxiv.org/abs/1409.1556 (visited on 09/06/2022) (cit. on p. 31).

[59] E. Sirin et al. "Pellet: A practical OWL-DL reasoner". In: *Journal of Web Semantics* 5.2 (2007). Software Engineering and the Semantic Web, pp. 51–53. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2007.03.004. URL: https://www.sciencedirect.com/science/article/pii/S1570826807000169 (cit. on p. 26).

[60] M. de Sousa Ribeiro and J. Leite. "Aligning Artificial Neural Networks and Ontologies towards Explainable AI". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.6 (May 2021), pp. 4932–4940. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16626 (cit. on pp. vii, viii, 2–5, 25, 26, 29, 53, 55, 57–60, 62, 78).

[61] N. Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. ISSN: 1533-7928. URL: http://jmlr.org/papers/v15/srivastava14a.html (visited on 08/25/2022) (cit. on p. 54).

[62] F. Sultana, A. Sufian, and P. Dutta. "Advancements in Image Classification using Convolutional Neural Network". In: *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*. 2018, pp. 122–129. DOI: 10.1109/ICRCICN.2018.8718718 (cit. on p. 1).

[63] C. Szegedy et al. "Going Deeper With Convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015 (cit. on p. 55).

[64] R. L. Teach and E. H. Shortliffe. "An analysis of physician attitudes regarding computer-based clinical consultation systems". In: *Computers and Biomedical Research* 14.6 (1981), pp. 542–558. ISSN: 0010-4809. DOI: https://doi.org/10.1016/0010-4809(81)90012-4. URL: https://www.sciencedirect.com/science/article/pii/0010480981900124 (cit. on p. 2).

[65] T. Terasvirta and H. M. Anderson. "Characterizing nonlinearities in business cycles using smooth transition autoregressive models". en. In: *Journal of Applied Econometrics* 7.S1 (Dec. 1992), S119–S136. ISSN: 08837252, 10991255. DOI: 10.1002/jae.3950070509. URL: https://onlinelibrary.wiley.com/doi/10.1002/jae.3950070509 (visited on 09/10/2022) (cit. on p. 54).

[66] C.-F. Tsai. "Financial decision support using neural networks and support vector machines". In: *Expert Systems* 25.4 (2008), pp. 380–393. DOI: 10.1111/j.1468-0394.2008.00449.x (cit. on p. 1).

[67] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697 (cit. on p. 50).

[68] S. Walczak. "Categorizing university student applicants with neural networks". In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. Vol. 6. 1994, 3680–3685 vol.6. DOI: 10.1109/ICNN.1994.374796 (cit. on p. 1).

[69] S.-C. Wang. "Artificial Neural Network". In: *Interdisciplinary Computing in Java Programming* (2003), pp. 81–100. DOI: 10.1007/978-1-4615-0377-4_5 (cit. on p. 9).

[70] X. Wang et al. *ZARTS: On Zero-order Optimization for Neural Architecture Search*. arXiv:2110.04743 [cs]. Jan. 2022. URL: http://arxiv.org/abs/2110.04743 (visited on 09/08/2022) (cit. on p. 55).

[71] *What is a Neural Network? (Image)*. https://www.tibco.com/reference-center/what-is-a-neural-network (cit. on p. 13).

[72] B. Wu et al. "SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. July 2017 (cit. on p. 1).

[73] R. Yan, Y. Song, and H. Wu. "Learning to Respond with Deep Neural Networks for Retrieval-Based Human-Computer Conversation System". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '16. Pisa, Italy: Association for Computing Machinery, 2016, pp. 55–64. ISBN: 9781450340694. DOI: 10.1145/2911451.2911542. URL: https://doi.org/10.1145/2911451.2911542 (cit. on p. 1).

[74] M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks". en. In: *Computer Vision – ECCV 2014*. Ed. by D. Fleet et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 818–833. ISBN: 9783319105901. DOI: 10.1007/978-3-319-10590-1_53 (cit. on p. 23).

[75] X. Zheng et al. "Multinomial Distribution Learning for Effective Neural Architecture Search". In: *arXiv:1905.07529 [cs]* (Aug. 2019). arXiv: 1905.07529. URL: http://arxiv.org/abs/1905.07529 (visited on 01/13/2022) (cit. on p. 32).

[76] B. Zoph and Q. V. Le. "Neural Architecture Search with Reinforcement Learning". In: (2017). URL: https://arxiv.org/abs/1611.01578 (cit. on p. 30).

A

Complete experimental results

This appendix contains the raw results of the conducted experiments in table form. The 'Architecture' column contains information about the architecture in question. DARTS_nLayers is the architecture learned by DARTS-CM with the learned cell stacked n times. The remaining pertain to the list of manual architectures defined in Section 5.6.2:

Code	Architecture
1_NEURON	Dense(1, sigmoid)
BN	BN - Dense(1, sigmoid)
BN_DO	BN - DO - Dense(1, sigmoid)
DO	DO - Dense(1, sigmoid)
RELU	Dense(128, ReLU) - Dense(1, sigmoid)
RELU_BN	BN - Dense(128, ReLU) - BN - Dense(1, sigmoid)
RELU_BN_DO	Dense(128, ReLU) - BN - DO - Dense(1, sigmoid)
RELU_DO	DO - Dense(128, ReLU) - DO - Dense(1, sigmoid)
NOIC_RELU_BN_DO	Dense(128, ReLU) - BN - DO - Dense(1, sigmoid)

The 'C' column refers to the complexity level of the dataset, 'Class' to the class that the main network in question was trained to extract, 'Concept' to the concept being mapped and 'Loss' and 'Acc' to the test-set metrics obtained by each row's concept mapper.

Architecture	C	Class	Concept	Loss	Acc
DARTS_1Layers	1	Commercial	Billboard	0.034635	0.98725
DARTS_2Layers	1	Commercial	Billboard	0.043129	0.98475
DARTS_3Layers	1	Commercial	Billboard	0.026884	0.98925
DARTS_4Layers	1	Commercial	Billboard	0.048303	0.9875
1_NEURON	1	Commercial	Billboard	0.154525	0.983125
BN	1	Commercial	Billboard	0.154727	0.983
BN_DO	1	Commercial	Billboard	0.122577	0.984375
DO	1	Commercial	Billboard	0.137334	0.987125
RELU	1	Commercial	Billboard	0.332267	0.988375
RELU_BN	1	Commercial	Billboard	0.037446	0.986125
RELU_BN_DO	1	Commercial	Billboard	0.036438	0.98725

RELU_DO	1	Commercial	Billboard	0.034715	0.9885
NOIC_RELU_BN_DO	1	Commercial	Billboard	0.030828	0.99
DARTS_1Layers	1	Commercial	Restaurant	0.027216	0.98475
DARTS_2Layers	1	Commercial	Restaurant	0.041541	0.9885
DARTS_3Layers	1	Commercial	Restaurant	0.03295	0.98625
DARTS_4Layers	1	Commercial	Restaurant	0.045162	0.98125
1_NEURON	1	Commercial	Restaurant	0.243018	0.975125
BN	1	Commercial	Restaurant	0.152522	0.98425
BN_DO	1	Commercial	Restaurant	0.178136	0.979125
DO	1	Commercial	Restaurant	0.215559	0.976375
RELU	1	Commercial	Restaurant	0.681733	0.783875
RELU_BN	1	Commercial	Restaurant	0.042987	0.9835
RELU_BN_DO	1	Commercial	Restaurant	0.047541	0.983375
RELU_DO	1	Commercial	Restaurant	0.09631	0.974875
NOIC_RELU_BN_DO	1	Commercial	Restaurant	0.036569	0.98575
DARTS_1Layers	1	Commercial	Statue	5.21E-19	1
DARTS_2Layers	1	Commercial	Statue	0	1
DARTS_3Layers	1	Commercial	Statue	5.75E-21	1
DARTS_4Layers	1	Commercial	Statue	6.55E-24	1
1_NEURON	1	Commercial	Statue	0.002142	0.9995
BN	1	Commercial	Statue	2.75E-12	1
BN_DO	1	Commercial	Statue	0.000663	0.999875
DO	1	Commercial	Statue	0.000859	0.999625
RELU	1	Commercial	Statue	0.004729	0.999625
RELU_BN	1	Commercial	Statue	9.16E-06	1
RELU_BN_DO	1	Commercial	Statue	3.47E-06	1
RELU_DO	1	Commercial	Statue	1.82E-05	1
NOIC_RELU_BN_DO	1	Commercial	Statue	4.59E-06	1
DARTS_1Layers	1	Industrial	Chimney	0.034139	0.9905
DARTS_2Layers	1	Industrial	Chimney	0.027576	0.99025
DARTS_3Layers	1	Industrial	Chimney	0.031749	0.9915
DARTS_4Layers	1	Industrial	Chimney	0.029468	0.991
1_NEURON	1	Industrial	Chimney	0.04011	0.994125
BN	1	Industrial	Chimney	0.067096	0.989
BN_DO	1	Industrial	Chimney	0.053648	0.992125
DO	1	Industrial	Chimney	0.056469	0.990875
RELU	1	Industrial	Chimney	0.192321	0.89625
RELU_BN	1	Industrial	Chimney	0.006131	0.998375
RELU_BN_DO	1	Industrial	Chimney	0.007693	0.997875
RELU_DO	1	Industrial	Chimney	0.019619	0.9945

NOIC_RELU_BN_DO	1	Industrial	Chimney	0.020608	0.992875
DARTS_1Layers	1	Industrial	Machine	0.015723	0.997
DARTS_2Layers	1	Industrial	Machine	0.011913	0.996
DARTS_3Layers	1	Industrial	Machine	0.008353	0.99575
DARTS_4Layers	1	Industrial	Machine	0.019088	0.99375
1_NEURON	1	Industrial	Machine	0.041957	0.994
BN	1	Industrial	Machine	0.007332	0.9995
BN_DO	1	Industrial	Machine	0.000719	0.999875
DO	1	Industrial	Machine	0.025702	0.996125
RELU	1	Industrial	Machine	0.110065	0.995625
RELU_BN	1	Industrial	Machine	0.000614	0.99975
RELU_BN_DO	1	Industrial	Machine	0.000883	0.999625
RELU_DO	1	Industrial	Machine	0.011556	0.997
NOIC_RELU_BN_DO	1	Industrial	Machine	0.000654	0.999875
DARTS_1Layers	1	Industrial	MiscIndustrial	0.042943	0.989
DARTS_2Layers	1	Industrial	MiscIndustrial	0.025794	0.98825
DARTS_3Layers	1	Industrial	MiscIndustrial	0.029553	0.99075
DARTS_4Layers	1	Industrial	MiscIndustrial	0.032734	0.992
1_NEURON	1	Industrial	MiscIndustrial	0.100059	0.9865
BN	1	Industrial	MiscIndustrial	0.085164	0.988375
BN_DO	1	Industrial	MiscIndustrial	0.091239	0.986625
DO	1	Industrial	MiscIndustrial	0.07831	0.98525
RELU	1	Industrial	MiscIndustrial	0.100115	0.990625
RELU_BN	1	Industrial	MiscIndustrial	0.038717	0.98925
RELU_BN_DO	1	Industrial	MiscIndustrial	0.040969	0.987875
RELU_DO	1	Industrial	MiscIndustrial	0.050159	0.986875
NOIC_RELU_BN_DO	1	Industrial	MiscIndustrial	0.043288	0.986875
DARTS_1Layers	1	Residential	MiscResidential	0.027737	0.9915
DARTS_2Layers	1	Residential	MiscResidential	0.028357	0.9905
DARTS_3Layers	1	Residential	MiscResidential	0.033262	0.9915
DARTS_4Layers	1	Residential	MiscResidential	0.039956	0.99025
1_NEURON	1	Residential	MiscResidential	0.088908	0.98925
BN	1	Residential	MiscResidential	0.189517	0.98775
BN_DO	1	Residential	MiscResidential	0.135576	0.9875
DO	1	Residential	MiscResidential	0.078543	0.986
RELU	1	Residential	MiscResidential	0.33277	0.984875
RELU_BN	1	Residential	MiscResidential	0.038209	0.98625
RELU_BN_DO	1	Residential	MiscResidential	0.034956	0.98625
RELU_DO	1	Residential	MiscResidential	0.033506	0.99025
NOIC_RELU_BN_DO	1	Residential	MiscResidential	0.031817	0.989875

DARTS_1Layers	1	Residential	Porch	6.49E-10	0.9995
DARTS_2Layers	1	Residential	Porch	0.016033	0.99875
DARTS_3Layers	1	Residential	Porch	0.004543	0.99925
DARTS_4Layers	1	Residential	Porch	0.008913	0.9995
1_NEURON	1	Residential	Porch	0.000672	0.999625
BN	1	Residential	Porch	0.000743	0.99975
BN_DO	1	Residential	Porch	0.000834	0.99975
DO	1	Residential	Porch	0.000116	1
RELU	1	Residential	Porch	0.001433	0.99975
RELU_BN	1	Residential	Porch	0.000319	1
RELU_BN_DO	1	Residential	Porch	0.000183	1
RELU_DO	1	Residential	Porch	0.00012	1
NOIC_RELU_BN_DO	1	Residential	Porch	2.83E-05	1
DARTS_1Layers	1	Residential	TiledRoof	0.017247	0.9905
DARTS_2Layers	1	Residential	TiledRoof	0.025883	0.9905
DARTS_3Layers	1	Residential	TiledRoof	0.028057	0.991
DARTS_4Layers	1	Residential	TiledRoof	0.035327	0.99125
1_NEURON	1	Residential	TiledRoof	0.062236	0.9925
BN	1	Residential	TiledRoof	0.053932	0.993125
BN_DO	1	Residential	TiledRoof	0.042283	0.9955
DO	1	Residential	TiledRoof	0.05128	0.994625
RELU	1	Residential	TiledRoof	0.298242	0.992625
RELU_BN	1	Residential	TiledRoof	0.022647	0.991
RELU_BN_DO	1	Residential	TiledRoof	0.014325	0.994625
RELU_DO	1	Residential	TiledRoof	0.021408	0.995625
NOIC_RELU_BN_DO	1	Residential	TiledRoof	0.014028	0.99575
DARTS_1Layers	2	Commercial	Billboard	0.080991	0.97975
DARTS_2Layers	2	Commercial	Billboard	0.06354	0.97925
DARTS_3Layers	2	Commercial	Billboard	0.102286	0.97875
DARTS_4Layers	2	Commercial	Billboard	0.117288	0.97925
1_NEURON	2	Commercial	Billboard	0.300098	0.96825
BN	2	Commercial	Billboard	0.216526	0.9685
BN_DO	2	Commercial	Billboard	0.204419	0.968625
DO	2	Commercial	Billboard	0.293504	0.963125
RELU	2	Commercial	Billboard	0.430979	0.878
RELU_BN	2	Commercial	Billboard	0.069055	0.975
RELU_BN_DO	2	Commercial	Billboard	0.073147	0.974
RELU_DO	2	Commercial	Billboard	0.084399	0.974375
NOIC_RELU_BN_DO	2	Commercial	Billboard	0.056689	0.982125
DARTS_1Layers	2	Commercial	Restaurant	0.132394	0.9655

DARTS_2Layers	2	Commercial	Restaurant	0.110074	0.9625
DARTS_3Layers	2	Commercial	Restaurant	0.115115	0.9625
DARTS_4Layers	2	Commercial	Restaurant	0.147397	0.958437
1_NEURON	2	Commercial	Restaurant	0.32245	0.965
BN	2	Commercial	Restaurant	0.298	0.95975
BN_DO	2	Commercial	Restaurant	0.233448	0.965375
DO	2	Commercial	Restaurant	0.295857	0.965
RELU	2	Commercial	Restaurant	0.234135	0.87975
RELU_BN	2	Commercial	Restaurant	0.074937	0.97425
RELU_BN_DO	2	Commercial	Restaurant	0.062071	0.97775
RELU_DO	2	Commercial	Restaurant	0.096571	0.9665
NOIC_RELU_BN_DO	2	Commercial	Restaurant	0.062994	0.9765
DARTS_1Layers	2	Commercial	Statue	0.006258	0.99875
DARTS_2Layers	2	Commercial	Statue	0.000183	1
DARTS_3Layers	2	Commercial	Statue	1.48E-07	0.99975
DARTS_4Layers	2	Commercial	Statue	3.19E-07	1
1_NEURON	2	Commercial	Statue	0.002471	0.99975
BN	2	Commercial	Statue	2.13E-10	1
BN_DO	2	Commercial	Statue	0.000362	0.99975
DO	2	Commercial	Statue	0.004115	0.998875
RELU	2	Commercial	Statue	0.06481	0.997875
RELU_BN	2	Commercial	Statue	4.16E-05	1
RELU_BN_DO	2	Commercial	Statue	1.00E-05	1
RELU_DO	2	Commercial	Statue	4.72E-05	1
NOIC_RELU_BN_DO	2	Commercial	Statue	0.005001	0.999375
DARTS_1Layers	2	Industrial	Chimney	0.016088	0.99825
DARTS_2Layers	2	Industrial	Chimney	0.01064	0.998
DARTS_3Layers	2	Industrial	Chimney	0.004999	0.99875
DARTS_4Layers	2	Industrial	Chimney	0.000441	1
1_NEURON	2	Industrial	Chimney	0.005988	0.998
BN	2	Industrial	Chimney	0.003857	0.999125
BN_DO	2	Industrial	Chimney	0.003292	0.9995
DO	2	Industrial	Chimney	0.002377	0.999
RELU	2	Industrial	Chimney	0.011921	0.9985
RELU_BN	2	Industrial	Chimney	0.000253	1
RELU_BN_DO	2	Industrial	Chimney	0.000216	1
RELU_DO	2	Industrial	Chimney	0.002074	0.9995
NOIC_RELU_BN_DO	2	Industrial	Chimney	0.006244	0.998375
DARTS_1Layers	2	Industrial	Machine	0.003977	0.99875
DARTS_2Layers	2	Industrial	Machine	0.003197	0.99875

DARTS_3Layers	2	Industrial	Machine	0.005148	0.998
DARTS_4Layers	2	Industrial	Machine	0.005344	0.9975
1_NEURON	2	Industrial	Machine	0.017896	0.9945
BN	2	Industrial	Machine	0.017882	0.997125
BN_DO	2	Industrial	Machine	0.009896	0.9985
DO	2	Industrial	Machine	0.014647	0.99625
RELU	2	Industrial	Machine	0.02269	0.996
RELU_BN	2	Industrial	Machine	0.003324	0.99875
RELU_BN_DO	2	Industrial	Machine	0.003318	0.999125
RELU_DO	2	Industrial	Machine	0.005268	0.99725
NOIC_RELU_BN_DO	2	Industrial	Machine	0.010454	0.997375
DARTS_1Layers	2	Industrial	MiscIndustrial	0.047201	0.98625
DARTS_2Layers	2	Industrial	MiscIndustrial	0.047567	0.98625
DARTS_3Layers	2	Industrial	MiscIndustrial	0.055695	0.98625
DARTS_4Layers	2	Industrial	MiscIndustrial	0.053847	0.9825
1_NEURON	2	Industrial	MiscIndustrial	0.104986	0.98375
BN	2	Industrial	MiscIndustrial	0.177743	0.98075
BN_DO	2	Industrial	MiscIndustrial	0.147553	0.9785
DO	2	Industrial	MiscIndustrial	0.114986	0.9825
RELU	2	Industrial	MiscIndustrial	0.384528	0.8855
RELU_BN	2	Industrial	MiscIndustrial	0.058783	0.98325
RELU_BN_DO	2	Industrial	MiscIndustrial	0.058069	0.984
RELU_DO	2	Industrial	MiscIndustrial	0.051349	0.9855
NOIC_RELU_BN_DO	2	Industrial	MiscIndustrial	0.053438	0.98725
DARTS_1Layers	2	Residential	MiscResidential	0.019817	0.9925
DARTS_2Layers	2	Residential	MiscResidential	0.018698	0.992
DARTS_3Layers	2	Residential	MiscResidential	0.021944	0.9935
DARTS_4Layers	2	Residential	MiscResidential	0.02235	0.98975
1_NEURON	2	Residential	MiscResidential	0.067925	0.9915
BN	2	Residential	MiscResidential	0.096572	0.989875
BN_DO	2	Residential	MiscResidential	0.080193	0.990875
DO	2	Residential	MiscResidential	0.051811	0.992125
RELU	2	Residential	MiscResidential	0.175321	0.9935
RELU_BN	2	Residential	MiscResidential	0.021235	0.99425
RELU_BN_DO	2	Residential	MiscResidential	0.022158	0.993625
RELU_DO	2	Residential	MiscResidential	0.165257	0.89375
NOIC_RELU_BN_DO	2	Residential	MiscResidential	0.017338	0.994875
DARTS_1Layers	2	Residential	Porch	0.008246	0.9985
DARTS_2Layers	2	Residential	Porch	0.000783	0.99875
DARTS_3Layers	2	Residential	Porch	0.005948	0.9985

DARTS_4Layers	2	Residential	Porch	0.000469	0.997
1_NEURON	2	Residential	Porch	0.011488	0.997375
BN	2	Residential	Porch	0.016089	0.9975
BN_DO	2	Residential	Porch	0.012151	0.998625
DO	2	Residential	Porch	0.004388	0.9985
RELU	2	Residential	Porch	0.022501	0.998125
RELU_BN	2	Residential	Porch	0.003584	0.998875
RELU_BN_DO	2	Residential	Porch	0.003639	0.998125
RELU_DO	2	Residential	Porch	0.005298	0.998625
NOIC_RELU_BN_DO	2	Residential	Porch	0.008197	0.998125
DARTS_1Layers	2	Residential	TiledRoof	0.010808	0.993
DARTS_2Layers	2	Residential	TiledRoof	0.016501	0.99425
DARTS_3Layers	2	Residential	TiledRoof	0.018537	0.99325
DARTS_4Layers	2	Residential	TiledRoof	0.015856	0.99125
1_NEURON	2	Residential	TiledRoof	0.081245	0.99
BN	2	Residential	TiledRoof	0.141931	0.986625
BN_DO	2	Residential	TiledRoof	0.133639	0.986375
DO	2	Residential	TiledRoof	0.089968	0.9915
RELU	2	Residential	TiledRoof	0.194491	0.990125
RELU_BN	2	Residential	TiledRoof	0.048093	0.9835
RELU_BN_DO	2	Residential	TiledRoof	0.045717	0.98525
RELU_DO	2	Residential	TiledRoof	0.067044	0.9915
NOIC_RELU_BN_DO	2	Residential	TiledRoof	0.023819	0.992625
DARTS_1Layers	3	Commercial	Billboard	0.23412	0.91825
DARTS_2Layers	3	Commercial	Billboard	0.241164	0.91725
DARTS_3Layers	3	Commercial	Billboard	0.220329	0.918
DARTS_4Layers	3	Commercial	Billboard	0.2334	0.9175
1_NEURON	3	Commercial	Billboard	0.196565	0.927
BN	3	Commercial	Billboard	0.198177	0.924125
BN_DO	3	Commercial	Billboard	0.195193	0.926875
DO	3	Commercial	Billboard	0.205076	0.924875
RELU	3	Commercial	Billboard	0.202533	0.923125
RELU_BN	3	Commercial	Billboard	0.192548	0.927875
RELU_BN_DO	3	Commercial	Billboard	0.190116	0.92625
RELU_DO	3	Commercial	Billboard	0.199525	0.9255
NOIC_RELU_BN_DO	3	Commercial	Billboard	0.188328	0.928
DARTS_1Layers	3	Commercial	Restaurant	0.462906	0.85525
DARTS_2Layers	3	Commercial	Restaurant	0.377012	0.85525
DARTS_3Layers	3	Commercial	Restaurant	0.367987	0.8555
DARTS_4Layers	3	Commercial	Restaurant	0.416071	0.85625

1_NEURON	3	Commercial	Restaurant	0.306461	0.874
BN	3	Commercial	Restaurant	0.307656	0.873125
BN_DO	3	Commercial	Restaurant	0.305109	0.872625
DO	3	Commercial	Restaurant	0.313092	0.86875
RELU	3	Commercial	Restaurant	0.330843	0.86825
RELU_BN	3	Commercial	Restaurant	0.306828	0.869375
RELU_BN_DO	3	Commercial	Restaurant	0.301967	0.875125
RELU_DO	3	Commercial	Restaurant	0.328772	0.872875
NOIC_RELU_BN_DO	3	Commercial	Restaurant	0.3027	0.8735
DARTS_1Layers	3	Commercial	Statue	0.368382	0.87325
DARTS_2Layers	3	Commercial	Statue	0.367081	0.87875
DARTS_3Layers	3	Commercial	Statue	0.56823	0.8735
DARTS_4Layers	3	Commercial	Statue	0.336663	0.883
1_NEURON	3	Commercial	Statue	0.249738	0.890125
BN	3	Commercial	Statue	0.240111	0.89125
BN_DO	3	Commercial	Statue	0.248702	0.88975
DO	3	Commercial	Statue	0.256303	0.885375
RELU	3	Commercial	Statue	0.26803	0.883
RELU_BN	3	Commercial	Statue	0.24732	0.888625
RELU_BN_DO	3	Commercial	Statue	0.249322	0.88625
RELU_DO	3	Commercial	Statue	0.28198	0.879375
NOIC_RELU_BN_DO	3	Commercial	Statue	0.255114	0.885
DARTS_1Layers	3	Industrial	Chimney	0.330398	0.8855
DARTS_2Layers	3	Industrial	Chimney	0.299241	0.8735
DARTS_3Layers	3	Industrial	Chimney	0.392369	0.88575
DARTS_4Layers	3	Industrial	Chimney	0.313106	0.8795
1_NEURON	3	Industrial	Chimney	1.493	0.89525
BN	3	Industrial	Chimney	0.979134	0.88175
BN_DO	3	Industrial	Chimney	0.897081	0.88575
DO	3	Industrial	Chimney	1.16711	0.890625
RELU	3	Industrial	Chimney	0.401535	0.909375
RELU_BN	3	Industrial	Chimney	0.299063	0.89375
RELU_BN_DO	3	Industrial	Chimney	0.289383	0.89975
RELU_DO	3	Industrial	Chimney	0.941992	0.893875
NOIC_RELU_BN_DO	3	Industrial	Chimney	0.219195	0.928125
DARTS_1Layers	3	Industrial	Machine	0.06185	0.97425
DARTS_2Layers	3	Industrial	Machine	0.104464	0.9735
DARTS_3Layers	3	Industrial	Machine	0.10661	0.96625
DARTS_4Layers	3	Industrial	Machine	0.102435	0.9695
1_NEURON	3	Industrial	Machine	0.245202	0.97025

BN	3	Industrial	Machine	0.196134	0.968
BN_DO	3	Industrial	Machine	0.21643	0.96475
DO	3	Industrial	Machine	0.177974	0.973875
RELU	3	Industrial	Machine	0.21676	0.9685
RELU_BN	3	Industrial	Machine	0.07981	0.973
RELU_BN_DO	3	Industrial	Machine	0.073553	0.976625
RELU_DO	3	Industrial	Machine	0.547907	0.964875
NOIC_RELU_BN_DO	3	Industrial	Machine	0.072696	0.97525
DARTS_1Layers	3	Industrial	MiscIndustrial	0.184444	0.9305
DARTS_2Layers	3	Industrial	MiscIndustrial	0.200325	0.9305
DARTS_3Layers	3	Industrial	MiscIndustrial	0.192119	0.92325
DARTS_4Layers	3	Industrial	MiscIndustrial	0.204283	0.9265
1_NEURON	3	Industrial	MiscIndustrial	0.482659	0.933875
BN	3	Industrial	MiscIndustrial	0.515587	0.919875
BN_DO	3	Industrial	MiscIndustrial	0.436471	0.9205
DO	3	Industrial	MiscIndustrial	0.417683	0.927625
RELU	3	Industrial	MiscIndustrial	0.440944	0.94125
RELU_BN	3	Industrial	MiscIndustrial	0.174083	0.940375
RELU_BN_DO	3	Industrial	MiscIndustrial	0.160221	0.943875
RELU_DO	3	Industrial	MiscIndustrial	0.455741	0.941625
NOIC_RELU_BN_DO	3	Industrial	MiscIndustrial	0.153403	0.9425
DARTS_1Layers	3	Residential	MiscResidential	0.069627	0.97425
DARTS_2Layers	3	Residential	MiscResidential	0.074597	0.97375
DARTS_3Layers	3	Residential	MiscResidential	0.075248	0.97475
DARTS_4Layers	3	Residential	MiscResidential	0.090414	0.97325
1_NEURON	3	Residential	MiscResidential	0.285749	0.97375
BN	3	Residential	MiscResidential	0.46005	0.969625
BN_DO	3	Residential	MiscResidential	0.388134	0.970375
DO	3	Residential	MiscResidential	0.260378	0.971625
RELU	3	Residential	MiscResidential	0.254864	0.9755
RELU_BN	3	Residential	MiscResidential	0.088228	0.974875
RELU_BN_DO	3	Residential	MiscResidential	0.080224	0.975
RELU_DO	3	Residential	MiscResidential	0.468214	0.976375
NOIC_RELU_BN_DO	3	Residential	MiscResidential	0.073838	0.976125
DARTS_1Layers	3	Residential	Porch	0.093939	0.979
DARTS_2Layers	3	Residential	Porch	0.107129	0.9715
DARTS_3Layers	3	Residential	Porch	0.078114	0.9705
DARTS_4Layers	3	Residential	Porch	0.107807	0.9695
1_NEURON	3	Residential	Porch	0.185191	0.975875
BN	3	Residential	Porch	0.267141	0.96375

BN_DO	3	Residential	Porch	0.201324	0.969375
DO	3	Residential	Porch	0.186293	0.972875
RELU	3	Residential	Porch	0.181241	0.975625
RELU_BN	3	Residential	Porch	0.073617	0.973125
RELU_BN_DO	3	Residential	Porch	0.067839	0.97575
RELU_DO	3	Residential	Porch	0.447259	0.974375
NOIC_RELU_BN_DO	3	Residential	Porch	0.041287	0.98675
DARTS_1Layers	3	Residential	TiledRoof	0.049236	0.98575
DARTS_2Layers	3	Residential	TiledRoof	0.036171	0.98425
DARTS_3Layers	3	Residential	TiledRoof	0.044582	0.983
DARTS_4Layers	3	Residential	TiledRoof	0.076843	0.9805
1_NEURON	3	Residential	TiledRoof	0.23915	0.982875
BN	3	Residential	TiledRoof	0.2145	0.985
BN_DO	3	Residential	TiledRoof	0.173966	0.984875
DO	3	Residential	TiledRoof	0.177395	0.983375
RELU	3	Residential	TiledRoof	0.188262	0.984625
RELU_BN	3	Residential	TiledRoof	0.055507	0.984
RELU_BN_DO	3	Residential	TiledRoof	0.050832	0.986875
RELU_DO	3	Residential	TiledRoof	0.67711	0.986
NOIC_RELU_BN_DO	3	Residential	TiledRoof	0.049086	0.987875
DARTS_1Layers	4	Commercial	Billboard	0.219685	0.9095
DARTS_2Layers	4	Commercial	Billboard	0.236745	0.9115
DARTS_3Layers	4	Commercial	Billboard	0.251736	0.914
DARTS_4Layers	4	Commercial	Billboard	0.28373	0.91125
1_NEURON	4	Commercial	Billboard	1.362873	0.903625
BN	4	Commercial	Billboard	0.924122	0.902
BN_DO	4	Commercial	Billboard	0.807652	0.89775
DO	4	Commercial	Billboard	1.151355	0.900125
RELU	4	Commercial	Billboard	0.424394	0.90725
RELU_BN	4	Commercial	Billboard	0.270118	0.909875
RELU_BN_DO	4	Commercial	Billboard	0.264148	0.91175
RELU_DO	4	Commercial	Billboard	0.467942	0.905125
NOIC_RELU_BN_DO	4	Commercial	Billboard	0.245888	0.91725
DARTS_1Layers	4	Commercial	Restaurant	0.403877	0.833
DARTS_2Layers	4	Commercial	Restaurant	0.376365	0.85575
DARTS_3Layers	4	Commercial	Restaurant	0.427812	0.84625
DARTS_4Layers	4	Commercial	Restaurant	0.368568	0.82925
1_NEURON	4	Commercial	Restaurant	2.003647	0.84125
BN	4	Commercial	Restaurant	1.426182	0.853625
BN_DO	4	Commercial	Restaurant	1.161304	0.862375

DO	4	Commercial	Restaurant	1.57004	0.851125
RELU	4	Commercial	Restaurant	0.444941	0.858875
RELU_BN	4	Commercial	Restaurant	0.383581	0.87575
RELU_BN_DO	4	Commercial	Restaurant	0.375664	0.8765
RELU_DO	4	Commercial	Restaurant	0.613303	0.8605
NOIC_RELU_BN_DO	4	Commercial	Restaurant	0.366633	0.880875
DARTS_1Layers	4	Commercial	Statue	0.205257	0.925
DARTS_2Layers	4	Commercial	Statue	0.23701	0.91825
DARTS_3Layers	4	Commercial	Statue	0.29899	0.92075
DARTS_4Layers	4	Commercial	Statue	0.205383	0.93125
1_NEURON	4	Commercial	Statue	0.705306	0.9325
BN	4	Commercial	Statue	0.510108	0.937125
BN_DO	4	Commercial	Statue	0.469029	0.93625
DO	4	Commercial	Statue	0.560469	0.937
RELU	4	Commercial	Statue	0.21454	0.936625
RELU_BN	4	Commercial	Statue	0.168795	0.9375
RELU_BN_DO	4	Commercial	Statue	0.1594	0.942375
RELU_DO	4	Commercial	Statue	0.351978	0.938125
NOIC_RELU_BN_DO	4	Commercial	Statue	0.124977	0.954375
DARTS_1Layers	4	Industrial	Chimney	0.260287	0.9055
DARTS_2Layers	4	Industrial	Chimney	0.309942	0.90925
DARTS_3Layers	4	Industrial	Chimney	0.28489	0.90175
DARTS_4Layers	4	Industrial	Chimney	0.265788	0.9055
1_NEURON	4	Industrial	Chimney	1.113035	0.91725
BN	4	Industrial	Chimney	0.639159	0.90475
BN_DO	4	Industrial	Chimney	0.565593	0.9095
DO	4	Industrial	Chimney	0.943181	0.913125
RELU	4	Industrial	Chimney	0.385498	0.918375
RELU_BN	4	Industrial	Chimney	0.217317	0.918625
RELU_BN_DO	4	Industrial	Chimney	0.197984	0.926875
RELU_DO	4	Industrial	Chimney	0.781115	0.916375
NOIC_RELU_BN_DO	4	Industrial	Chimney	0.171672	0.938125
DARTS_1Layers	4	Industrial	Machine	0.167378	0.95625
DARTS_2Layers	4	Industrial	Machine	0.167829	0.9575
DARTS_3Layers	4	Industrial	Machine	0.171529	0.9515
DARTS_4Layers	4	Industrial	Machine	0.140253	0.95125
1_NEURON	4	Industrial	Machine	0.363364	0.957125
BN	4	Industrial	Machine	0.32335	0.941875
BN_DO	4	Industrial	Machine	0.247304	0.950375
DO	4	Industrial	Machine	0.35623	0.95475

RELU	4	Industrial	Machine	0.27136	0.95925
RELU_BN	4	Industrial	Machine	0.108147	0.96225
RELU_BN_DO	4	Industrial	Machine	0.099579	0.96275
RELU_DO	4	Industrial	Machine	0.312185	0.956375
NOIC_RELU_BN_DO	4	Industrial	Machine	0.093719	0.965375
DARTS_1Layers	4	Industrial	MiscIndustrial	0.183862	0.9495
DARTS_2Layers	4	Industrial	MiscIndustrial	0.198016	0.948
DARTS_3Layers	4	Industrial	MiscIndustrial	0.221502	0.95025
DARTS_4Layers	4	Industrial	MiscIndustrial	0.195269	0.94525
1_NEURON	4	Industrial	MiscIndustrial	0.549094	0.934875
BN	4	Industrial	MiscIndustrial	0.494457	0.92525
BN_DO	4	Industrial	MiscIndustrial	0.393269	0.926
DO	4	Industrial	MiscIndustrial	0.517896	0.92675
RELU	4	Industrial	MiscIndustrial	0.428148	0.937375
RELU_BN	4	Industrial	MiscIndustrial	0.187486	0.9385
RELU_BN_DO	4	Industrial	MiscIndustrial	0.18474	0.93875
RELU_DO	4	Industrial	MiscIndustrial	0.409268	0.934875
NOIC_RELU_BN_DO	4	Industrial	MiscIndustrial	0.164474	0.94575
DARTS_1Layers	4	Residential	MiscResidential	0.071764	0.984
DARTS_2Layers	4	Residential	MiscResidential	0.054094	0.97925
DARTS_3Layers	4	Residential	MiscResidential	0.083853	0.97975
DARTS_4Layers	4	Residential	MiscResidential	0.072343	0.98025
1_NEURON	4	Residential	MiscResidential	0.358403	0.96725
BN	4	Residential	MiscResidential	0.39834	0.969875
BN_DO	4	Residential	MiscResidential	0.328076	0.971
DO	4	Residential	MiscResidential	0.364619	0.966
RELU	4	Residential	MiscResidential	0.272415	0.969375
RELU_BN	4	Residential	MiscResidential	0.089624	0.97325
RELU_BN_DO	4	Residential	MiscResidential	0.082458	0.976
RELU_DO	4	Residential	MiscResidential	0.537835	0.971375
NOIC_RELU_BN_DO	4	Residential	MiscResidential	0.079075	0.98175
DARTS_1Layers	4	Residential	Porch	0.068831	0.973
DARTS_2Layers	4	Residential	Porch	0.07415	0.973
DARTS_3Layers	4	Residential	Porch	0.131897	0.977
DARTS_4Layers	4	Residential	Porch	0.054645	0.976
1_NEURON	4	Residential	Porch	0.156972	0.975
BN	4	Residential	Porch	0.21445	0.9605
BN_DO	4	Residential	Porch	0.191613	0.964625
DO	4	Residential	Porch	0.133424	0.98225
RELU	4	Residential	Porch	0.09861	0.981875

RELU_BN	4	Residential	Porch	0.079927	0.96725
RELU_BN_DO	4	Residential	Porch	0.070749	0.972625
RELU_DO	4	Residential	Porch	0.215422	0.977125
NOIC_RELU_BN_DO	4	Residential	Porch	0.026204	0.990625
DARTS_1Layers	4	Residential	TiledRoof	0.073157	0.984
DARTS_2Layers	4	Residential	TiledRoof	0.053292	0.98125
DARTS_3Layers	4	Residential	TiledRoof	0.067465	0.98225
DARTS_4Layers	4	Residential	TiledRoof	0.053243	0.97875
1_NEURON	4	Residential	TiledRoof	0.145742	0.98575
BN	4	Residential	TiledRoof	0.159986	0.986
BN_DO	4	Residential	TiledRoof	0.123015	0.986375
DO	4	Residential	TiledRoof	0.142771	0.9865
RELU	4	Residential	TiledRoof	0.273104	0.980375
RELU_BN	4	Residential	TiledRoof	0.048136	0.9875
RELU_BN_DO	4	Residential	TiledRoof	0.044555	0.988375
RELU_DO	4	Residential	TiledRoof	0.267344	0.984
NOIC_RELU_BN_DO	4	Residential	TiledRoof	0.029267	0.992
DARTS_1Layers	5	Commercial	Billboard	0.263071	0.89625
DARTS_2Layers	5	Commercial	Billboard	0.269954	0.89775
DARTS_3Layers	5	Commercial	Billboard	0.262851	0.90075
DARTS_4Layers	5	Commercial	Billboard	0.267564	0.90625
1_NEURON	5	Commercial	Billboard	0.295987	0.8845
BN	5	Commercial	Billboard	0.294536	0.88725
BN_DO	5	Commercial	Billboard	0.294782	0.884625
DO	5	Commercial	Billboard	0.299694	0.883625
RELU	5	Commercial	Billboard	0.303973	0.882625
RELU_BN	5	Commercial	Billboard	0.290025	0.88875
RELU_BN_DO	5	Commercial	Billboard	0.290802	0.888375
RELU_DO	5	Commercial	Billboard	0.296022	0.883375
NOIC_RELU_BN_DO	5	Commercial	Billboard	0.288953	0.888
DARTS_1Layers	5	Commercial	Restaurant	0.409261	0.81825
DARTS_2Layers	5	Commercial	Restaurant	0.594763	0.81475
DARTS_3Layers	5	Commercial	Restaurant	0.406538	0.82425
DARTS_4Layers	5	Commercial	Restaurant	0.414898	0.824
1_NEURON	5	Commercial	Restaurant	0.434742	0.820875
BN	5	Commercial	Restaurant	0.414328	0.82475
BN_DO	5	Commercial	Restaurant	0.410753	0.826375
DO	5	Commercial	Restaurant	0.438079	0.825
RELU	5	Commercial	Restaurant	0.432963	0.83225
RELU_BN	5	Commercial	Restaurant	0.409718	0.8295

RELU_BN_DO	5	Commercial	Restaurant	0.409405	0.832875
RELU_DO	5	Commercial	Restaurant	0.443892	0.834625
NOIC_RELU_BN_DO	5	Commercial	Restaurant	0.409784	0.830125
DARTS_1Layers	5	Commercial	Statue	0.297124	0.875
DARTS_2Layers	5	Commercial	Statue	0.292849	0.87575
DARTS_3Layers	5	Commercial	Statue	0.30031	0.878
DARTS_4Layers	5	Commercial	Statue	0.289821	0.88025
1_NEURON	5	Commercial	Statue	0.270219	0.893375
BN	5	Commercial	Statue	0.256248	0.892125
BN_DO	5	Commercial	Statue	0.261563	0.890875
DO	5	Commercial	Statue	0.276887	0.89075
RELU	5	Commercial	Statue	0.32727	0.88925
RELU_BN	5	Commercial	Statue	0.265886	0.8915
RELU_BN_DO	5	Commercial	Statue	0.272977	0.89225
RELU_DO	5	Commercial	Statue	0.321118	0.889875
NOIC_RELU_BN_DO	5	Commercial	Statue	0.273895	0.8915
DARTS_1Layers	5	Industrial	Chimney	0.520729	0.75375
DARTS_2Layers	5	Industrial	Chimney	0.528812	0.754
DARTS_3Layers	5	Industrial	Chimney	0.555521	0.75225
DARTS_4Layers	5	Industrial	Chimney	0.557624	0.745
1_NEURON	5	Industrial	Chimney	0.513225	0.754625
BN	5	Industrial	Chimney	0.510879	0.755375
BN_DO	5	Industrial	Chimney	0.508681	0.758625
DO	5	Industrial	Chimney	0.517733	0.75425
RELU	5	Industrial	Chimney	0.512798	0.755625
RELU_BN	5	Industrial	Chimney	0.508893	0.7595
RELU_BN_DO	5	Industrial	Chimney	0.506552	0.763125
RELU_DO	5	Industrial	Chimney	0.507517	0.759375
NOIC_RELU_BN_DO	5	Industrial	Chimney	0.509203	0.75975
DARTS_1Layers	5	Industrial	Machine	0.380907	0.83125
DARTS_2Layers	5	Industrial	Machine	0.382478	0.8295
DARTS_3Layers	5	Industrial	Machine	0.381227	0.8305
DARTS_4Layers	5	Industrial	Machine	0.390785	0.82925
1_NEURON	5	Industrial	Machine	0.374034	0.836875
BN	5	Industrial	Machine	0.377733	0.836125
BN_DO	5	Industrial	Machine	0.377433	0.8355
DO	5	Industrial	Machine	0.374498	0.83725
RELU	5	Industrial	Machine	0.376687	0.83625
RELU_BN	5	Industrial	Machine	0.374112	0.835625
RELU_BN_DO	5	Industrial	Machine	0.374558	0.837

RELU_DO	5	Industrial	Machine	0.37493	0.836
NOIC_RELU_BN_DO	5	Industrial	Machine	0.373251	0.836125
DARTS_1Layers	5	Industrial	MiscIndustrial	0.362913	0.859
DARTS_2Layers	5	Industrial	MiscIndustrial	0.308554	0.86525
DARTS_3Layers	5	Industrial	MiscIndustrial	0.316981	0.871
DARTS_4Layers	5	Industrial	MiscIndustrial	0.305741	0.867
1_NEURON	5	Industrial	MiscIndustrial	0.311337	0.861625
BN	5	Industrial	MiscIndustrial	0.312888	0.8625
BN_DO	5	Industrial	MiscIndustrial	0.312649	0.86175
DO	5	Industrial	MiscIndustrial	0.313069	0.862375
RELU	5	Industrial	MiscIndustrial	0.31537	0.86225
RELU_BN	5	Industrial	MiscIndustrial	0.311368	0.8615
RELU_BN_DO	5	Industrial	MiscIndustrial	0.312585	0.86225
RELU_DO	5	Industrial	MiscIndustrial	0.315377	0.864625
NOIC_RELU_BN_DO	5	Industrial	MiscIndustrial	0.312781	0.86225
DARTS_1Layers	5	Residential	MiscResidential	0.18465	0.9345
DARTS_2Layers	5	Residential	MiscResidential	0.177286	0.934
DARTS_3Layers	5	Residential	MiscResidential	0.198692	0.93525
DARTS_4Layers	5	Residential	MiscResidential	0.180567	0.9365
1_NEURON	5	Residential	MiscResidential	0.14834	0.9505
BN	5	Residential	MiscResidential	0.146829	0.949
BN_DO	5	Residential	MiscResidential	0.150009	0.94775
DO	5	Residential	MiscResidential	0.149404	0.948
RELU	5	Residential	MiscResidential	0.15862	0.945
RELU_BN	5	Residential	MiscResidential	0.147765	0.949125
RELU_BN_DO	5	Residential	MiscResidential	0.14805	0.944375
RELU_DO	5	Residential	MiscResidential	0.159472	0.945875
NOIC_RELU_BN_DO	5	Residential	MiscResidential	0.148503	0.94675
DARTS_1Layers	5	Residential	Porch	0.29509	0.8805
DARTS_2Layers	5	Residential	Porch	0.301459	0.8795
DARTS_3Layers	5	Residential	Porch	0.29005	0.87875
DARTS_4Layers	5	Residential	Porch	0.31476	0.87975
1_NEURON	5	Residential	Porch	0.252487	0.901125
BN	5	Residential	Porch	0.248631	0.901375
BN_DO	5	Residential	Porch	0.2497	0.902
DO	5	Residential	Porch	0.254575	0.90175
RELU	5	Residential	Porch	0.255724	0.901
RELU_BN	5	Residential	Porch	0.249184	0.897625
RELU_BN_DO	5	Residential	Porch	0.256188	0.90025
RELU_DO	5	Residential	Porch	0.258335	0.89975

NOIC_RELU_BN_DO	5	Residential	Porch	0.252131	0.900875
DARTS_1Layers	5	Residential	TiledRoof	0.282141	0.9035
DARTS_2Layers	5	Residential	TiledRoof	0.285773	0.9035
DARTS_3Layers	5	Residential	TiledRoof	0.364977	0.89
DARTS_4Layers	5	Residential	TiledRoof	0.391176	0.88775
1_NEURON	5	Residential	TiledRoof	0.267358	0.9005
BN	5	Residential	TiledRoof	0.264404	0.901
BN_DO	5	Residential	TiledRoof	0.276791	0.8975
DO	5	Residential	TiledRoof	0.285917	0.895
RELU	5	Residential	TiledRoof	0.266905	0.89975
RELU_BN	5	Residential	TiledRoof	0.256959	0.903
RELU_BN_DO	5	Residential	TiledRoof	0.270218	0.8975
RELU_DO	5	Residential	TiledRoof	0.286512	0.893125
NOIC_RELU_BN_DO	5	Residential	TiledRoof	0.273575	0.898625
DARTS_1Layers	6	Commercial	Billboard	0.267498	0.89325
DARTS_2Layers	6	Commercial	Billboard	0.267237	0.892
DARTS_3Layers	6	Commercial	Billboard	0.267441	0.8895
DARTS_4Layers	6	Commercial	Billboard	0.268592	0.8925
1_NEURON	6	Commercial	Billboard	0.279169	0.899125
BN	6	Commercial	Billboard	0.276769	0.899625
BN_DO	6	Commercial	Billboard	0.276018	0.900625
DO	6	Commercial	Billboard	0.278323	0.898875
RELU	6	Commercial	Billboard	0.280548	0.895125
RELU_BN	6	Commercial	Billboard	0.276951	0.900875
RELU_BN_DO	6	Commercial	Billboard	0.277433	0.900125
RELU_DO	6	Commercial	Billboard	0.277738	0.896875
NOIC_RELU_BN_DO	6	Commercial	Billboard	0.275573	0.899125
DARTS_1Layers	6	Commercial	Restaurant	0.376386	0.83475
DARTS_2Layers	6	Commercial	Restaurant	0.381562	0.83675
DARTS_3Layers	6	Commercial	Restaurant	0.38033	0.83325
DARTS_4Layers	6	Commercial	Restaurant	0.379033	0.8305
1_NEURON	6	Commercial	Restaurant	0.395535	0.83725
BN	6	Commercial	Restaurant	0.397233	0.83025
BN_DO	6	Commercial	Restaurant	0.394505	0.8345
DO	6	Commercial	Restaurant	0.398283	0.836
RELU	6	Commercial	Restaurant	0.409278	0.834375
RELU_BN	6	Commercial	Restaurant	0.390757	0.836875
RELU_BN_DO	6	Commercial	Restaurant	0.392336	0.836
RELU_DO	6	Commercial	Restaurant	0.401897	0.838
NOIC_RELU_BN_DO	6	Commercial	Restaurant	0.391522	0.837

DARTS_1Layers	6	Commercial	Statue	0.315139	0.8725
DARTS_2Layers	6	Commercial	Statue	0.348942	0.86425
DARTS_3Layers	6	Commercial	Statue	0.299886	0.87375
DARTS_4Layers	6	Commercial	Statue	0.303491	0.87275
1_NEURON	6	Commercial	Statue	0.290317	0.87525
BN	6	Commercial	Statue	0.293261	0.874125
BN_DO	6	Commercial	Statue	0.294456	0.874
DO	6	Commercial	Statue	0.294665	0.875375
RELU	6	Commercial	Statue	0.309186	0.8735
RELU_BN	6	Commercial	Statue	0.294756	0.87425
RELU_BN_DO	6	Commercial	Statue	0.296169	0.8735
RELU_DO	6	Commercial	Statue	0.310354	0.8705
NOIC_RELU_BN_DO	6	Commercial	Statue	0.293264	0.874
DARTS_1Layers	6	Industrial	Chimney	0.529594	0.76025
DARTS_2Layers	6	Industrial	Chimney	0.527586	0.75875
DARTS_3Layers	6	Industrial	Chimney	0.526065	0.7595
DARTS_4Layers	6	Industrial	Chimney	0.522618	0.75625
1_NEURON	6	Industrial	Chimney	0.49571	0.779
BN	6	Industrial	Chimney	0.496315	0.7805
BN_DO	6	Industrial	Chimney	0.493083	0.784125
DO	6	Industrial	Chimney	0.497088	0.783125
RELU	6	Industrial	Chimney	0.492917	0.786625
RELU_BN	6	Industrial	Chimney	0.49875	0.78225
RELU_BN_DO	6	Industrial	Chimney	0.491921	0.784875
RELU_DO	6	Industrial	Chimney	0.495085	0.785
NOIC_RELU_BN_DO	6	Industrial	Chimney	0.491961	0.78625
DARTS_1Layers	6	Industrial	Machine	0.380153	0.8545
DARTS_2Layers	6	Industrial	Machine	0.351567	0.851
DARTS_3Layers	6	Industrial	Machine	0.367986	0.8485
DARTS_4Layers	6	Industrial	Machine	0.391988	0.85075
1_NEURON	6	Industrial	Machine	0.339255	0.850125
BN	6	Industrial	Machine	0.322138	0.85175
BN_DO	6	Industrial	Machine	0.332169	0.849875
DO	6	Industrial	Machine	0.339457	0.84875
RELU	6	Industrial	Machine	0.336378	0.85075
RELU_BN	6	Industrial	Machine	0.33126	0.851375
RELU_BN_DO	6	Industrial	Machine	0.334265	0.851625
RELU_DO	6	Industrial	Machine	0.349024	0.851375
NOIC_RELU_BN_DO	6	Industrial	Machine	0.340128	0.85025
DARTS_1Layers	6	Industrial	MiscIndustrial	0.311788	0.87675

DARTS_2Layers	6	Industrial	MiscIndustrial	0.319045	0.87375
DARTS_3Layers	6	Industrial	MiscIndustrial	0.368815	0.8675
DARTS_4Layers	6	Industrial	MiscIndustrial	0.356621	0.86825
1_NEURON	6	Industrial	MiscIndustrial	0.307315	0.877875
BN	6	Industrial	MiscIndustrial	0.306694	0.875625
BN_DO	6	Industrial	MiscIndustrial	0.308575	0.877375
DO	6	Industrial	MiscIndustrial	0.308807	0.877375
RELU	6	Industrial	MiscIndustrial	0.310176	0.875625
RELU_BN	6	Industrial	MiscIndustrial	0.308541	0.876875
RELU_BN_DO	6	Industrial	MiscIndustrial	0.309602	0.876125
RELU_DO	6	Industrial	MiscIndustrial	0.312264	0.876625
NOIC_RELU_BN_DO	6	Industrial	MiscIndustrial	0.3101	0.878
DARTS_1Layers	6	Residential	MiscResidential	0.161987	0.95275
DARTS_2Layers	6	Residential	MiscResidential	0.15336	0.952
DARTS_3Layers	6	Residential	MiscResidential	0.194413	0.95075
DARTS_4Layers	6	Residential	MiscResidential	0.20885	0.949
1_NEURON	6	Residential	MiscResidential	0.153142	0.949625
BN	6	Residential	MiscResidential	0.153138	0.947625
BN_DO	6	Residential	MiscResidential	0.159779	0.945875
DO	6	Residential	MiscResidential	0.164632	0.948125
RELU	6	Residential	MiscResidential	0.16927	0.94825
RELU_BN	6	Residential	MiscResidential	0.14888	0.9485
RELU_BN_DO	6	Residential	MiscResidential	0.160753	0.947375
RELU_DO	6	Residential	MiscResidential	0.171588	0.94325
NOIC_RELU_BN_DO	6	Residential	MiscResidential	0.151888	0.948625
DARTS_1Layers	6	Residential	Porch	0.275287	0.873
DARTS_2Layers	6	Residential	Porch	0.280778	0.87675
DARTS_3Layers	6	Residential	Porch	0.391286	0.8755
DARTS_4Layers	6	Residential	Porch	0.348825	0.8775
1_NEURON	6	Residential	Porch	0.248339	0.884625
BN	6	Residential	Porch	0.247223	0.887625
BN_DO	6	Residential	Porch	0.248541	0.89075
DO	6	Residential	Porch	0.251814	0.886875
RELU	6	Residential	Porch	0.25627	0.887375
RELU_BN	6	Residential	Porch	0.25286	0.89
RELU_BN_DO	6	Residential	Porch	0.254895	0.892625
RELU_DO	6	Residential	Porch	0.262867	0.894375
NOIC_RELU_BN_DO	6	Residential	Porch	0.2512	0.893625
DARTS_1Layers	6	Residential	TiledRoof	0.200503	0.92925
DARTS_2Layers	6	Residential	TiledRoof	0.196054	0.93175

DARTS_3Layers	6	Residential	TiledRoof	0.213652	0.92725
DARTS_4Layers	6	Residential	TiledRoof	0.207991	0.92825
1_NEURON	6	Residential	TiledRoof	0.207175	0.919375
BN	6	Residential	TiledRoof	0.209143	0.915375
BN_DO	6	Residential	TiledRoof	0.226424	0.91325
DO	6	Residential	TiledRoof	0.233348	0.913375
RELU	6	Residential	TiledRoof	0.209293	0.920125
RELU_BN	6	Residential	TiledRoof	0.208607	0.919
RELU_BN_DO	6	Residential	TiledRoof	0.218657	0.915625
RELU_DO	6	Residential	TiledRoof	0.226436	0.91575
NOIC_RELU_BN_DO	6	Residential	TiledRoof	0.199233	0.92