UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

COMMITTEE OF NAS-BASED MODELS

BRUNO SILVA SETTE

ORIENTADOR: PROF. DR. DIEGO FURTADO SILVA

São Carlos – SP

Maio/2021

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

COMMITTEE OF NAS-BASED MODELS

BRUNO SILVA SETTE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Aprendizado de máquina
Orientador: Prof. Dr. Diego Furtado Silva

São Carlos – SP

Maio/2021

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato Bruno Silva Sette, realizada em 28/05/2021.

Comissão Julgadora:

Prof. Dr. Diego Furtado Silva (UFSCar)

Prof. Dr. Ricardo Cerri (UFSCar)

Prof. Dr. Moacir Antonelli Ponti (ICMC/USP)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Aos deuses.

AGRADECIMENTOS

Agradeço, primeiramente, a minha família, que em todos os momentos me ajudaram a superar as dificuldades durante meu percurso, me proporcionando abrigo, conselhos e sabedoria para que eu pudesse dar o meu melhor. Agradeço também pela criação que me foi dada, a qual permitiu que eu chegasse a ser quem sou hoje.

Ao orientador deste presente projeto, Prof. Dr. Diego Furtado Silva, que através das suas vastas experiências e competências, me auxiliou em todas as dúvidas e questionamentos que apresentei. Sem sua assistência e dedicação a fim de transmitir seus conhecimentos para realização das pesquisas, não seria possível o correto desenvolvimento deste trabalho.

Aos membros da banca examinadora, Profª. Dr. Ricardo Cerri e Prof. Dr. Moacir Antonelli Ponti, e aos membros suplentes, Profª. Dra. Helena de Medeiros Caseli, Prof. Dr. Ricardo Augusto Souza Fernandes e Prof. Dr. Vinícius Mourão Alves de Souza, que tão gentilmente aceitaram participar e colaborar com este projeto de pesquisa. Suas contribuições serão utilizadas prontamente a fim de aprimorar as ideias e atingir os resultados esperados.

À empresa B2W Digital, parceira desta pesquisa, que me proporcionou a possibilidade de trabalhar dentro de um projeto real da empresa em paralelo com a realização do meu projeto de mestrado. Seu suporte que tornou possível a saída de minha cidade natal para realizar meus estudos e alcançar meus sonhos e objetivos.

Aos professores escolhidos para auxiliar no projeto Apache Marvin-AI em conjunto com meu orientador, Profª. Dra. Helena de Medeiros Caseli e Prof. Dr. Daniel Lucrédio, que além de prestar assistências e apoios durante o projeto da empresa, também realizaram diversas contribuições para o projeto de pesquisa do mestrado. Os três professores citados foram muito coerentes e não ajudaram apenas nas relações acadêmicas, mas também foram extremamente compreensíveis nas relações humanas. Os três são ótimas pessoas e quero levar para a vida toda como mestres e amigos.

Aos companheiros que entraram no mestrado e no projeto da B2W Digital na mesma época que eu, Fernando Rezende Zagatti, Lucas Cardoso Silva e Lucas Nildaimon dos Santos Silva,

que além de acrescentarem muito conhecimento durante nossas discussões e trocas de experiências, são amigos muito próximos que tive o prazer de conhecer. Sem eles, nada disso seria possível.

Ao corpo docente do Departamento de Computação da Universidade Federal de São Carlos pelo conteúdo que me apresentaram e da forma que me foi apresentado, tenho certeza que cada dia foi uma experiência de aprendizado única, tanto para mim quanto para os professores e demais alunos.

Por fim, a todos os amigos e amigas que conheci no Departamento de Computação, pelo convívio, amizade e apoio. Também, a todos os profissionais e acadêmicos que contribuem compartilhando seus conhecimentos e experiências, direta ou indiretamente, permitindo a realização desta pesquisa, o meu sincero agradecimento. Tenho certeza que todas as novas ideias e descobertas serão levadas para toda a vida.

Don’t worry about it if you don’t understand

Andrew Ng

RESUMO

A Busca por Arquitetura de Redes Neurais (*Network Architecture Search* – NAS) tem obtido ótimos resultados e gerou modelos comparáveis às classificações humanas. Automatizar a definição de uma arquitetura neural reduz a necessidade de esforços de trabalho especializado e mitiga o preconceito humano do projeto de arquitetura. As técnicas de NAS geralmente consistem em um algoritmo para buscar a melhor arquitetura em um espaço pré-determinado de parâmetros ou funções. Devido ao número de parâmetros de arquiteturas neurais profundas, esse espaço de busca inclui milhões de combinações, o que torna o NAS um procedimento custoso e pode levar a busca a super ajustar o conjunto de treinamento. Para reduzir a complexidade dos espaços de busca NAS e ainda obter resultados competitivos, propomos o CoNAS, um comitê de modelos baseados em NAS, restringindo os espaços de busca para realizar a Busca Diferenciável de Arquitetura (*Differentiable ARchiTecture Search* – DARTS). Nossos resultados apontam para uma maior precisão em relação ao DARTS tanto em cenários em que a rede é treinada do zero quanto usando uma abordagem de aprendizagem por transferência.

Keywords: CoNAS, ensemble,DARTS

ABSTRACT

Network Architecture Search (NAS) has achieved great results and generated models comparable with humans' classifications. Automating the definition of a neural architecture reduces the need for expert work efforts and mitigates human bias from architecture design. NAS techniques usually consist of an algorithm to search for the best architecture in a predetermined space of parameters or functions. Due to the number of deep neural architectures' parameters, this search space includes millions of combinations, making NAS a cost procedure and may lead the search to overfit the training set. To reduce NAS search spaces' complexity and still obtain competitive results, we propose CoNAS, a committee of NAS-based models, by restricting the search spaces to perform Differentiable ARchiTecture Search (DARTS). Our results point to improved accuracy over DARTS on two experimental scenarios: raining from scratch and using a transfer learning approach.

Keywords: CoNAS, ensemble,DARTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

BO – *Bayesian Optimization*

CONAS – *Cmmitte Of Nas-based Models*

DARTS – *Differentiable Architecture Search*

GD – *Gradient Descent*

GS – *Grid Search*

HPO – *Hyperparameter optimization*

RL – *Reinforcement learning*

RS – *Random Search*

SGD – *Stochastic Gradient Descent*

SMBO – *Sequential Model-Based Optimization*

TL – *Transfer learning*

CONTENTS

Chapter 1

INTRODUCTION

Machine Learning (ML) (WU; XIE, 2021) has become a part of people's everyday life. Due to its relevance to society, many research efforts on ML have been made in the last few decades. ML applications are seen in a wide range of domains, such as automatic speech recognition, personal assistants, autonomous cars, computer vision, and natural language processing tasks.

Motivated by industry needs and ML issues in general, Automated Machine Learning (AutoML) (HE; ZHAO; CHU, 2019) has appeared in recent years as a subarea of ML. AutoML's goal is to decrease human interference in the learning process and make it increasingly accessible to non-expert users and less costly for the industry. For this, some techniques have emerged to automate or optimize some stages of the machine learning pipeline, for another side. Transfer learning (TL) works well in relatively similar domains. This becomes clear when we look at one of our experiments that will be presented in this dissertation. Hutter, Kotthoff and Vanschoren (2018) define two classes of AutoML approaches:

- The first is a general case. The learning process considered in this case is a combination of feature engineering ,model selection, algorithm selection , and hyperparameter optimization. This process is usually applied to structured data.
- The second is the Network Architecture Search (NAS), which aims to look for suitable architectures of deep networks that solve a particular learning problem. Where NAS is considered a sub-area of AutoML.

There are three main reasons why we discuss these cases in parallel. First, the NAS itself is currently a hot research topic, in which many articles are being published (REN et al., 2021). The second reason is that the application domain for deep networks is relatively clear, such as low semantic level data, such as image pixels.

This work presents a novel research effort on improving the accuracy of NAS-based models. For this reason, this chapter continues with a brief introduction to this knowledge domain, motivating the need and presenting the main contributions of our proposal.

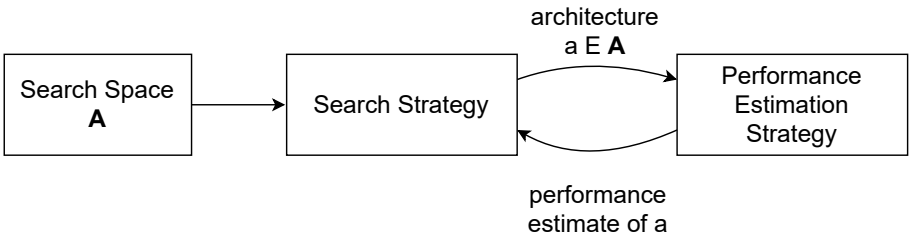
1.1 Motivation and hypothesis

Deep Learning (DL) has achieved great success in many applications (LECUN; BENGIO; HINTON, 2015; HUTTER; KOTTHOFF; VANSCHOREN, 2019) such as image analysis (SHEN; WU; SUK, 2017), speech recognition (HINTON et al., 2012), and text understanding (COLLOBERT; WESTON, 2008). DL can be performed in a supervised or unsupervised way and it is able to learn multi-level representations and features in hierarchical architectures for the tasks of classification and pattern recognition (ZHANG et al., 2018).There are several networks trained for different domains and that can be used as transfer learning, they are: VGG, ResNet, Inception, and others.

In this context, Neural Architecture Search (NAS) aims to automatically find the best neural network architecture for a network architecture for one or more specific learning tasks and datasets and promises to advance the field by removing human bias from architecture design. NAS can automatically suggest neural architectures that are comparable or outperform those manually designed (ZHONG et al., 2018a, 2018b; REAL et al., 2018).

Figure 1.1 illustrates the search process performed by NAS in which a search strategy selects an architecture *a* from a predefined search space *A*. The architecture is passed to a performance estimation strategy, which returns the estimated performance of *A* to the search strategy (ELSKEN; METZEN; HUTTER, 2019).

Figure 1.1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space *A*. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.



Source: (ELSKEN; METZEN; HUTTER, 2019)

Elsken, Metzen and Hutter (2019) categorize the methods applied for NAS according to three dimensions:

- *Search Space*: Defines the set of potential candidate network architectures to explore. To reduce the size of the search space, NAS takes advantage of prior knowledge about common properties of architectures that are suitable for specific tasks.
- *Search Strategy*: Guides the investigation through the search space, trying to find the best architecture by trading-off exploitation and exploration.
- *Performance Estimation Strategy*: Defines the process to estimate the quality of each candidate architecture.

Due to the extensive search space, NAS techniques suffer from high computational costs.

1.2 Hypothesis

We hypothesize that using combinations of architectures generated by NAS brings the possibility of using smaller search spaces and parallel searching, resulting in specific and different architectures. The advantages of this strategy are twofold. First, the reduced search space implicates in a reduced search runtime. Besides, combined (or ensemble) methods can improve classification results compared to individual models in a wide variety of applications (DIETTERICH, 1997).

1.3 Our proposal

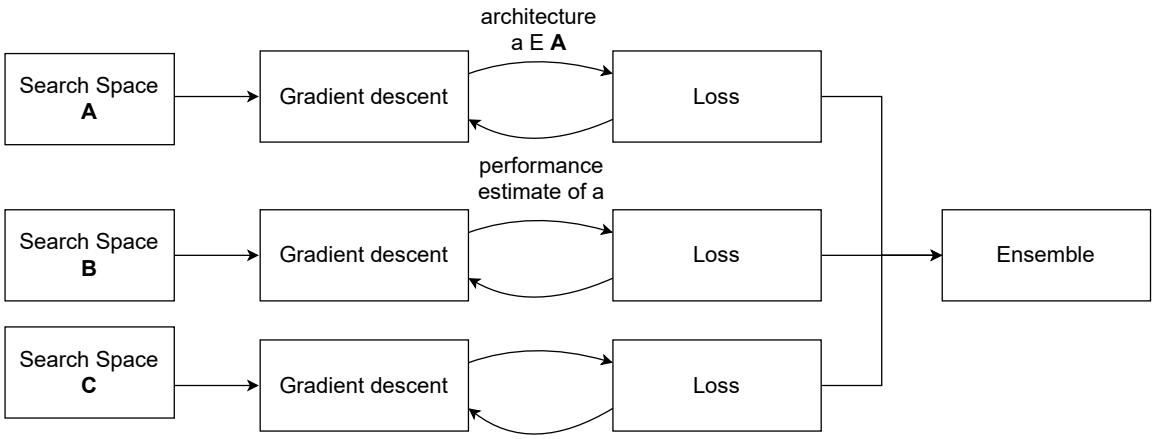
Combined neural networks have shown an improvement of accuracy in classification problems (TAO, 2019), showing that the possibility of a NAS looking for smaller networks in different search spaces can be a good search strategy. However, to ensure this improvement, the combined models need to make independent decisions (PONTI, 2011). In other words, the ensemble requires the individual classifiers to be diverse, which requires a few additional steps to ensure such variety.

In this scenario, we propose using an ensemble of neural networks obtained with NAS, referred to as Committee of NAS-based models (CoNAS). To ensure the diversity of neural architectures in CoNAS, we impose different search space limitations to the Differentiable ARchiTecture Search (DARTS) (LIU; SIMONYAN; YANG, 2018) by fixing different sets of search spaces.

Figure 1.2 illustrates how CoNAS works. From left to right, the search subspaces are triggered by the training set, and, from there, DARTS works in the conventional way to search

for the best network architectures restricted to each search space. In the end, the combination of the predictions of each neural network defines the final decision to a given input data.

Figure 1.2: Abstract illustration of CoNAS. A search strategy selects an architecture a from a predefined search space *A*. The architecture is passed to a performance estimation strategy, which returns the estimated performance of a to the search strategy. The same procedure is repeated for the search spaces B and C. Finally, when CoNAS need to predict the class of a new example, it combines the output of these three models in a single decision.



Source: Elaborated by the author

1.4 Main Contributions

We evaluated the committee of three architectures obtained by varying search sub-spaces with DARTS in two distinct scenarios. First, we trained for neural architectures from scratch using the dataset CIFAR-10. Later, we used the found architectures to classify images from the Imagenette dataset in a transfer-learning fashion. In both cases, we achieved slightly better accuracy rates. Besides, the combined search takes the same time as searching in the entire search space when done in parallel.

Thus, the main contributions of this work are:

- We propose and evaluate for the first time the combination between neural models found by DARTS, favoring the diversity between these models by limiting their search spaces with different restrictions.
- The proposed method achieved better classification performance than DARTS on CIFAR-10 data when trained from scratch. Moreover, we demonstrate the capacity of reusing discovered network architectures to perform transfer learning, obtaining an increase of accuracy on the Imagenette dataset.

- Through our experimental evaluation, we discuss possible paths for future developments that may bring meaningful advances in NAS research.

1.5 Dissertation organization

We started with the background and related work, explaining related works and subjects relevant to the proper understanding of this work, in Chapter 2. Chapter 3 presents the methodology, providing details of our proposal, showing how we divided the search space and how we performed the tests. Chapter 4 shows the results obtained with the proposed methodology. Finally, in Chapter 5, we present the final considerations, including a discussion on the main difficulties and future work.

Chapter 2

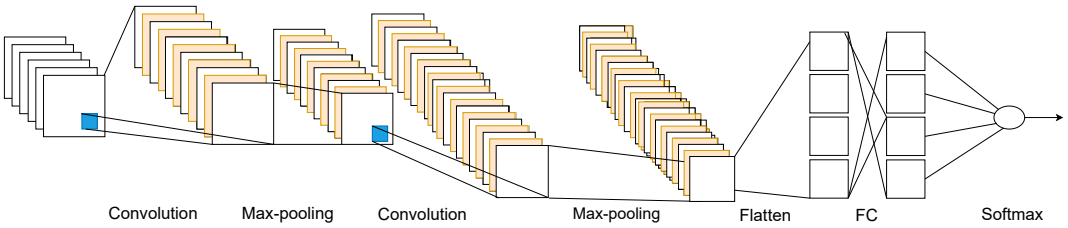
BACKGROUND AND RELATED WORK

This chapter is essential for the precise understanding of elements used in this work. This section presents basic concepts of artificial neural networks, focused on convolutional networks, which is the scope of this work. Since our proposal comprehends the ensemble of classifiers based on Neural Architecture Search, this chapter also presents the main concepts and related work on these topics. Some techniques are used in this work and others are used in other works.

2.1 Convolutional Neural Network

Convolutional Neural Network (CNN) has achieved groundbreaking results over the past decade in various fields related to pattern recognition; from image processing to voice recognition. The most beneficial aspect of CNNs is reducing the number of parameters in ANN. This achievement has prompted researchers and developers to build larger models to solve complex tasks, which was impossible with classic ANNs due to the vanishing gradient problem (ALBAWI; MOHAMMED; AL-ZAWI, 2017).In a face detection application, we do not need to pay attention to where the faces are located in the images. The only concern is to detect them regardless of their position in the given images (ALBAWI; MOHAMMED; AL-ZAWI, 2017). Another important aspect of CNN is that it obtains abstract features when propagating the deeper layers' input. For example, in image classification, the edge might be detected in the first layers, and then the simpler shapes in the second layers, and then the higher level features (ALBAWI; MOHAMMED; AL-ZAWI, 2017). Figure 2.1 illustrates a CNN.

Figure 2.1: An illustrative example of a Convolutional Neural Network. In this case, the network comprises two blocks of a convolutional layer with a dimensionality reduction operation (max-pooling), followed by flattening the representation to serve as input to two fully-connected layers (FC) and a decision/output layer, which uses a softmax function.



Source: Elaborated by the author

2.2 Regularization of parameters

Regularization has been used for decades before the advent of deep learning. Linear models, such as linear regression and logistics, allow simple, direct, and effective regularization strategies. This section will discuss some of the more popular approaches to regularization in deep learning models.

Many regularization approaches are based on limiting the capacity of the models, such as neural networks, linear regression or logistic regression, adding a $\Omega(\theta)$ penalty to the objective function J. We denote the objective function regularized as J $_R$:

$$J_r(\theta;X,y) = J(\theta;X,y) + \alpha\Omega(\theta) \tag{2.1}$$

Where $\alpha \in [0,\infty)$, is a hyper-parameter that directly contributes to the effect of the Ω function on J. Small values result in little regularization, larger values, in greater regularization.

Different choices for the Ω parameter function can result in different solutions. There are several options in the literature, for example, regularization L ^{1} and regularization L ^{2} (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.3 Dataset Augmentation

The best way to better generalize a machine learning model is to train it on more data (GOODFELLOW; BENGIO; COURVILLE, 2016). Obviously, in practice, the amount of data we have is limited. One way to get around this problem is to create fake data and add it to the training set. For some machine learning tasks, it is reasonably simple to create new false data. This approach is easier for classification. A classifier needs to take a complicated, high-dimension

entry x and summarize it with a single category y identity. This means that the main task that a classifier faces is to be invariable to a wide variety of transformations. We can generate new pairs (x, y) easily, just by transforming the x entries in our training set. This approach is not so easily applicable to many other tasks. For example, it is difficult to generate new false data for a density estimation task, unless we have already solved the density estimation problem. Increasing the data set has been a particularly effective technique for a specific classification problem: object recognition. The images are large in size and include a huge variety of factors of variation, many of which can be easily simulated. Operations such as translating some *pixels* of the training images in some direction can generally greatly improve generalization. Many other operations, such as rotating or scaling the image, have also proved to be quite effective.

Care must be taken not to apply transformations that would alter the class. For example, character recognition tasks require the recognition of the difference between 'b' and 'd' and the difference between '6' and '9'; therefore, horizontal inversions and 180º rotations are not suitable ways to increase the data sets for these tasks.

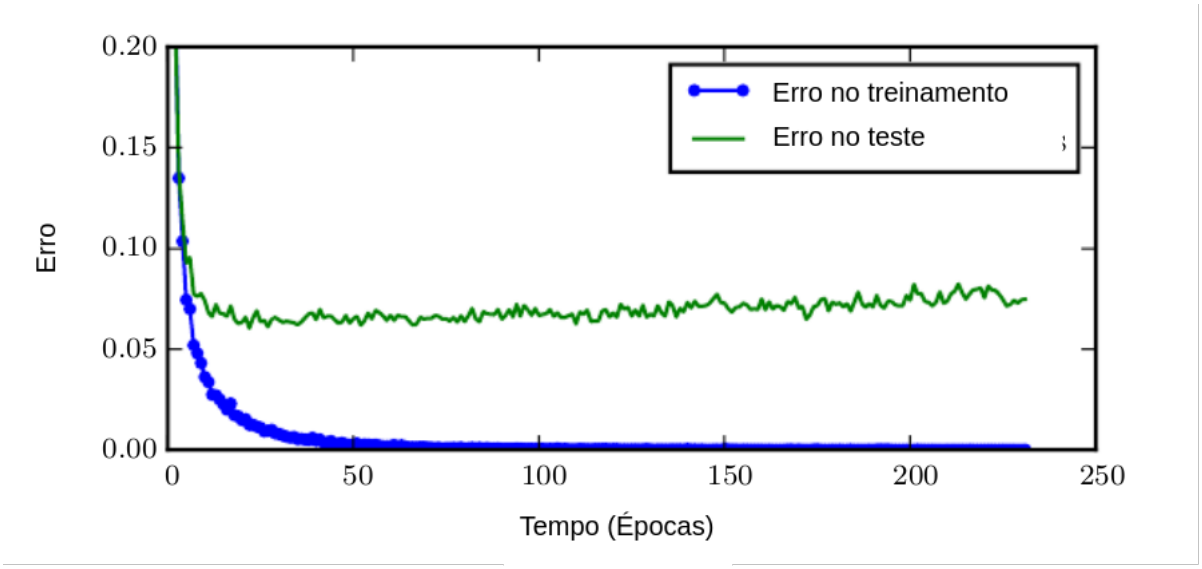
2.4 Early Stopping

When training large models with sufficient representational capacity to over-adjust the task, we generally observe that the training error decreases constantly over time, but the validation set error starts to increase. See Figure 2.2 for an example of this behavior. This behavior occurs very often. This means that we can get a model with a better error from the validation set (and therefore we expect it to have a better error from the test set) by returning to the parameter setting at the moment with the smallest error in the validation set. Whenever the error in the validation set improves, we store a copy of the model's parameters. When the training algorithm ends, we return these parameters, instead of the most recent parameters. The algorithm ends when no parameters have been improved on the best validation error recorded for some pre-specified number of iterations. This strategy is known as *Early Stopping*. It is probably the most used form of regularization in deep learning. Its popularity is due to both its effectiveness and its simplicity.

2.5 Momentum

Although the descent of the stochastic gradient remains a very popular optimization strategy, learning from it can sometimes be slow. The method *momentum* (POLYAK, 1964) it is

Figure 2.2: Learning curves showing how the loss changes over time (indicated as the number of training iterations in the data set). In this example, Goodfellow, Bengio and Courville (2016) trained a maxout network at MNIST. Observe that the training error decreases consistently over time, but the average loss of the validation set eventually begins to increase, forming an asymmetric curve.



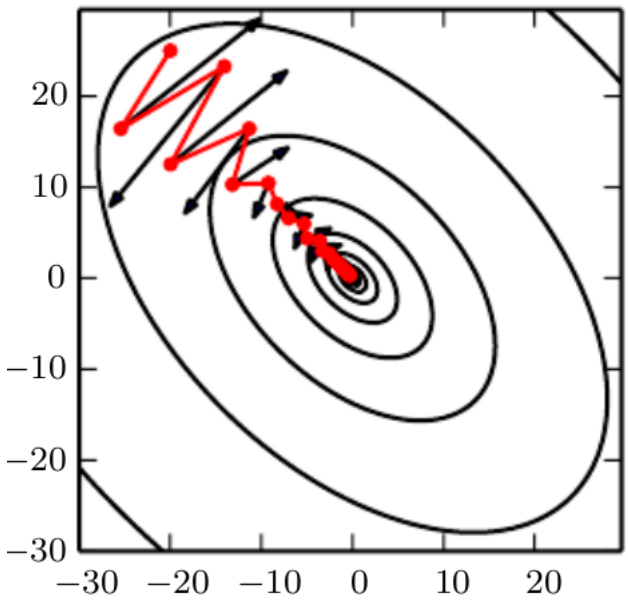
designed to accelerate learning, especially in the face of a high curve, small but consistent gradients or noisy gradients. The algorithm of *momentum* accumulates an exponentially decaying moving average from past gradients and continues to move towards it. The effect of *momentum* is illustrated in Figure 2.3.

Formally, the *momentum* technique introduces a variable v that plays the role of speed - it is the direction and speed at which parameters move through the parameter space. The speed is adjusted to an exponential decaying average of the negative gradient. The name *momentum* derives from a physical analogy, in which the negative gradient is a force that moves a particle through parametric space, according to Newton's laws of motion. The momentum of physics is mass times speed. In the learning algorithm with *momentum*, we assume the unit mass; therefore, the velocity vector v can also be considered the moment of the particle. A α *in* [0, 1) hyperparameter determines how quickly the contributions of the previous gradients decline exponentially. The update rule is provided by:

$$v \leftarrow \alpha v - \varepsilon \bigtriangledown \theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{i})) \right), \tag{2.2}$$

$$\theta \leftarrow \theta + v \tag{2.3}$$

Figure 2.3: The red path indicates the path followed by the learning rule with textit momentum. At each step along the way, Goodfellow, Bengio and Courville (2016) draws an arrow indicating the step that the gradient descent would take at that point. We can see that a quadratic objective looks like a long, narrow valley or canyon with steep sides. The *momentum* correctly crosses the canyon longitudinally, while the gradient steps waste time moving back and forth through the narrow canyon axis.



The higher the ratio of α to ε, the more previous gradients affect the current direction.

2.6 Reinforcement learning

Reinforcement learning it is learning what to do how to map actions in order to maximize a numerical reward signal. The model is not informed about which actions to perform, but it must find out which actions generate the most reward by trying them. In the most interesting and challenging cases, actions can affect not only the immediate reward, but also the next situation and, with that, all subsequent rewards (SUTTON; BARTO, 2018).

The basic idea is to capture the most important aspects of the real problem that a learning agent faces, interacting over time with their environment to achieve a goal. A learning agent must be able to sense the state of his environment to some extent and must be able to perform actions that affect the state. The agent must also have one or more goals related to the state of the environment. Any method that is well suited to solving these problems can be considered a reinforcement learning method.

One of the challenges that arise in reinforcement learning is the treatment of *exploration* and *exploitation*. In order to get a lot of reward, a reinforcement learning agent must prefer actions that they have tried in the past and that are effective in producing reward. But, to discover these actions, he must try actions that he had not selected before. The agent needs to explore (*exploration*) what he has already experienced (*exploitation*) to get a reward, but he also needs to explore to make better action selections in the future. The dilemma is that neither exploration nor experimentation can be pursued exclusively without failing in the task. The agent must try a variety of actions and progressively favor those that appear to be the best. In a stochastic task, each action must be tried several times to obtain a reliable estimate of its expected reward. The exploration-experimentation dilemma has been intensively studied by mathematicians for many decades, but remains unsolved.

2.7 Adaptation of the learning rate

Researchers of neural networks have long realized that the learning rate is one of the most difficult hyper-parameters to define, as it has a significant impact on the performance of the model. The cost is generally highly sensitive to some directions in the parameter space and insensitive to others. The *momentum* algorithm can mitigate these problems a little, but it does so at the expense of introducing another hyper-parameter.

The *delta-bar-delta* (JACOBS, 1988) algorithm is a heuristic approach for adapting individual learning rates to model parameters during training. The approach is based on a simple idea: if the partial derivative of the loss, with respect to a given model parameter, remains the same sign, the learning rate should increase. If the partial derivative in relation to this parameter changes sign, the learning rate should decrease (GOODFELLOW; BENGIO; COURVILLE, 2016).

More recently, several incremental methods (or based on *mini-batchs*) have been introduced that adapt the learning rates of the model parameters. Methods like *AdaGrad* (DUCHI; BARTLETT; WAINWRIGHT, 2012), *RMSProp* (Unpublished adaptive learning rate method proposed by Geoff Hinton in Class 6e of his course in Coursera[1]), *Adam* (KINGMA; BA, 2014) *Adadelta* (ZEILER, 2012), *AdaMax* (KINGMA; BA, 2015) and *Nadam* (DOZAT, 2016) are widely used and compared in the literature (RUDER, 2016).

[1] $http://www.cs.toronto.edu/\ tijmen/csc321/slides/lecture_slides_lec6.pdf$

2.8 Gradient descent optimization

The most popular deep learning techniques involve some optimization algorithm. The optimization itself can be seen as a minimization or maximization of a function **f (x)**, changing the form of **x**.

Commonly, the function to be maximized or minimized is called an objective function. In the figure 2.4, we have a function $y = f(x)$, where x and y are real numbers. The derivative of this function is denoted as $f'(x)$ or as $\frac{dy}{dx}$. The derivative $f'(x)$ gives the slope of $f(x)$ at point x. In other words, it specifies how to scale a small change in the input to obtain the corresponding change in the output. Therefore, the derivative is useful for minimizing a function because it tells us how to change x to make a small improvement on y. Thus, we can reduce $f(x)$ by moving x in small steps as a derivative function. This technique is called gradient descent (CAUCHY, 1847).

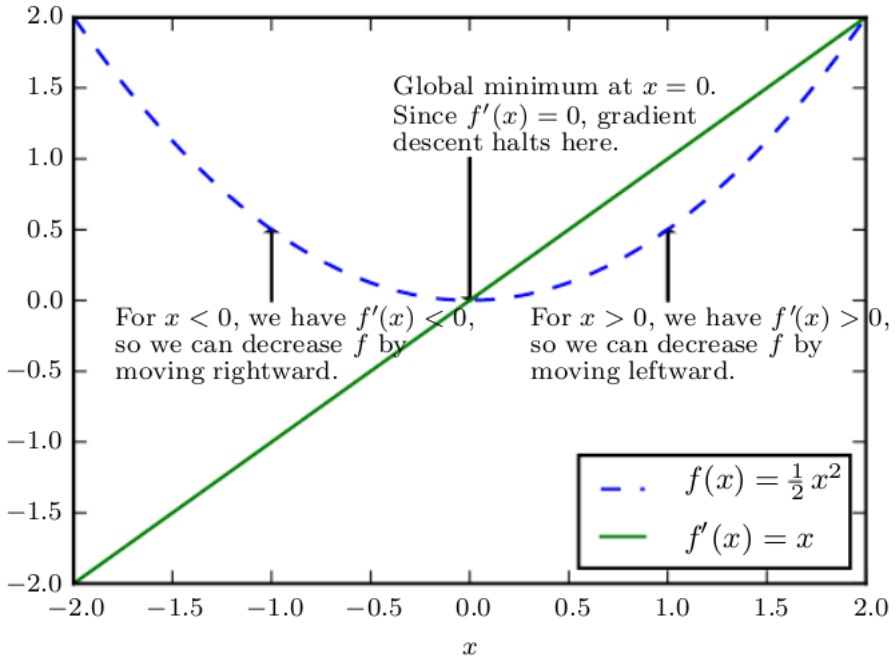
When we use a feed-forward neural network for an input **x** and produce an output $\hat{\mathbf{y}}$, information flows forward through the network. The **x** entries provide the initial information that is then propagated to the hidden units in each layer and finally produces $\hat{\mathbf{y}}$. This is called *forward propagation*. During training, *forward propagation* can continue forward until it produces an L error (θ). The (RUMELHART; HINTON; WILLIAMS, 1986) back-propagation algorithm allows cost information to flow backward through the network to calculate the gradient and stimulate the network to obtain a lower error value, causing the graph of the error function to behave similarly to Figure 2.4, on the left.

The term back-propagation is often misinterpreted as meaning the entire learning algorithm for multilayered neural networks. Back-propagation refers only to calculating the gradient, while another algorithm, such as *stochastic gradient descent*, is used to perform the learning (GOODFELLOW; BENGIO; COURVILLE, 2016), as illustrated in Algorithm 1 and Figure 2.5.

Algorithm 1: Stochastic gradient descent

Entry: A training set with m examples, regularization parameters, η learning rate.
while *evaluation criteria not met* **do**
  for *(x,y)* $\in$ *m* **do**
    calculates Z, A, J na propagation
    calculates dA, dZ, db na backpropagation with J(W,b,x,y)
    updates the weights W, b in the backpropagation with dA,dZ,db
  end
end

Figure 2.4: An illustration of how the gradient descent algorithm uses the derivatives of a function to search for the minimum.



Source: Elaborated by the author

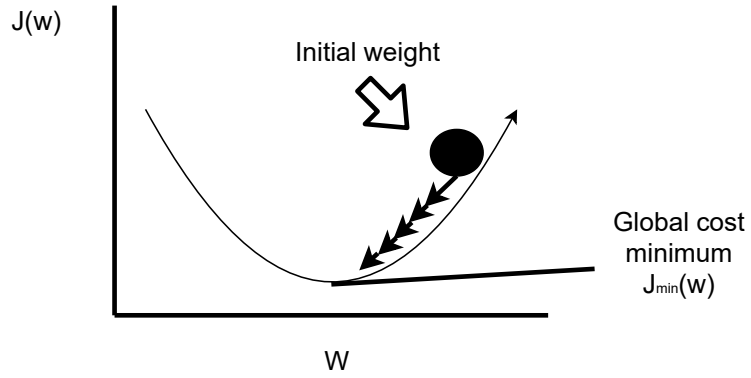
2.9 Optimization techniques

Of all the many optimization problems associated with deep learning, the most difficult is training the neural networks (GOODFELLOW; BENGIO; COURVILLE, 2016). It is pretty common to invest days or months in hundreds of machines to solve a single machine learning problem with a neural network. As this problem is so significant and expensive, a specialized set of optimization techniques has been developed to solve it. This section presents some of the main optimization techniques for training neural networks.

2.10 Mini Batch GD

Compare two hypothetical gradient estimates, one based on 100 examples and the other based on 10,000 examples. The latter has a computation cost 100 times greater than the former, reducing the standard error of the mean by only a factor of 10. Most optimization algorithms converge much more quickly (in terms of computation cost) if they can quickly calculate approximate estimates of the gradient instead of slowly calculating the exact gradient.

Figure 2.5: Illustration of a gradient drop in a cost function L (w).



Source: Elaborated by the author

Another consideration that motivates the estimation of the gradient from a small number of samples is the redundancy in the training set. At worst, all samples in the training set can be identical copies of each other. A sampling-based gradient estimate could calculate the correct gradient with a single sample, using *m* times less computation than an all-sample approach. In practice, it is unlikely that we will encounter this worst-case situation, but we can find many examples that make very similar contributions to the gradient.

Typically, the term *batch gradient descent* implies using the entire training set while using the term *batch* to describe a group of examples. For example, it is very common to use the term size of *bath* to describe the size of a *minibatch*, that tries to capture the advantages of both, which is the minibatch (uses batches that do not imply the entire dataset, but a subset. Optimization algorithms that use only a single example at a time are called stochastic methods (*stochastic*). The classic example of a stochastic method is *stochastic gradient descent*, presented in the section 2.8.

2.11 Transfer learning

Transfer learning (TL) refers to the situation where what has been learned in one configuration (ie, P1 distribution) is exploited to improve generalization in another configuration (for example, a P2 distribution) (GOODFELLOW; BENGIO; COURVILLE, 2016).

In transfer learning, the model must perform two or more different tasks (GOODFELLOW; BENGIO; COURVILLE, 2016), but we assume that many of the factors that explain the variations in P1 are relevant to the variations that need to be captured for learning in P2. This is generally understood in a supervised learning context, where the input is the same, but the target may be of a different nature. For example, we can learn about a set of visual categories, such as

dogs and cats, in the first configuration, and learn about a different set of visual categories, such as ants and wasps, in the second configuration. If there is significantly more data in the first configuration (sampled in P1), this can help to learn useful representations to generalize quickly with just a few examples taken from P2.

2.12 SMBO

Sequential Model-Based Optimization (SMBO) (HUTTER; HOOS; LEYTON-BROWN, 2011a) is an optimization *framework* that can work explicitly with categorical and continuous hyper-parameters. You can explore conditional spaces, such as when a hyper-parameter is relevant only if another hyper-parameter (or some combination hyperparameters) assumes a certain value (KOTTHOFF et al., 2019). SMBO uses a substitute (supplementary) model to assess fitness, which is very useful when assessing fitness function is costly. It uses objective functions, such as the Gaussian Process, to select good values for hyper-parameters and then update the values sequentially based on the results (NAGARAJAH; PORAVI, 2019).

A common approach to apply SMBO is by *Bayesian Optimization* (BO) (BROCHU; CORA; FREITAS, 2009). Bayesian optimization iteratively adjusts a model to the observations of the target function. It then uses an acquisition function to determine the next candidate points based on the predictive distribution of the model. With the increase in observation numbers, the predictive distribution of the model improves. Consequently, the algorithm better assumes which regions of the parameter space are worth exploring and which are not.

2.13 Neural Architecture Search

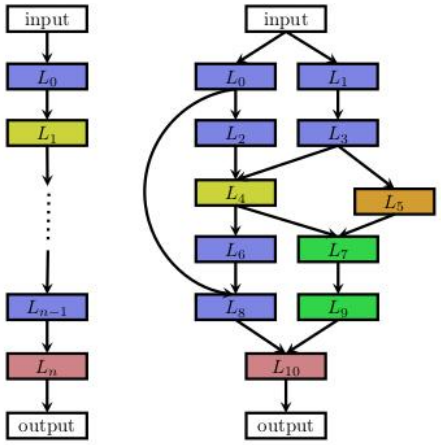
Currently, NAS methods can successfully handle some of the deep learning tasks such as image classification, object detection (ZOPH et al., 2018; LIU; SIMONYAN; YANG, 2018), and semantic segmentation (CHEN et al., 2018). NAS has significant overlap with transfer learning (ZOPH; LE, 2016), for example, DARTS uses transfer learning on the network generated in CIFAR-10 for the imagenet.

Next, we present fundamental concepts to define NAS techniques.

2.13.1 Search Space

In the context of a *convolutional neural network* (CNN), the search space is parameterized by (i) the maximum number of layers n; (ii) the type of operation every layer executes, e.g., convolution, pooling, or more advanced operations like depthwise separable convolutions (Chollet, 2017); and (iii) hyperparameters regarding operations, e.g., number of filters, kernel size and strides for a convolutional layer (BAKER et al., 2017), or only the number of units for fully-connected networks, activation functions and regularization methods (MENDOZA et al., 2016). Figure 2.6 illustrates a simple search space of chain-structured neural networks.

Figure 2.6: An illustration of different architecture spaces .



Source: (ELSKEN; METZEN; HUTTER, 2019)

In this figure, each node in the graphs corresponds to a layer in a neural network, e.g., a convolutional or pooling layer. Different layer types are visualized by different colors. An edge from layer L_i to layer L_j denotes that L_j receives the output of L_i as input. In the left it is illustrated an element of a chain-structured space; and in the right we can see a more complex search space with additional layer types and multiple branches and skip connections (ELSKEN; METZEN; HUTTER, 2019).

Many NAS approaches require a large number of layers to achieve good performance. Furthermore, in many approaches, each of the neural networks has to be trained from scratch taking a very long time to output the result (JIN; SONG; HU, 2018).

2.13.2 Search Strategy

Many different search strategies can be used to explore the search space of neural architectures including Random Search (RS), Bayesian Optimization (BO), Evolutionary Methods (EM), Reinforcement learning (RL) and methods based on Gradient Descent (GD). The main search strategies are summarized in Table 2.1.

Table 2.1: Overview of different methods to search network architectures in a NAS search space.

Search Method	Results
Random Search	Randomly selects network architecture settings (LI; TALWALKAR, 2019).
Reinforcement Learning	Highly competitive results with humans in CIFAR-10 and Treebank. (ZOPH et al., 2018)
Evolutionary Algorithms	Performance similar to **reinforcement learning**, however, generating smaller models. (REAL et al., 2019)
Bayesian Optimization	Some results of this approach may also overcome the **evolutionary algorithms** (KLEIN et al., 2017)
Gradient Descent	Four hours of GPU on CIFAR-10 with competitive results compared to **reinforcement learning** and **evolutionary algorithms**. (DONG; YANG, 2019)

NAS became one of the top research topics in the machine learning community after Zoph Zoph et al. (2018) achieved competitive performance on the CIFAR-10 and Penn Treebank (MARCUS; SANTORINI; MARCINKIEWICZ, 1993) benchmarks with a search strategy based on Zoph et al. (2018) used huge computational power to achieve this result (800 GPUs for four weeks). After their work, a wide variety of methods were published to reduce computational costs and obtain additional performance improvements.

Bayesian Optimization (BO) achieved several early successes on NAS by suggesting high-level architectures (BERGSTRA; YAMINS; COX, 2013). NAS algorithms using some search strategies obtained competitive architectures for the CIFAR-10 and generated the first automatically adjusted neural networks to win some competitions against human experts.

BO (Shahriari et al., 2016) is one of the most popular methods for optimizing hyperparameters, but it has been applied to NAS only in a few works, since typical BO toolboxes are based on Gaussian processes and focus on low-dimensional continuous optimization problems. On the other hand, several works use tree-based models (in particular, Parzen (BERGSTRA; YAMINS;

COX, 2013) tree estimators or random forests (HUTTER; HOOS; LEYTON-BROWN, 2011b)) to effectively research high-dimensional conditional spaces and obtain cutting-edge performance over a wide range of problems, optimizing neural architectures and their hyperparameters together (BERGSTRA; YAMINS; COX, 2013).

To use NAS as a reinforcement learning (RL) problem (ZOPH et al., 2018; ZHONG et al., 2018a), the generation of a neural architecture can be considered the action of the agent of RL. The agent's reward is based on an estimate of the performance of the architecture trained on unseen data. RL approaches differ in how they represent the agent's policy and how they optimize it.

An alternative to RL is the neuro-evolutionary approaches, which use evolutionary algorithms to optimize the neural architecture. For the best of our knowledge, the first neuro-evolutionary approach was proposed at least three decades ago (1989) by (MILLER; TODD; HEGDE, 1989) and their genetic algorithms applied to propose architectures with backpropagation and to optimize their weights. Many neuro-evolutionary approaches have used genetic algorithms to optimize neural architecture and its weights since then; however, when scaling to contemporary neural architectures with millions of weights for supervised learning tasks, weight optimization methods based on Stochastic Gradient Descent (SGD) currently surpass evolutionary ones (ELSKEN; METZEN; HUTTER, 2019).

The newest neuro-evolutionary approaches (REAL et al., 2017, 2019; ELSKEN; METZEN; HUTTER, 2019; LIU et al., 2018) use gradient-based methods to optimize weights and evolutionary algorithms only to optimize the neural architecture. Evolutionary algorithms give rise to a population of models, that is, a set of networks; and at each stage of evolution at least one model is picked up from the population to generate children by applying mutations on it. In the context of NAS, mutations are local operations, such as adding or removing a layer, changing the hyper-parameters of a layer, adding skip-connections and changing the training hyper-parameters. After training the children, their performance (for example, in a validation set) is assessed and they only the bests ones are added to the population.

Neuro-evolutionary methods differ in the way they select architectures, update populations and generate children. For example, (REAL et al., 2018, 2017; LIU et al., 2018) use *tournament selection* to select parents, while (ELSKEN; METZEN; HUTTER, 2019) selects parents with *multi-objective Pareto*. (REAL et al., 2018) removes the worst individual from a population, while (REAL et al., 2019) considered removing the oldest individual and (LIU et al., 2018) does not remove individuals. To generate children, most approaches initialize a population at random, while (ELSKEN; METZEN; HUTTER, 2019) employs *Lamarckian inheritance* where the knowl-

edge (in the form of learned weights) is passed from a parent network to its children using morphisms in the network. (REAL et al., 2018) also allows a child to inherit (from their parents) all parameters that are not affected by the applied mutation; even if this inheritance does not strictly preserve the architecture, it can speed up learning compared to random startup.

Another possible approach to research network architectures is by using gradient descent in neural network architectures (LIU; SIMONYAN; YANG, 2018; DONG; YANG, 2019). (LIU; SIMONYAN; YANG, 2018) shows that the search for network architecture through gradient descent achieves highly competitive results in CIFAR-10, in addition to surpassing the state of the art in Penn Treebank (PTB). This is a very interesting result, considering that until now the best architectural research methods used non-differentiable research techniques, for example, based on RL (ZOPH et al., 2018) or neuro-evolution (REAL et al., 2018; LIU et al., 2018).

Another proposal is a NAS with gradient descent (DONG; YANG, 2019). Such an approach represents the search space as a directed acyclic graph (DAG). This DAG can contains billions of subgraphs, each one representing a type of neural architecture. The results are surprising: the NAS completed a search procedure in four hours of GPU on the CIFAR-10, obtaining an error in the test set of only 2.82% with around 2.5 million parameters.

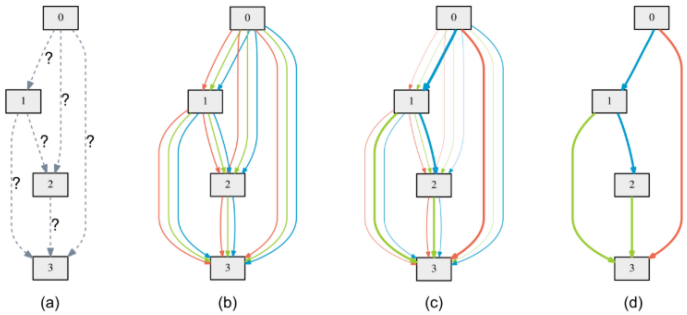
2.13.3 Performance Estimation Strategy

The search strategies presented in section 2.13.2 generate network architectures that need to be evaluated somehow to conduct the exploration through the search space. The simplest way to accomplish that is to integrally perform the training stage and then evaluate the generated architecture with the test data. However, considering the order of magnitude $O(nt)$(JIN; SONG; HU, 2018), where n is the number of networks to be evaluated during the search and t is the average time spent to evaluate each of these n architectures, performing the training stage leads to a high computational and time cost, spending several days in a GPU (ZOPH; LE, 2016; REAL et al., 2018; ZOPH et al., 2018; REAL et al., 2017).

2.13.4 DARTS

DARTS (Differentiable ARchiTecture Search) is one of most popular NAS techniques. The new paradigm proposed by (LIU; SIMONYAN; YANG, 2018) uses gradient descent to search for a computation cell as the building block of the final architecture. To use gradient descent, Liu, Simonyan and Yang (2018) creates a continuous search space by relaxing the original search space for possible operations with softmax:

Figure 2.7: An overview of DARTS



Source: (LIU; SIMONYAN; YANG, 2018)

$$o^{-(i,j)}(x) = \sum_{o \varepsilon O} \frac{exp(\alpha_o^{(i,j)})}{\sum_{o' \varepsilon O} exp(\alpha_{o'}^{(i,j)})} o(x) \tag{2.4}$$

.

Darts starts the search with a complete network loaded with operations, as illustrated in Figure 2.7. The set of possible operations on each neural network node is individually parameterized by the weights represented by α. The gradient descent allows us to select the operations contributing to a lower loss. This way, after a fixed number of epochs (50 by default), we will have the chosen architecture.

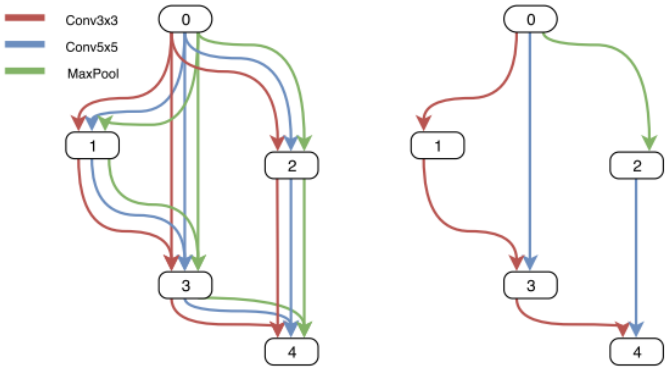
DARTS can learn high-performance architecture building blocks with complex graph topologies in a rich search space with 10^{18} architectures.

The following definitions are assumed to define the architectures explored by DARTS. Each directed edge (i, j) is associated with some operation $o(i, j)$ that transforms $x(i)$. Each cell has two input nodes and a single output node. For convolutional cells, the input nodes are defined as the cell outputs in the previous two layers(ZOPH et al., 2018), where $x(i)$ is a latent representation (e.g. a feature map in convolutional networks).

Figure 2.4 shows an overview of DARTS: : (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bi-level optimization problem. (d) Inducing the final architecture from the learned mixing probabilities (LIU; SIMONYAN; YANG, 2018).

Illustration of one-shot architecture search in Figure 2.8. Simple network with an input node (denoted as 0), three hidden nodes (denoted as 1,2,3) and one output node (denoted as 4). Instead of applying a single operation (such as a 3x3 convolution) to a node, the one-shot model (left) contains several candidate operations for every node, namely 3x3 convolution (red

Figure 2.8: An overview of DARTS



Source: (LIU; SIMONYAN; YANG, 2018)

edges), 5x5 convolution (blue edges) and MaxPooling (green edges) in the above illustration. Once the one-shot model is trained, its weights are shared across different architectures, which are simply subgraphs of the one-shot model (right). Figure inspired by (LIU; SIMONYAN; YANG, 2018).

Liu, Simonyan and Yang (2018) conducts NAS experiments on CIFAR-10 and PTB and transferable architectures to ImageNet and WikiText-2, respectively.

In CIFAR-10, Liu, Simonyan and Yang (2018) include the following operations: 3×3, 5×5 and 7 x 7 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and zero. All operations are of stride one. Liu, Simonyan and Yang (2018) hold out half of the CIFAR-10 training data as the validation set. A small network of 8 cells is trained using DARTS for 50 epochs, with batch size 64 and the initial number of channels 16. The momentum SGD are used to optimize the weights w, with initial learning rate $\eta w = 0.025$.

Unfortunately, researchers have pointed out that DARTS suffers from overfitting if executed for too many epochs. For this reason, we can find proposals to improve the chances of DARTS finding architectures that generalize better, such as applying early stopping (LIANG et al., 2019).

2.14 Classifiers ensemble

The term ensemble is used to identify a set of predictors whose individual decisions are combined or aggregated in some way to predict new examples (DIETTERICH, 1997). This article focuses only on classification problems. However, the combination of techniques can be used for regression problems with only minor cost function changes.

According to the no-free lunch theorem (Wolpert; Macready, 1997), there is no optimal algorithm for all decision problems. For this reason, ensemble techniques betake the decisions of distinct classifiers and aggregate them in a single decision, like a decision made by a committee. Thereby, even if a small number of models make a mistake, the combined answer can still be correct..

The most straightforward technique to aggregate individual classifiers decisions is by majority voting. In this case, the final decision is for the class label with the individual highest number of votes. A model "votes" for a class label k if its output is k. Formally, the majority voting is defined by Equation 2.5.

$$argmax_{k \in C}(\sum_{i=1}^{m} \phi(i,k)) \tag{2.5}$$

where C is the set of class labels to assign to an example, m is the number of individual models, and $\phi(i,k)$ is defined by Equation 2.6.

$$\phi(i,k) = \begin{cases} 1, \text{ if } k = argmax_{j \in C} P_{ij} \\ 0, \text{ otherwise} \end{cases} \tag{2.6}$$

where P_{ij} is the output score of the i-th model to the j-th label in C.

Although simple, the majority voting disregards the confidence of each classifier in its decision. For instance, consider a scenario with three individual binary classification models. Besides, consider the following obtained scores for a given instance: $P_{10} = 0.51$, $P_{11} = 0.49$, $P_{20} = 0.51$, $P_{21} = 0.49$, $P_{30} = 0.01$, $P_{31} = 0.99$. The committee would decide for the class 0, despite the models who voted for this class have low confidence in their decisions.

An alternative to avoid this limitation of majority voting is considering the models output scores when aggregating their votes in a weighted voting manner. The ensemble technique uses a simple function of each class's scores, S_k, and decides for that with the highest obtained value. Two examples of aggregation rules are the sum (Equation 2.7) and product (Equation 2.8).

$$S_k = \sum_{i=1}^{m} P_{ik} \tag{2.7}$$

$$S_k = \prod_{i=1}^{m} P_{ik} \tag{2.8}$$

Kittler (2005) concludes that the sum rule is the most conservative and most used. The product may have superior results but they are risky.

A good observation regarding the ensemble is that combining similar models is useless. The variability of characteristics and domain spaces are crucial for good ensemble results. Thus, the models to be combined must have a certain level of disagreement.

Chapter 3

METHODOLOGY

In this chapter, we present the methodology, including details of our proposal and experimental decisions. We recall that the main idea behind our proposal was presented in Figure 1.2. Then, we describe how we define the search sub-spaces to compose the ensemble. We also present the training and test strategies. Due to cost constraints, we train from scratch only in one dataset. Then, we use transfer learning to evaluate the architectures when applied to other data.

3.1 Our Proposal

Our proposal consists of creating a Comitee of NAS-based models (CoNAS). CoNAS is based on the diversity of search spaces, due to the need for diversity between the models to be combined. For this, we defined 3 search darts sub-spaces. The 3 sub-spaces have the following operations:

- **Space A**: 3 × 3, 5 × 5 separable convolutions, 3 × 3 and 5 × 5 dilated separable convolutions, 3 × 3 max pooling, 3 × 3 average pooling, identity, and zero.
- **Space B**: 5 × 5 and 7 x 7 separable convolutions, 5 × 5 dilated separable convolutions, 3 × 3 max pooling, 3 × 3 average pooling, identity, and zero.
- **Space C**: 3 × 3 and 7 x 7 separable convolutions, 3 × 3 dilated separable convolutions, 3 × 3 max pooling, 3 × 3 average pooling, identity, and zero.

Although we set specific parameters, our proposal is a framework. Therefore, changing fundamental elements of the algorithm is straightforward. We implemented our proposal so

that these changes are made by execution parameters like learning rate, epochs, dir of logs, and others.

This work will use the sum of scores to combine the models. The first reason for that choice is because DARTS convolutional networks return probabilistic values. Besides, the sum rule technique is the most conservative and widely applied.

Several previous works use an ensemble strategy based on neural networks of the most diverse types (Krishnakumar; Williamson, 2019; Nagahamulla; Ratnayake; Ratnaweera, 2016; Marino; Virupakshappa; Oruklu, 2020; Abouelnaga et al., 2016; Yang et al., 2013; Rijal et al., 2018). The most common approach found in these works is the use of different subsets of data for training each network architecture and, then, making up the combination. In this work, we use an alternative technique, changing architectures instead of data, using DARTS.

3.2 Training step

The model is trained in cifar-10 and then the resulting network architecture is transferred to the other datasets to be trained from scratch, with weight sharing.

For a better comparison and reproducibility, data sets applied to evaluate state-of-the-art network architecture algorithms will be used, they are:

CIFAR-10 (KRIZHEVSKY, 2009), which consists of 50,000 training images and 10,000 test images. A total of 60,000 RGB images, 32x32 pixels, with 10 classes (for example, airplanes, cars, animals) and perfectly balanced, that is, with 6,000 images belonging to each class.

ImageNette (HOWARD, 2019), a state of the art reference for Imagenet *subsets*. It contains 10 easily classifiable classes from Imagnet. As this dataset does not have a test set, we use the evaluation set for the final evaluation.

Intel Dataset [1] Created by Intel for an image classification contest, this expansive image dataset contains approximately 25,000 images. Furthermore, the images are divided into the following categories: buildings, forest, glacier, mountain, sea, and street. The dataset has been divided into folders for training, testing, and prediction. The training folder includes around 14,000 images and the testing folder has around 3,000 images.

Cellulas dataset [2] This data comes from the Recursion 2019 challenge. This goal of the competition was to use biological microscopy data to develop a model that identifies replicates.

[1] https://www.kaggle.com/puneet6060/intel-image-classification

[2] https://www.kaggle.com/xhlulu/recursion-cellular-image-classification-224-jpg

3.3 Test step

Samples not used in training will be pre-processed in the same way as in the previous phase. Then, for each architectural search experiment, the samples will serve as input for the models generated by the architectural search. Thus, having an estimate of evaluation of the generated architectures. Soon after, the test data are presented to the combination of the generated models (through the sum of the liabilities), having at the end, the final evaluation of the combination of the generated networks.

This work will use the sum of the scores to combine the models. The first reason for this choice is because conventional DARTS networks return probabilistic values. In addition, the sum rule technique is the most conservative and widely applied.

3.4 Work Development Methodology

All the necessary resources for the execution of this project are available at the Federal University of São Carlos - campus São Carlos.The institution has a network of workstations and personal computers available for the project, in addition to the necessary programs for the good development of the project, which are installed and are, in the majority, in the public domain. The institution also has a server with a dedicated graphics card and several CPU cores (Intel Core i9-7900X with 20 cores, which operated at 3.3 GHz with 15 MB of cache, 128 GB ram and a Nvidia RTX GPU. 2080 with 12 GB of DDR5 memory). The entire bibliographic reference is available in the library and on the Internet through the institution's contracts with the repositories of articles and academic works.

The proposed methodology to achieve the objective of this project is described below, in the following phases:

- The developed algorithm is based on the implementation of DARTS (LIU; SIMONYAN; YANG, 2018).
- In the pre-processing stage, the state-of-the-art training methodology will be used for a better comparison of the results obtained.
- In the evaluation step, the test methodology present in the DARTS algorithm was used, obtaining the error present in the architecture, the number of parameters present in the architecture and the time spent in GPU processing.

- Finally, the models are combined in the test stage by adding the probabilities of each model evaluated in the test set.

Chapter 4

EXPERIMENTAL EVALUATION

Here we present the results obtained with the training of the models. The Intel model was the one that had the greatest accuracy. We present graphs and figures for a better understanding of this work.

Liu, Simonyan and Yang (2018) determines the architecture for final evaluation by running DARTS four times with different random seeds and pick the best cell based on its validation performance obtained by training from scratch for a short period (100 epochs on CIFAR-10).

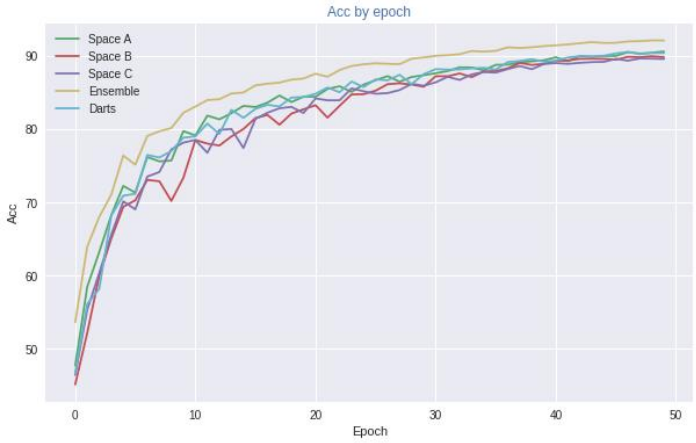
Our experiments[1] run DARTS a single time for each specific subspace, with the same seeds, and pick the best cell based on its validation performance to determine the final evaluation architecture. We run the DARTS algorithm in each search space and generate the models to combine them.

Figure 4.1 shows how the accuracy obtained by DARTS, with complete and the three proposed reduced search spaces, and CoNAS evolves through the training epochs on CIFAR-10. All experiments used the default cutout from DARTS.

The results show a difference between the models generated in different search spaces. Specifically, we can note that the accuracy decreases for a model in some epochs while it increases to another one. It seems a piece of evidence that these models are making independent decisions. This difference may have contributed to a superior result in the combination. The combination results were superior to the individual search spaces and the full DARTS search space during the entire process.

[1]The experiments were performed on a computer with Intel (R) Core (TM) i9-7900X CPU @ 3.30GHz, 128 GB of RAM, and a single NVIDIA 2080 TI 12GB RAM GPU.

Figure 4.1: Accuracy for each search spaces on CIFAR-10.



Source: Elaborated by the author

Table 4.1 shows the test set's accuracy after 50 training epochs on CIFAR-10. We note that the ensemble of DART sub-spaces outperforms the complete DARTS by approximately 1.72%. The accuracy rates of the subspace models shows a small difference between them.

Table 4.1: Experimental results on CIFAR-10. Accuracy, number of parameters and GPU cost.

Architecture	ACC	Params (M)	GPU (Hours)
DARTS	90.36%	1.93	6
DARTS space A	90.58%	1.93	6
DARTS space B	89.80%	0.92	4
DARTS space C	89.56%	1.14	4
DARTS ensemble	**92.08%**	**3.99**	**14***

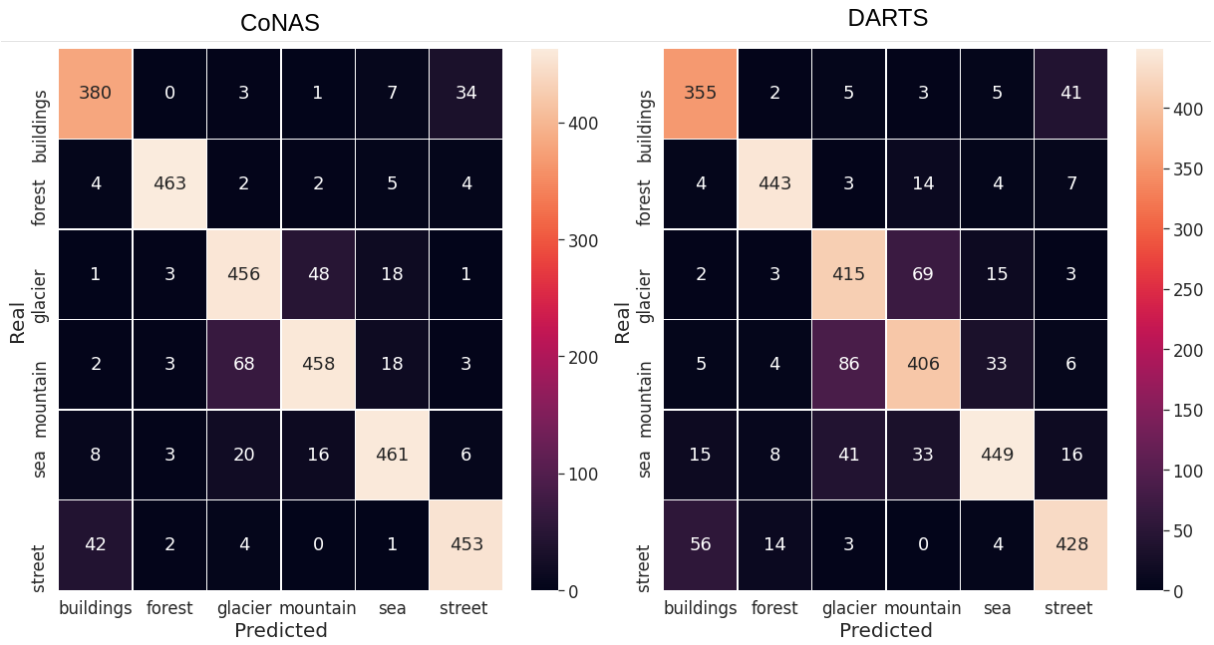
* This time considers we run each search space sequentially.

It is worth mentioning that we executed the DARTS ensemble sequentially (one at a time in the GPU). However, CoNAS is embarrassingly parallel. If we execute the base models simultaneously, we will obtain a runtime similar to the maximum runtime for a single model. In this case, approximately six hours.

Figure 4.2 presents the confusion matrix obtained by our method and DARTS using the complete search space.

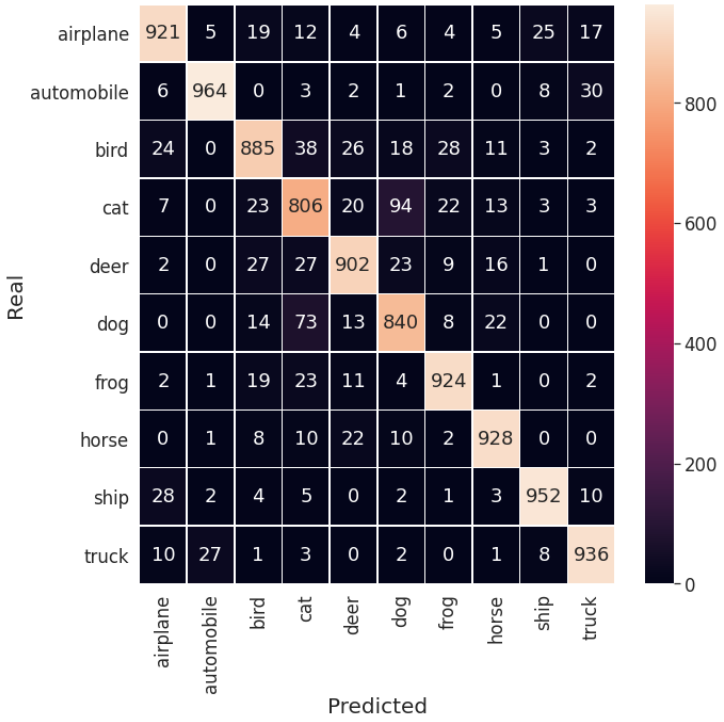
The ensemble proves to be more efficient than DARTS with the complete search space. The results show a substantial increase in the number of correctly classified examples in almost all classes of CIFAR-10. The only exception is the class "cat", where CoNAS mislabeled two examples more than DARTS.

Figure 4.2: Confusion matrix obtained by CoNAS (left) and DARTS (right) on CIFAR-10.



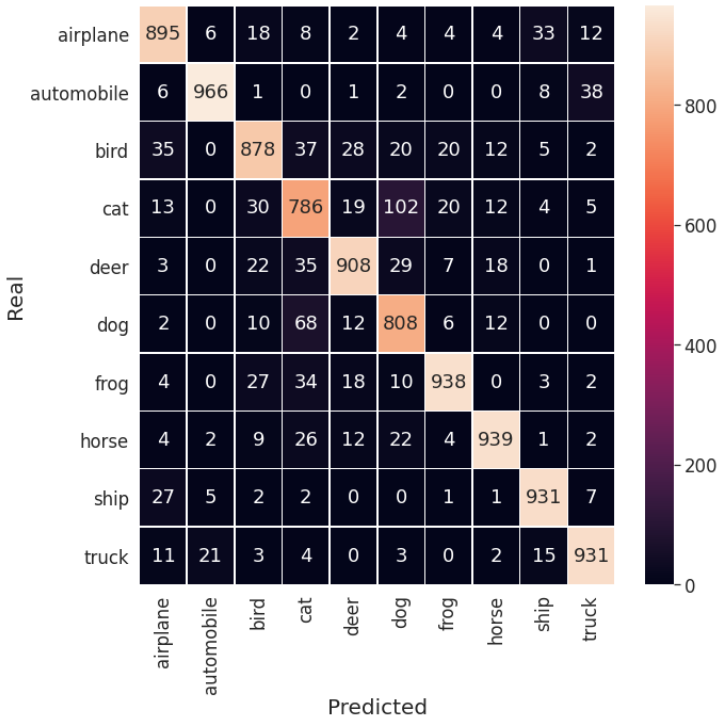
Source: Elaborated by the author

Figure 4.3: Confusion matrix obtained by CoNAS space A



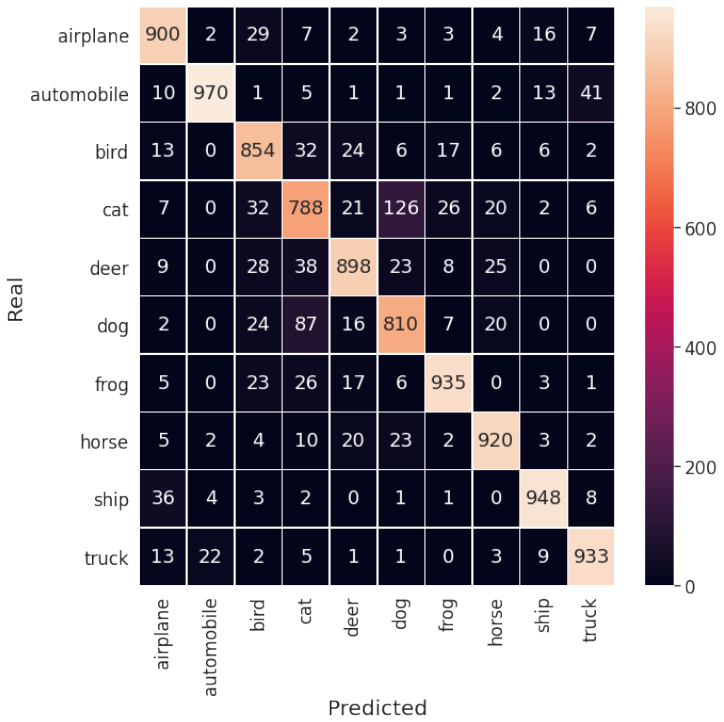
Source: Elaborated by the author

Figure 4.4: Confusion matrix obtained by CoNAS space B



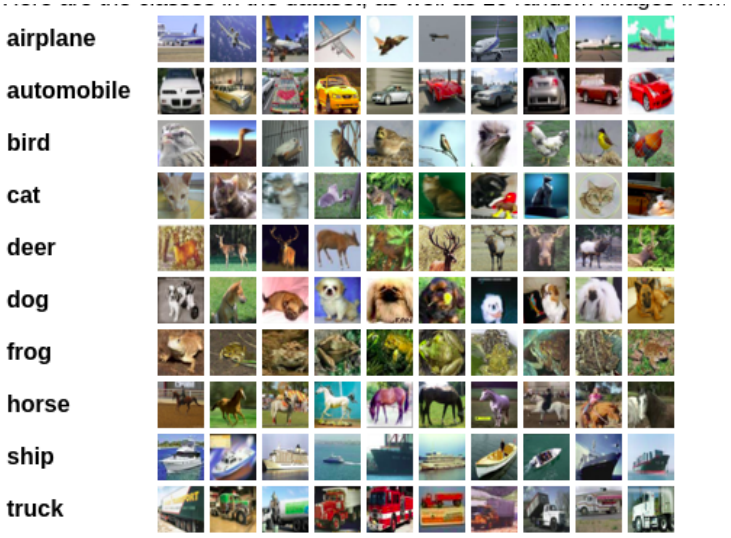
Source: Elaborated by the author

Figure 4.5: Confusion matrix obtained by CoNAS space C.



Source: Elaborated by the author

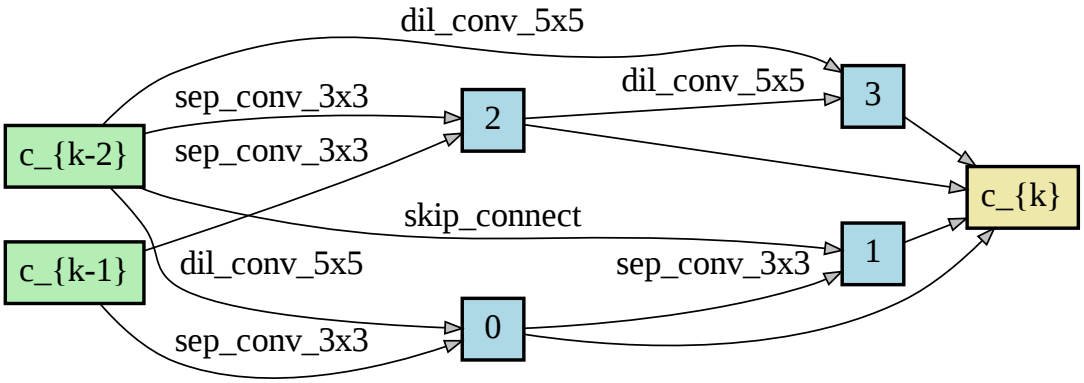
Figure 4.6: Cifar 10 examples.



Source: Elaborated by the author

Figures 4.7, 4.8, and 4.9 illustrate the differences obtained in the final architectures caused by the segmentation of the search spaces, showing that it is possible to obtain diversity of representations by segmenting the DARTS search space.

Figure 4.7: Snapshots of the most likely normal conv from search space 1.



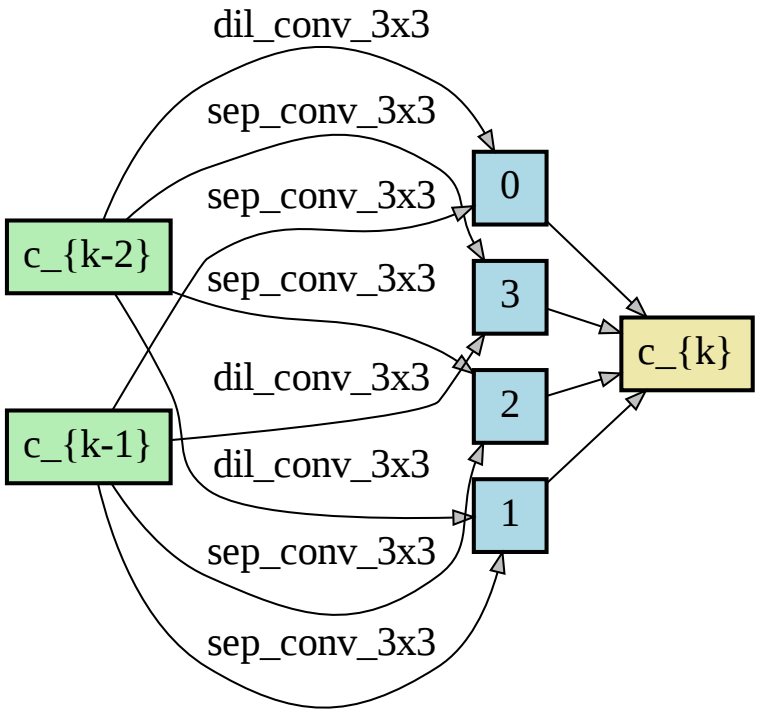
Source: Elaborated by the author

The model generated by the darts classified the image as being a cat, CoNAS got it right by classifying the image as a bird. Figure 4.10.

The model generated by the darts classified the image erroneously, whereas CoNAS got the image classification right Figure 4.11.

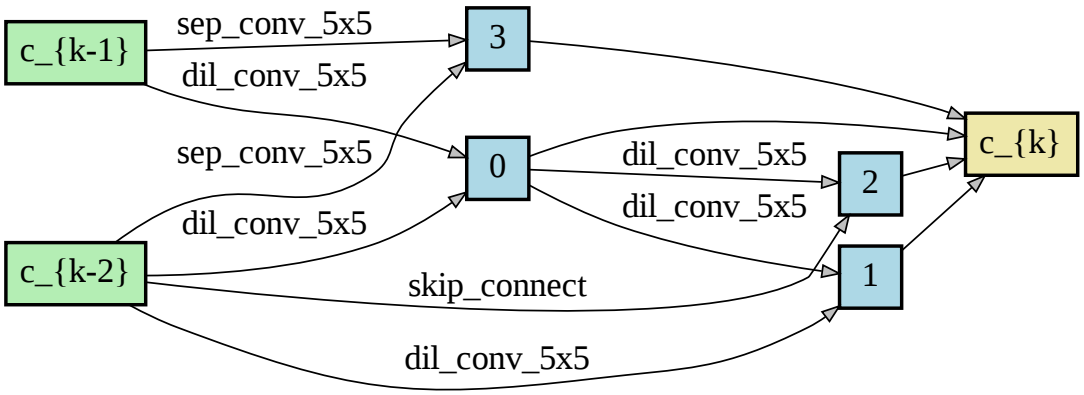
A image of the intel dataset class named buildings can be seen in Figure 4.12.

Figure 4.8: Snapshots of the most likely normal conv from search space 2.



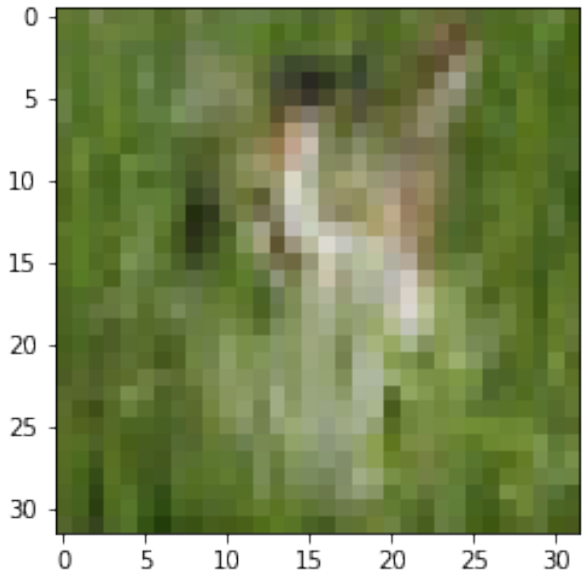
Source: Elaborated by the author

Figure 4.9: Snapshots of the most likely normal conv from search space 3.



Source: Elaborated by the author

Figure 4.10: Example taken from cifar-10 for class bird.



Source: Elaborated by the author

Figure 4.11: A image of the dataset imagenette.

Source: Elaborated by the author

Figure 4.12: A image of the dataset intel from class buildings.

Source: Elaborated by the author

The model generated by the darts classified the image as being forest, whereas CoNAS agreed on the image classification; The train test split is 0,36 for test size; Figure 4.12.

A random image of the cellulas dataset class named HUVEC-03 can be seen in Figure 4.13.

For all of the following datasets, we use the same DARTS transform for imagenet, making them all have the same resolution.

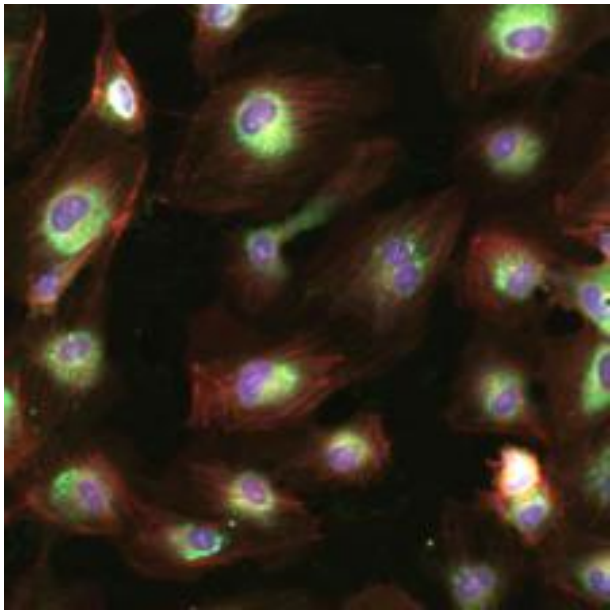
The Imagenette dataset (HOWARD, 2019) consists of a subset of 10 "easy" classes from the Imagenet dataset (DENG et al., 2009). The goal behind assembling a small version of the Imagenet dataset was mainly because running new experiments using Imagenet takes a long time.

We experimented with the Imagenette training with transfer learning from the networks trained in CIFAR-10. Since Imagenette does not have a test set, the results shown in Table 4.2 refer to the validation set. Overall, the results show that CoNAS presents an increase of about 0.76% in accuracy when compared to the regular DARTS.

Figure 4.14 illustrates the results of the evaluations in the training step. The results show the possibility of using early stopping to avoid a possible overfit and reduce the time to evaluate the architecture.

We experimented with the Intel test set with transfer learning from the networks trained in CIFAR-10. The results shown in Table 4.3 refer to the test set. Overall, the results show that CoNAS presents an increase of about 7% in accuracy when compared to the regular DARTS.

Figure 4.13: A random image of the dataset cellulas.



Source: Elaborated by the author

Table 4.2: Experimental results on Imagenette. Accuracy, number of parameters and GPU cost.

Architecture	ACC	Params (M)	GPU (Hours)
DARTS	88.05%	4.71	2
DARTS space A	87.23%	4.83	2
DARTS space B	85.63%	5.47	2
DARTS space C	86.67%	4.65	2
DARTS ensemble	**88.81%**	**14.95**	**6***

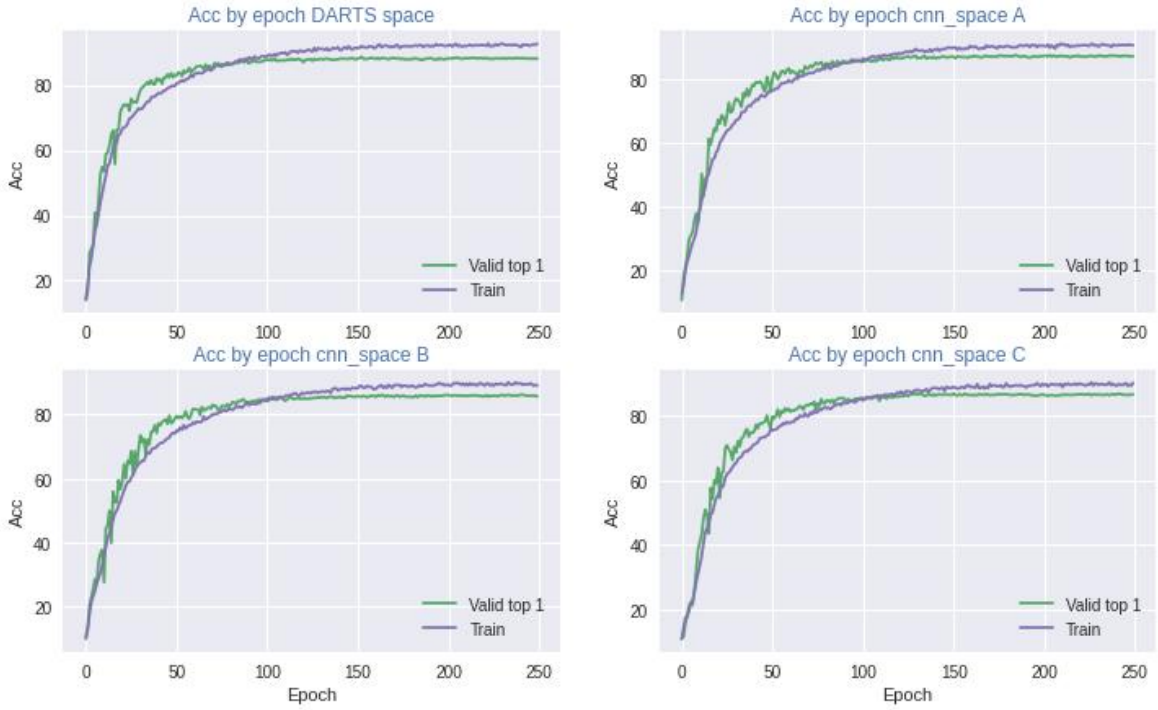
* This time considers we run each search space sequentially.

Table 4.3: Experimental results on Intel Dataset. Accuracy, number of parameters and GPU cost.

Architecture	ACC	Params (M)	GPU (Hours)
DARTS	74.06%	3.95	0.43
DARTS space A	71.83%	3.89	0.43
DARTS space B	75.33%	3.95	0.43
DARTS space C	71.43%	3.89	0.43
DARTS ensemble	**81.26%**	**11.73**	**1.29***

* This time considers we run each search space sequentially.

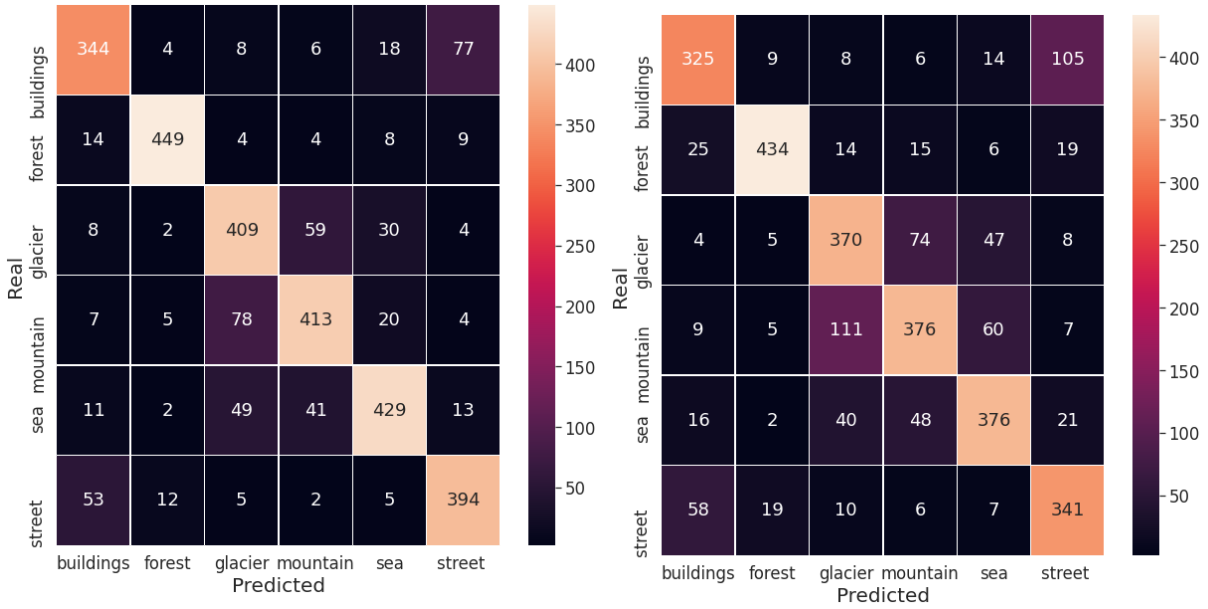
Figure 4.14: Results of the validation and training set in the training of neural networks in Imagenette.



Source: Elaborated by the author

Figure 4.15 ilustrates confusion matrix of the Intel dataset. On the left, we have the ensemble and while on the right we have the intel in the full DARTS.

Figure 4.15: Confusion matrix obtained by CoNAS (left) and DARTS (right) on Intel.

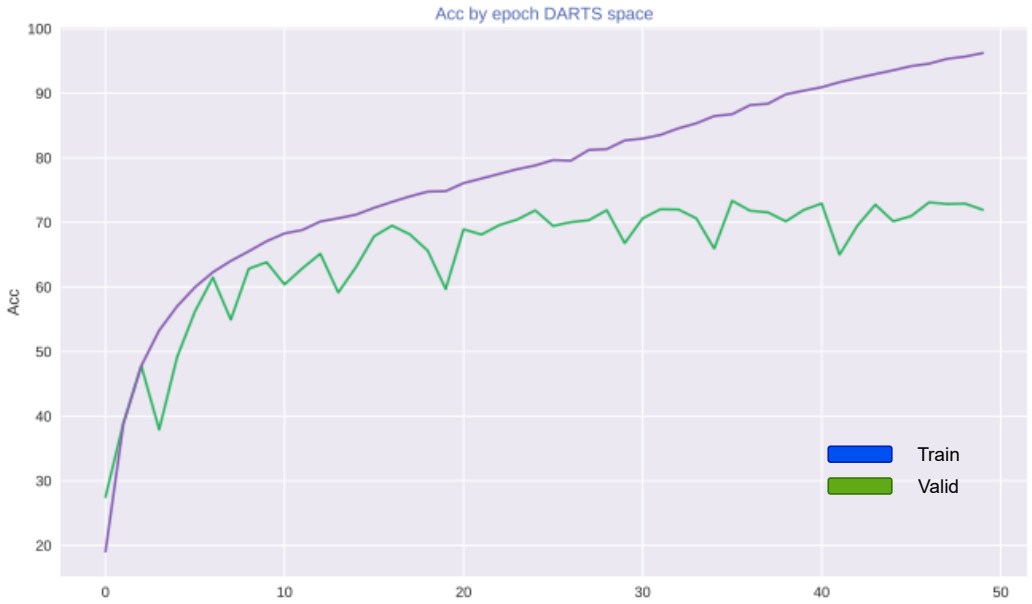


Source: Elaborated by the author

We experimented with the Cellulas train/val with transfer learning from the networks trained in CIFAR-10. The results shown in Figure 4.16 Darts manages to overfit the model in training time.

In the test set, the model reach even 1.364% accuracy. The models for CONAS were executed and obtained the same result.

Figure 4.16: Accuracy for each epoch in Cellulas dataset with DARTS.



Source: Elaborated by the author

Chapter 5

FINAL CONSIDERATIONS

{Esse também espaço para melhorar. Eu acredito que aqui deva entrar um apanhado da importância de NAS de novo, levando a falar o que está começando agora[1]). Além disso, vale a pena melhorar a discussão sobre as limitações para falar sobre como podemos melhorar. em vez de um parágrafo de future work, vale falar a limitação a imagens (aí falar que seria legal ter em outros domínios, mas que isso demandaria adicionar operações, etc). Depois, falar que os sub-espaços foram definidos manualmente, o que pode enviesar um pouco o CoNAS. Daí terminar falando que isso pode ser abordado com aleatoriedade, de forma que a gente possa escolher aleatoriamente subconjuntos de operações. Além disso, com isso, o usuário poderá escolher número de redes para fazer o ensemble e tal.}

NAS is a hot research topic and has been obtaining competitive results with humans in several domains such as image classification, object detection, and even natural language processing. This dissertation presents an investigation that aimed to take a step further on NAS research.

We introduced CoNAS, a committee of NAS-based models, to classify images. To the best of our knowledge, this is the first proposal of an ensemble algorithm based on automated neural architecture search to explore different search spaces to provide diversity to the induced models.

CoNAS resulted in superior results in CIFAR-10 over sub-spaces of DARTS and full DARTS. These results may indicate that investigation on ensemble-based network architecture is a promising path for NAS researchers' future efforts.

Also, we evaluated the architectures found for CIFAR-10 on predicting images from the Imagenette, Intel, and Cellulas datasets using transfer learning. Once again, CoNAS have shown superior accuracy over the other evaluated neural models.

[1] We presented CoNAS, a committee of NAS-based models, to classify images.

The CoNAS subspaces were divided manually, which can bias the final result. It can be solved by using a random choice of operations that compose each subspace. Besides, it allows the user to choose the number of spaces as a hyper-parameter. It has been left to future work.

If executed in parallel, the gain in accuracy is evident in almost all tested datasets, however, sequentially it takes a lot of time, which makes the cost-benefit ratio low.

Moreover, we also intend to extend CoNAS to a broader set of data, including different modalities, such as text and time series. It may be done by considering other operations in the search spaces. Finally, we plan to evaluate how the limitation of different functions impacts creating more (or less) diverse models, improving the committee's effectiveness.

5.1 Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. We also thank B2W Digital, a partner company of this project, which financed and provided computational resources and data to make possible the study, through the extension activity #23112.000186/2020-97, Federal University of São Carlos.

REFERENCES

Abouelnaga, Y. et al. Cifar-10: Knn-based ensemble of classifiers. In: *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. [S.l.: s.n.], 2016. p. 1192–1195.

ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. In: *2017 International Conference on Engineering and Technology (ICET)*. [S.l.: s.n.], 2017. p. 1–6.

BAKER, B. et al. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2017.

BERGSTRA, J.; YAMINS, D.; COX, D. D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: *International Conference on International Conference on Machine Learning - Volume 28*. [S.l.]: JMLR.org, 2013. p. I–115–I–123.

BROCHU, E.; CORA, V. M.; FREITAS, N. D. *A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning*. Oxford, UK, 2009.

CAUCHY, M. A. Méthode générale pour la résolution des systèmes d'équations simultanées. Übersetzt von Richard Pulskamp, 2010. *Compte rendu des séances de l'académie des sciences*, n. 2, p. 536–538, 1847. Available at: <https://cs.uwaterloo.ca/ y328yu/classics/cauchy-en.pdf>.

Searching for efficient multi-scale architectures for dense image prediction. 8699–8710 p.

Xception: Deep Learning with Depthwise Separable Convolutions. 1800-1807 p. ISSN 1063-6919.

ACM. *A unified architecture for natural language processing: Deep neural networks with multitask learning*. 160–167 p.

DENG, J. et al. ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09*. [S.l.: s.n.], 2009.

DIETTERICH, T. G. Machine-Learning Research. *AI Magazine*, v. 18, n. 4, p. 97, 1997.

DONG, X.; YANG, Y. Searching for A robust neural architecture in four GPU hours. *CoRR*, abs/1910.04465, 2019.

DOZAT, T. Incorporating Nesterov Momentum into Adam. *ICLR Workshop*, n. 1, p. 2013–2016, 2016.

DUCHI, J. C.; BARTLETT, P. L.; WAINWRIGHT, M. J. Randomized smoothing for (parallel) stochastic optimization. *Proceedings of the IEEE Conference on Decision and Control*, v. 12, p. 5442–5444, 2012. ISSN 01912216.

ELSKEN, T.; METZEN, J. H.; HUTTER, F. *Neural Architecture Search: A Survey*. 2019.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. Http://www.deeplearningbook.org.

HE, X.; ZHAO, K.; CHU, X. Automl: A survey of the state-of-the-art. *CoRR*, abs/1908.00709, 2019. Available at: <http://arxiv.org/abs/1908.00709>.

HINTON, G. et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, v. 29, 2012.

HOWARD, J. *imagenette*. 2019. Available at: <https://github.com/fastai/imagenette/>.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. Sequential model-based optimization for general algorithm configuration. In: SPRINGER. *International Conference on Learning and Intelligent Optimization*. [S.l.], 2011. p. 507–523.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. Sequential model-based optimization for general algorithm configuration. In: SPRINGER. *International Conference on Learning and Intelligent Optimization*. [S.l.], 2011. p. 507–523.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. (Ed.). *Automated Machine Learning: Methods, Systems, Challenges*. 1. ed. [S.l.]: Springer, 2018. ISBN 978-3-030-05318-5.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. *Automatic machine learning: methods, systems, challenges*. [S.l.]: Springer, 2019.

JACOBS, R. A. Increased rates of convergence through learning rate adaptation. *Neural Networks*, v. 1, n. 4, p. 295–307, 1988. ISSN 08936080.

JIN, H.; SONG, Q.; HU, X. Auto-keras: An efficient neural architecture search system. *arXiv preprint arXiv:1806.10282*, 2018.

KINGMA, D. P.; BA, J. *Adam: A Method for Stochastic Optimization*. 2014.

KINGMA, D. P.; BA, J. L. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, p. 1–15, 2015.

KITTLER, J. Combining classifiers: A theoretical framework. *Pattern Analysis and Applications*, v. 1, p. 18–27, 2005.

KLEIN, A. et al. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In: SINGH, A.; ZHU, J. (Ed.). *International Conference on Artificial Intelligence and Statistics*. Fort Lauderdale, FL, USA: PMLR, 2017. (Proceedings of Machine Learning Research, v. 54), p. 528–536.

KOTTHOFF, L. et al. Auto-weka: Automatic model selection and hyperparameter optimization in weka. In: HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. (Ed.). *Automated Machine Learning: Methods, Systems, Challenges*. Cham, Switzerland: Springer International Publishing, 2019. p. 81–95.

Krishnakumar, H.; Williamson, D. S. A comparison of boosted deep neural networks for voice activity detection. In: *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. [S.l.: s.n.], 2019. p. 1–5.

KRIZHEVSKY, A. Learning multiple layers of features from tiny images. In: . [S.l.]: Technical Report TR-2009, University of Toronto, Toronto, 2009.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436, 2015.

LI, L.; TALWALKAR, A. *Random Search and Reproducibility for Neural Architecture Search*. 2019.

LIANG, H. et al. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

LIU, H. et al. Hierarchical representations for efficient architecture search. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, p. 1–13, 2018.

LIU, H.; SIMONYAN, K.; YANG, Y. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018. Available at: <http://arxiv.org/abs/1806.09055>.

MARCUS, M. P.; SANTORINI, B.; MARCINKIEWICZ, M. A. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, v. 19, n. 2, p. 313–330, 1993.

Marino, M.; Virupakshappa, K.; Oruklu, E. A stacked ensemble neural network classifier for ultrasonic non-destructive evaluation applications. In: *2020 IEEE International Ultrasonics Symposium (IUS)*. [S.l.: s.n.], 2020. p. 1–4.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. (Ed.). *Towards Automatically-Tuned Neural Networks*, v. 64 of *Proceedings of Machine Learning Research*, (Proceedings of Machine Learning Research, v. 64). New York, New York, USA: PMLR, 2016. 58–65 p.

MILLER, G. F.; TODD, P. M.; HEGDE, S. U. Designing neural networks using genetic algorithms. In: *International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989. p. 379–384. ISBN 1-55860-066-3.

Nagahamulla, H.; Ratnayake, U.; Ratnaweera, A. Optimizing member selection for neural network ensembles using genetic algorithms. In: *2016 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*. [S.l.: s.n.], 2016. p. 1–5.

NAGARAJAH, T.; PORAVI, G. An extensive checklist for building automl systems. In: *AMIR@ ECIR*. [S.l.: s.n.], 2019. p. 56–70.

POLYAK, B. T. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, v. 4, n. 5, p. 1–17, 1964. ISSN 00415553.

PONTI, M. Combining classifiers: from the creation of ensembles to the decision fusion. In: IEEE. *2011 24th SIBGRAPI Conference on Graphics, Patterns, and Images Tutorials*. [S.l.], 2011. p. 1–10.

REAL, E. et al. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

REAL, E. et al. Aging evolution for image classifier architecture search. In: *AAAI 2019*. [S.l.: s.n.], 2019.

Large-scale Evolution of Image Classifiers, (ICML'17). [S.l.]: JMLR.org, 2017. 2902–2911 p.

REN, P. et al. *A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions*. 2021.

Rijal, N. et al. Ensemble of deep neural networks for estimating particulate matter from images. In: *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*. [S.l.: s.n.], 2018. p. 733–738.

RUDER, S. An overview of gradient descent optimization algorithms. p. 1–14, 2016. Available at: <http://arxiv.org/abs/1609.04747>.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning Representations by Back-propagating Errors. *Nature*, v. 323, n. 6088, p. 533–536, 1986. Available at: <http://www.nature.com/articles/323533a0>.

Shahriari, B. et al. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, v. 104, n. 1, p. 148–175, Jan 2016.

SHEN, D.; WU, G.; SUK, H.-I. Deep learning in medical image analysis. *Annual review of biomedical engineering*, Annual Reviews, v. 19, p. 221–248, 2017.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Leaning*. [S.l.: s.n.], 2018. 481 p. ISBN 9780262039246.

TAO, S. Deep neural network ensembles. *CoRR*, abs/1904.05488, 2019.

Wolpert, D. H.; Macready, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, v. 1, n. 1, p. 67–82, 1997.

WU, N.; XIE, Y. *A Survey of Machine Learning for Computer Architecture and Systems*. 2021.

Yang, J. et al. Effective neural network ensemble approach for improving generalization performance. *IEEE Transactions on Neural Networks and Learning Systems*, v. 24, n. 6, p. 878–887, 2013.

ZEILER, M. D. *ADADELTA: An Adaptive Learning Rate Method*. 2012.

ZHANG, Q. et al. A survey on deep learning for big data. *Information Fusion*, Elsevier, v. 42, p. 146–157, 2018.

Practical block-wise neural network architecture generation. 2423–2432 p.

ZHONG, Z. et al. Blockqnn: Efficient block-wise neural network architecture generation. *arXiv preprint arXiv:1808.05584*, 2018.

ZOPH, B.; LE, Q. V. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.

ZOPH, B. et al. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2018.