Université de Lille

Thèse

pour obtenir le grade de :
Docteur de l'Université de Lille
dans la spécialité Informatique

par

Meyssa Zouambi

Optimizing Deep Learning: Navigating the Field of Neural Architecture Search from Theory to Practice

Optimiser l'apprentissage profond: Explorer la recherche d'architecture neuronale, de la théorie à la pratique

Thèse soutenue le 8 décembre 2023 devant le jury composé de :

M.	Philippe PREUX	Professeur des universités Université de Lille	(Examinateur - Président du Jury)
M.	Alexandre CAMINADA	Professeur des universités Polytech Nice Sophia	(Rapporteur)
Mme	Samira SADAOUI	Full professor University of Regina	(Rapporteur)
M.	Nicolas MONMARCHÉ	Maître de conférences Université de Tours	(Examinateur)
Mme.	Clarisse DHAENENS	Professeur des universités Université de Lille	(Directrice de Thèse)
Mme.	Julie JACQUES	Maîtresse de conférences Université de Lille	(Co-Encadrante de Thèse)

Laboratoire CRIStAL
Université de Lille - Campus scientifique Bâtiment ESPRIT
Avenue Henri Poincaré 59655 Villeneuve d'Ascq France

Abstract

In the realm of deep learning, the design and optimization of neural architectures are crucial for achieving high-performance models. This process, based on trial and error, has been done manually for decades and is both time and resource-consuming. This thesis delves into the domain of Neural Architecture Search (NAS), a promising technique that seeks to automate this process. The research explores the complexities inherent in NAS, highlighting the challenges of navigating the vast search space of potential architectures. It investigates methods based on Local Search and proposes two efficient algorithms built around it, namely LS-Weight and LS-PON. Each method offers a distinctive approach to integrate knowledge during the search to offer efficient and more frugal strategies to NAS. Furthermore, as deep learning models are often governed by multiple competing objectives such as accuracy, complexity, and computational efficiency, this research also delves into multi-objective optimization within NAS. This ensures that the resulting architectures are not only performant for the task they are designed for but also aligned with multiple criteria essential for real-world applications. For this purpose, this research offers an alternative approach to multi-objective NAS that addresses certain issues found in strategies from the literature. On top of that, it also analyzes the complexity of moving from benchmarks to real data, offering a protocol that guides practitioners in their usage of NAS for their applications. Lastly, by recognizing the importance of domain applications, this work focuses on healthcare images to validate these contributions. It also presents a detailed survey on the use of NAS for healthcare, by analyzing more than 40 contributions in the literature and laying the ground for future works in the field.

Résumé

Dans le domaine de l'apprentissage profond, la conception et l'optimisation des architectures neuronales sont essentielles pour obtenir des modèles performants. Ce processus, basé sur une démarche d'essais et d'erreurs, a été effectué manuellement pendant des décennies et consomme beaucoup de temps et de ressources. Cette thèse explore le domaine de la recherche d'architectures neuronales, ou Neural Architecture Search (NAS), une technique prometteuse visant à automatiser ce processus. Elle aborde les complexités inhérentes aux NAS, mettant en lumière les défis liés à l'exploration de l'immense espace de recherche des architectures. Cette thèse étudie des méthodes basées sur la recherche locale et propose deux algorithmes efficaces construits autour de celle-ci, à savoir LS-Weights et LS-PON. Chacune de ces méthodes offre une approche distincte pour intégrer des connaissances pendant la recherche, afin d'obtenir une stratégie plus efficace et économe en resources pour les NAS. Par ailleurs, puisque les modèles d'apprentissage profond sont souvent régis par plusieurs objectifs, tels que la précision, la complexité, et l'efficacité de calcul, cette recherche s'intéresse aussi à l'optimisation multi-objectifs au sein des NAS. Cela garantit que les architectures résultantes ne sont pas seulement performantes pour la tâche pour laquelle elles sont conçues, mais aussi alignées avec plusieurs critères essentiels pour des applications réelles. Pour cela, une méthode alternative aux NAS multi-objectifs est proposée, résolvant certains problèmes identifiés dans les approches existantes de la littérature. De plus, ce travail examine la complexité du passage des benchmarks aux données réelles, offrant un protocole qui guide les praticiens dans l'utilisation des NAS pour leurs applications. Enfin, reconnaissant l'importance des applications dans des domaines spécifiques, ce travail met l'accent sur des tâches d'imagerie médicale pour valider ces contributions. Il présente également une analyse approfondie de l'utilisation des NAS dans le secteur de la santé, étudiant plus de 40 contributions de la littérature afin d'établir une base pour des recherches futures dans ce domaine.

Acknowledgements

I would like to express my deepest gratitude to my thesis supervisors, Clarisse DHAENENS and Julie JACQUES, for their unwavering support and guidance throughout the research and writing of this thesis. Their expertise and insightful advice have been invaluable in shaping this work. I will always be grateful for their kindness, their patience, and their presence. They have made this journey a wonderful experience.

My heartfelt thanks extend to the members of the jury, Alexandre CAMINADA, Samira SADAOUI, Philippe PREUX, and Nicolas MONMARCHÉ, for their valuable insights, their feedback, and the time they dedicated to evaluating my work.

I am profoundly thankful to the ORKAD team, particularly my fellow peers, for all the laughter and the fun we had in the last three years. I will miss our gaming breaks, endless trips to the water fountain, and all the cheerful moments we spent together.

To the people I met during my travels, who made every place feel special. I will eternally cherish the peace and joy I found in foreign lands in your company.

To my dear friends, even miles away, you consistently managed to take care of me. I have yet to find kindheartedness that could match yours. Thank you for everything.

To my ”home away from home,” the friends who have become like family to me here, your presence in my life has been a source of immeasurable joy and comfort. I look forward to the many adventures that await us in the future.

Last but not least, my deepest love and gratitude to my wonderful family. Their love, support, relentless encouragement, and belief in me have been the backbone of my journey. They are my constant source of motivation, and I carry their love with me in everything I do.

Table of Contents

List of Figures viii
List of Tables x

General Introduction **1**
Motivations 1
Outline 2

1 The Anatomy of Deep Learning: Foundations, Architectures, and Challenges **5**
1.1 Introduction 5
1.2 Demystifying deep learning 6
1.2.1 The building blocks of neural networks 6
1.2.2 Training and evaluation 7
1.3 Famous deep neural networks 8
1.3.1 Convolutional neural networks 8
1.3.2 Recurrent neural networks 10
1.3.3 Autoencoders networks 10
1.3.4 Restricted Boltzmann machines and Deep belief networks 11
1.3.5 Generative adversarial networks 11
1.3.6 Transformers networks. 12
1.4 Deep learning challenges 13
1.5 Conclusion 15

2 The Tale of Neural Architecture Search **16**
2.1 Introduction 16
2.2 Automated machine learning 17
2.3 Neural architecture search 18
2.3.1 NAS as an optimization problem 19
2.3.2 The building blocks of the NAS framework 20
2.3.3 NAS benchmarks 31
2.4 Conclusion 35

3 A Quest for Efficiency: Building on Local Search for NAS **36**
3.1 Introduction . . . 36
3.2 Local search for NAS . . . 38
3.2.1 Defining the important components . . . 38
3.2.2 How to make LS more efficient for NAS . . . 40
3.3 LS-Weights: Local Search and the Weight of the Parameters . . . 41
3.3.1 Determining the importance of architectural elements . . . 42
3.3.2 Experiments . . . 42
3.3.3 Discussion . . . 44
3.4 LS-PON: Local Search and the Predicted Order of Neighbors . . . 45
3.4.1 How it works . . . 46
3.5 Experiments . . . 48
3.5.1 Results . . . 50
3.6 Conclusion . . . 53

4 Multi-objectively, NAS **55**
4.1 Introduction . . . 55
4.2 Multi-objective optimization in a nutshell . . . 56
4.2.1 Problem formulation . . . 56
4.2.2 Pareto dominance and Pareto optimality . . . 56
4.2.3 Techniques in multi-objective optimization . . . 56
4.2.4 Performance metrics . . . 59
4.3 The state of the literature in multi-objective NAS . . . 60
4.3.1 Common objectives in neural network design . . . 60
4.3.2 Approaches for multi-objective NAS . . . 61
4.4 Conclusion . . . 65

5 An Alternative Pareto-based Approach for Multi-objective NAS **66**
5.1 Introduction . . . 66
5.2 Introducing dominance-based multi-objective local search . . . 67
5.3 DMLS for NAS . . . 69
5.3.1 Solution encoding . . . 70
5.3.2 DMLS components . . . 70
5.4 Experimentation protocol . . . 72
5.5 Results and discussion . . . 73
5.5.1 Hypervolume results . . . 73
5.5.2 Analysis of Pareto points found . . . 74
5.5.3 Discussion . . . 76
5.6 Conclusion and future directions . . . 76

6 Bridging Barriers: from the Benchmarks to the Real World **77**
6.1 Introduction . . . 77
6.2 The challenges of NAS in the real world . . . 77
6.3 Diving in with medical image classification . . . 79
6.3.1 Datasets . . . 79
6.3.2 Behind the scenes . . . 80
6.3.3 Experimental protocol . . . 82
6.3.4 Results . . . 83
6.4 Conclusion . . . 87

7 Neural Architecture Search in Healthcare: An Introductory Survey to Unlocking Opportunities **88**
7.1 Introduction . . . 88
7.2 Context . . . 89
7.3 Lesion detection and classification . . . 89
7.3.1 Our takeaway . . . 90
7.4 Medical image segmentation . . . 93
7.4.1 2D segmentation . . . 93
7.4.2 3D segmentation . . . 97
7.5 Other medical-related tasks . . . 101
7.5.1 Our takeaway . . . 102
7.6 Global discussion and future research directions . . . 104

General Conclusion **106**
Contribution Summary . . . 106
Future Research . . . 108

Published and Submitted Works **110**

Bibliography **111**

List of Figures

1 Thesis outline. 4

1.1 A multilayer perceptron is composed of multiple layers. Each layer contains a number of perceptrons. We distinguish three types of layers: input, output, and hidden layers. 6
1.2 Perceptron. 7
1.3 hierarchical learning in CNNs, figure taken from (LeCun et al., 2015). 9
1.4 Example of a CNN architecture. 9
1.5 RNN architecture. 10
1.6 composition of an RNN block. 10
1.7 Autoencoder. 11
1.8 DBN architecture. 12
1.9 GAN representation. 12
1.10 Transformer Architecture, figure inspired from (Vaswani et al., 2017). 13

2.1 The basic machine learning pipeline. 17
2.2 The NAS framework 20
2.3 Examples of two architectures from a global search space 21
2.4 Example of a generic architecture in a cell-based search space 22
2.5 Constructing motifs in the hierarchical search space 23

3.1 Encoding of an architecture from NAS-Bench-201. The top represents the encoding which contains the components of the cell that is present in the architecture backbone in red (best seen in colors). Each cell has the same structure. 39
3.2 Neighbor generation with *One exchange neighborhood* function. 40
3.3 Neighbor generation with LS-Weights. * The mutation selects the new values based on the given probabilities. 41
3.4 The evolution of the validation accuracy across the number of sampled architectures. 44
3.5 Representation of a simplified architecture and the corresponding values used for the linear regression model. Smaller rank scores correspond to better architectures . 47
3.6 LS-PON Ordereing. 47
3.7 Main steps of LS-PON. 48

3.8 Example of the evolution of the validation accuracy across the number of evaluated architectures for a dataset of each benchmark. NAS201-IMNT, MNBench-C100, and NAS301-C10 correspond to NAS-Bench-201 for ImageNet16-120, MacroNas-Benchmark for Cifar100, and NAS-Bench-301 for Cifar10. Results are averaged for 150 runs for all algorithms. The shaded region indicates the standard deviation of each search method. 52
3.9 Box plots represent the distribution of distances between the validation accuracy of a solution and its neighbors, and diamonds represent outliers. NAS201, MNBench, NAS301 stand for NAS-Bench-201, MacroNasBenchmark and NAS-Bench-301. C10, C100 and IMNT are respectively Cifar10, Cifar100 and ImageNet16-120. . . 53

4.1 Example of a Pareto-front for a bi-objective problem. 57

5.1 DMLS iteration 68
5.2 Solution encoding 70
5.3 Neighbor generation with the neighborhood function 71
5.4 Hypervolume using different initial sizes for the archive 73
5.5 Evolution of the hypervolume across the number of evaluated architectures for CIFAR10 and CIFAR 100. Results are averaged for 30 runs for each algorithm. . . 74
5.6 Evolution of the percentage of Pareto optimal solutions found across the number of evaluated architectures for CIFAR10 and CIFAR 100. Results are averaged for 30 runs for each algorithm. 75

6.1 Training curve of a sample of architectures from the NAS-Bench-201 search space. The x-axis represents the number of epochs and the y-axis the accuracy. 81
6.2 Training curve of a sample of architectures from the NAS-Bench-301 search space. The x-axis represents the number of epochs and the y-axis the accuracy. 81
6.3 Training curve of a sample of architectures from the NAS-Bench-301 search space using different learning rates. The best learning rate is the one in orange as it gives the smoothest curve with appropriate progress. 82
6.4 Experimental protocol 83
6.5 Results on PneumoniaMNIST dataset 84
6.6 Results on DermaMNIST dataset 86

List of Tables

2.1 Summary of famous NAS benchmarks. The highlighted benchmarks will be used in our studies. 31

3.1 Number of convolution and pooling operations in popular CNN architectures. . . . 42
3.2 Parameter values, ranks, and probability of selection 43
3.3 The table indicates the mean and std of the test accuracy found at the end of the search. The *Optimal* value is given by the benchmark and represents the highest accuracy possible using this search space. 43
3.4 The table indicates the mean and std of the number of evaluated architectures before convergence. The *Speedup* represents how fast LS-Weights is compared to LS. 44
3.5 Performance evaluation on the three benchmarks. Each algorithm uses the validation accuracy as a search signal. *Optimal* indicates the highest validation accuracy provided by the benchmark. We report the mean and std deviation of 150 runs for Random search, LS, LS-PON. † taken from Dong and Yang (2020). 50
3.6 Speed evaluation on the three benchmarks. The table indicates the mean and std deviation of the number of evaluated architectures before convergence. Values in bold mean statistically better (Wilcoxon's test). 51
3.7 Results on all benchmarks. The table indicates the mean and std deviation of the number of required evaluations before improving on the current solution during the search. Values in bold mean statistically better (Wilcoxon's test) 51

4.1 Illustrative work from the literature 64

5.1 Table reporting the hypervolume (scaled for better readability) for each method on CIFAR10 and CIFAR100. Results at presented for 500, 1000, 3000, and 6000 evaluations. Values in bold are statistically better (Mann-Whitney U rank test). . . 74
5.2 Table reporting the percentage of Pareto optimal solutions found for each method on CIFAR10 and CIFAR100. The results at presented for 500, 1000, 3000, and 6000 evaluations. Values in bold are statistically better (Mann-Whitney U rank test) 75

6.1 The table indicates the Area Under the Curve and the Accuracy on the Pneumonia test set for each model. (28) or (224) corresponds to the size of the input image. The images are resized to 224 x 224 by PIL.Image.NEAREST option using the Pillow package. Results of the other methods are found in (Yang et al., 2023). . . . 85
6.2 The table indicates the Area Under the Curve and the Accuracy on the DermaMNIST test set for each model. (28) or (224) corresponds to the size of the input image of the model. The images are resized to 224 x 224 by PIL.Image.NEAREST option using the Pillow package. Results of the other methods are found in (Yang et al., 2023). . . . 87

7.1 Applications of lesion detection and classification papers . . . 91
7.2 NAS details for Lesion detection and classification papers . . . 92
7.3 Applications of 2D segmentation papers . . . 95
7.4 NAS details for 2D segmentation papers . . . 96
7.5 Applications of 3D segmentation papers . . . 99
7.6 NAS details for 3D segmentation papers . . . 100
7.7 Applications of healthcare supporting tasks papers . . . 102
7.8 NAS details for healthcare supporting tasks papers . . . 103

General Introduction

"Nanos gigantum humeris insidentes"

— Bernard of Chartres

Motivations

At the core of recent technological advancements, deep learning has emerged as a transformative power, enabling machines to interpret, learn from, and make predictions based on data. It can solve complex problems across various domains, ranging from medical diagnoses and autonomous vehicle navigation to financial forecasting and content recommendation. Advances in deep learning showcase an impressive ability to identify patterns in images, understand natural language, recognize speech, and much more.

Unlike traditional machine learning models, deep learning employs neural networks, and the effectiveness of the models heavily relies on its architecture. Neural networks are composed of several layers through which the data is transformed. Designing these architectures is a meticulous task. Layers, nodes, and connections need to be defined, a process that is mainly based on trial and error and human intuition. It demands an in-depth understanding of deep learning, the data, and the task at hand. The complexity of this design process, as well as the immense choice possibilities, requires significant time and expertise. And with the ever-evolving possibilities and domains that deep learning undertakes, the architecture of modern deep learning become significantly large and complex, making this manual task even more challenging.

Neural Architecture Search (NAS) offers a solution to address these issues. While traditional methods of designing neural network architectures relied heavily on human expertise, NAS automates this process. It leverages algorithms to discover high-performing architectures, cutting down on manual effort and potentially leading to more innovative and efficient designs. However, searching through the vast space of potential architectures is not easy, and many complex methods were developed to facilitate it. One major challenge that NAS research encounters is the immense computational costs, making it resource-intensive for small research teams or institutions. On top of that, the task of tailoring complex NAS methods to new, unseen tasks presents another layer of difficulty.

Local search is one technique that although simple has proven its superiority in optimization problems for decades. Through this research, we want to shed light on its potential to provide a

straightforward and flexible approach to finding promising architectures. Our goal is to develop an approach that is more time-efficient while still offering the simplicity for implementing to new tasks and domains.

In real-world scenarios, optimizing multiple conflicting criteria is important, especially in modern applications. For instance, while we want a neural network that is highly accurate, we might also desire it to be computationally efficient or to have a small memory footprint. Nowadays, models can run on different platforms which adds new constraints to architecture design. This introduces the realm of multi-objective NAS, where the goal is to find a set of architectures that strike a balance among various objectives, offering a set of trade-offs for practitioners to choose from. This area of research requires more attention to find approaches that are time-efficient and easy to use.

Lastly, it is important to acknowledge the significance of application domains that can take advantage of NAS, particularly healthcare. This sector is filled with challenges that require innovative and well-defined models. Achieving a high accuracy in diagnosing diseases, predicting patient outcomes, and optimizing treatment plans is crucial. By applying NAS to healthcare, we have the potential to enhance medical AI, improving patient care, reducing costs, and potentially saving lives.

In this thesis, we embark on a journey to explore the field of NAS, local search, and multi-objective optimization, with a focus on healthcare applications. Through this work, we aim to contribute to the ever-evolving narrative of AI and hopefully, help in democratizing its usage.

Outline

Following this general introduction, this thesis is organized into 6 main chapters, Figure 1 presents how these chapters are connected.

Chapter 1: The Anatomy of Deep Learning: Foundations, Architectures, and Challenges

This chapter sets the stage by introducing the foundations of deep learning. It presents the principles governing deep learning from the simple perceptron to complex neural networks. This chapter serves as an introduction to the field enabling readers to have the necessary knowledge to understand this work. It also presents the different challenges that deep learning faces and motivates the emergence of neural architecture search.

Chapter 2: The Tale of Neural Architecture Search

Chapter 2 highlights the complexities and challenges inherent in neural network design and introduces neural architecture search as an automated solution to this task. The chapter first presents

AutoML, the field of automated machine learning that encapsulates NAS. It then explains in detail the field of neural architecture search, providing a comprehensive overview of its framework, components, and prominent benchmarks.

Chapter 3: A Quest for Efficiency: Building on Local Search for NAS

Building upon the foundation presented in Chapter 2, Chapter 3 focuses on using local search strategies for NAS. Acknowledging the sometimes unnecessary complexities of prevalent NAS algorithms, this chapter explores the potential of enhancing local search heuristics for NAS, offering simpler yet effective approaches to navigating through the architectural search space. Contributions named LS-Weight and LS-PON are introduced, discussed, and validated through experiments and results.

Chapter 4: Multi-objectively, NAS

Taking a closer look at the various aspects involved in designing a neural network, Chapter 4 delves into multi-objective optimization within NAS. It underscores the importance of considering various objectives, such as predictive accuracy, model complexity, and resource efficiency, particularly in practical applications. A comprehensive exploration and classification of approaches within Multi-objective NAS are provided, highlighting the trade-offs and benefits associated with balancing multiple objectives in neural network design.

Chapter 5: An Alternative Pareto-based Approach for Multi-objective NAS

This chapter introduces and explores Dominance-based Multi-objective Local Search (DMLS) as an alternative approach to navigating the multi-objective optimization space within NAS. The design of this method in the NAS context is presented in detail, with a formulation of the multi-objective NAS problem, the different components of DMLS, and the protocol used for its implementation. This study is backed by experiments showcasing its potential in effectively exploring the multi-objective NAS landscape.

Chapter 6: Bridging Barriers: from Benchmarks to the Real World

Moving from benchmarks to practical applications, Chapter 6 identifies and navigates through the challenges and considerations that emerge when applying NAS in real-world scenarios, particularly in the healthcare domain. Through a careful design and experimental protocol, the chapter validates the approaches proposed in previous chapters, offering insights into their applicability and performance on real healthcare data.

Chapter 7: Neural Architecture Search in Healthcare: An Introductory Survey to Unlocking Opportunities

This chapter aims to bridge the gap between NAS and its practical implementation within the healthcare sector. By providing a structured overview of the opportunities, challenges, and current openings in leveraging NAS for healthcare. This chapter serves as a comprehensive guide for researchers and practitioners. It synthesizes insights from studies that have explored automatically designed neural networks in healthcare, offering a structured study into the potential of NAS within this domain.

Conclusion and Future Work

Concluding the thesis, this section will encapsulate the key contributions derived throughout the exploration of NAS, local search, and multi-objective optimization, particularly within the healthcare domain. It will also shed light on potential future research, exploring how the findings from this thesis can be expanded upon, refined, or applied in new and innovative ways.

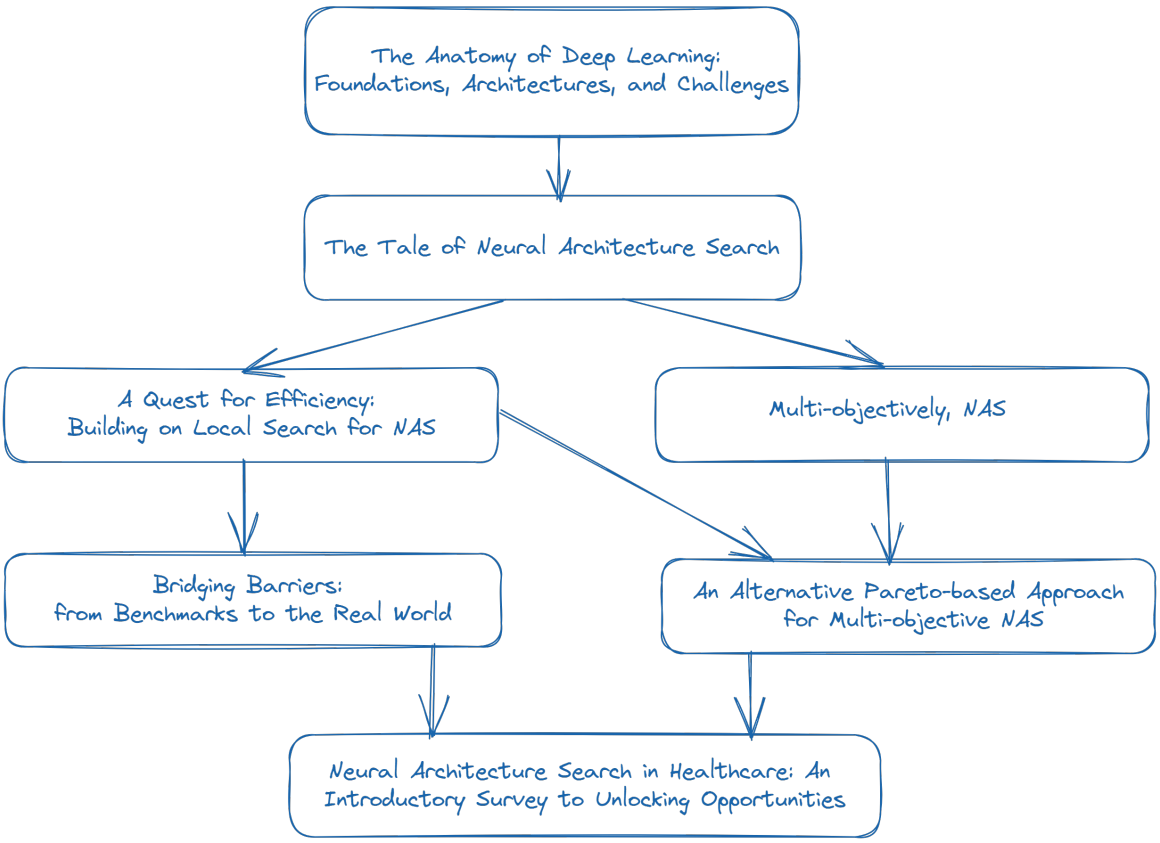


Figure 1: Thesis outline.

Chapter 1

The Anatomy of Deep Learning: Foundations, Architectures, and Challenges

> "What we want is a machine that can learn from experience."
>
> — Alan Turing

1.1 Introduction

Deep Learning (DL) is a powerful technology that has the ability to learn complex patterns from data. It has gained a lot of popularity in recent years due to its high performance in many tasks. Deep learning can solve complex problems across various domains. Tasks such as object detection, voice recognition, or natural language processing were once considered highly challenging for computers. Nowadays, they have seen significant advances thanks to this technology. At the core of deep learning lies the neural network, which represents the foundation of every deep learning model. It is composed of multiple layers responsible for extracting knowledge from data. The design of the neural network highly impacts the performance of its model, and Neural Architecture Search (NAS) stands out as a promising direction for automating its design. However, before delving into this topic and its implications in the next chapters, it's essential to establish a solid understanding of the foundational concepts and techniques that define deep learning.

This chapter serves as an introduction to this field. It enables readers to have a solid background in deep learning to better understand the field of NAS and the contributions around it. It starts by presenting the foundation of neural networks, how they are trained and evaluated, and presents the most famous types of neural networks. Lastly, this chapter shows some of the most significant challenges deep learning faces and places some of the motivation behind this thesis.

1.2 Demystifying deep learning

Machine Learning (ML) is a branch of artificial intelligence that enables computers to learn from data. It relies on algorithms that learn from experience rather than being hard-coded to perform certain tasks. Deep learning, a subset of machine learning, encapsulates techniques based on neural networks which enable models to learn more complex structures in the data. Unlike shallow learning models, which might have just one or two layers, deep models have many layers. This *depth* allows for feature extraction at various levels of abstraction, making them particularly powerful for a range of applications. To understand how they work, this section presents the fundamentals of neural networks.

1.2.1 The building blocks of neural networks

A neural network can be represented by a graph with interconnected nodes. This graph serves as a function that enables the translation of the input data into the desired output. The quintessential example of a neural network is the multi-layer perceptron (Figure 1.1). A perceptron (the basic block of this network) represents a non-linear transformation, as illustrated in Figure 1.2. It is associated with weights w, a bias b, and an activation function σ. It transforms the input $x = \{x_1, x_2, ...x_m\}$ into the output $y = \sigma\left(\sum_1^m w_i x_i + b\right)$.

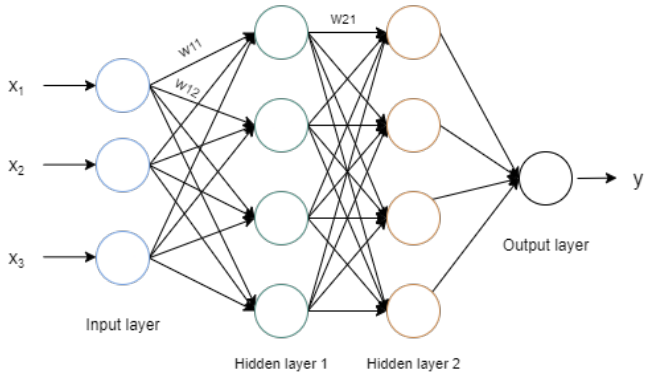


Figure 1.1: A multilayer perceptron is composed of multiple layers. Each layer contains a number of perceptrons. We distinguish three types of layers: input, output, and hidden layers.

Stacking multiple perceptrons creates a more complex operation. The weights and biases of these perceptrons represent the parameters of the function. Their values need to be adjusted by *training the network*. Training a network enables the network to *learn* the values of these parameters in order to get the desired outputs.

The Multi-Layer Perceptron (MLP) represents one of the most basic neural networks and is useful for easy classification or regression tasks. It is, however, too simple for more advanced problems or complex data. Different types of neural networks that are more suited for modern applications have emerged as a consequence. Each of these types can have distinct architecture

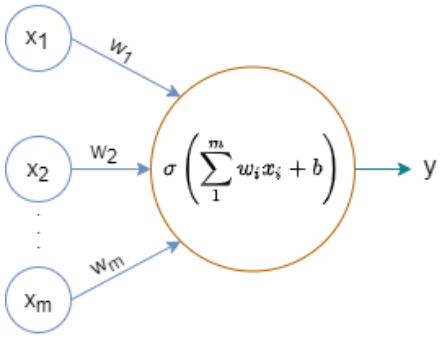


Figure 1.2: Perceptron.

components that are more complex than simple perceptrons. The term *neural architecture* refers to the specific layout, structure, or design of a neural network. It dictates how neurons are organized, how they connect, and often includes specifics about other elements like activation functions, pooling operations, skip connections, etc. More details will be provided on these elements when we present prominent types of neural networks later in this chapter.

1.2.2 Training and evaluation

Training a neural network is an iterative process where the model fine-tunes its weights based on the feedback it receives from its predictions. The objective is to minimize the difference between its predictions and the actual values, often referred to as error or loss.

To compute this error, we need to define the *loss function*, a mathematical measure of how far a network's predictions are from the actual values. Different loss functions can be selected depending on the task at hand. Examples of loss functions are Mean Squared Error, Cross-Entropy, Categorical Cross-Entroy, etc.

In terms of its practical applicability, another metric is usually provided as well: the *evaluation metric*. This metric, such as accuracy, precision, or recall, provides a more granular insight into how well the model is performing in terms of its actual objective. For instance, while a loss function might indicate how close predicted probabilities are to true labels, accuracy will directly measure the fraction of predictions the model gets right.

Another important element for training is the *Backpropagation*. It is a method used for calculating how much each weight contributes to the loss to adjust it accordingly. This is achieved by calculating the gradient of the loss function with respect to each weight.

While backpropagation calculates the necessary adjustments to the weights, optimization algorithms dictate how those adjustments are made. Gradient Descent (GD) is the most straightforward optimization method, where weights are updated in the direction of the negative gradient. Variations of gradient descent, like Stochastic Gradient Descent (SGD), Mini-batch Gradient Descent, and more sophisticated optimizers like Adam, RMSprop, and Adagrad, introduce elements like momentum and adaptive learning rates to speed up convergence and navigate the loss landscape more effectively.

The weights of the network are updated several times before reaching the desired values. The process of training includes multiple iterations often called *epochs*. For each epoch, predictions are made, the loss is computed, gradients are calculated via backpropagation, and the weights are updated by the optimizer. During this procedure, the loss typically reduces, indicating that the model is learning the patterns in the data.

During training, we must be careful that the network is learning the correct patterns and not just overlearning from the data used. This could be a symptom of extracting patterns that are too specific to the data it trained on. This concept is referred to as *overfitting*, where the network is more memorizing than learning. To monitor when this starts to happen, it is essential to validate the performance on a separate set of data the network hasn't seen during its training. If the performance on the validation set starts to degrade while the training set performance improves, it is a sign of overfitting. The size of the validation set usually represents a small fraction of the total dataset (around 20%). At the end of the training, we assess the final performance using a *test set*. This data is held out and not used during the whole training process and permits the true evaluation of the model.

This approach is usually the same for most neural networks. Different types of networks are available for different types of tasks. We will present the most famous deep neural networks in the following paragraphs.

1.3 Famous deep neural networks

Numerous types of neural networks have been developed, each with unique architecture components and suited to different kinds of tasks. Some of the most classical and widely-recognized models include:

1.3.1 Convolutional neural networks

Convolutional Neural Networks (CNN) are popular methods often used for image analysis applications. They contain layers called convolutional layers that can detect patterns in images. Each convolutional layer is made of *filters* (also called *kernels*) that can be seen as specialized pattern detectors. In the early layers of the CNN, these filters detect simple patterns such as edges, shapes, textures, etc., and output feature maps that highlight these patterns in the image. As the network goes deeper, the filters detect more complex patterns using the outputs of the previous layers and slowly combine them to detect high-level patterns such as eyes, noses, faces, etc. Figure 1.3, which is taken from LeCun et al. (2015), illustrates the hierarchical learning of concepts inside a CNN. It shows how the network sequentially builds information starting from pixels, then edges, then shapes, until reaching the full information in its output.

Working with full-sized images is computationally expensive, so convolution layers process images by small patches. Most CNNs use another type of operation called pooling. Their role is to reduce the dimensions of the feature maps while keeping the important information they carry.

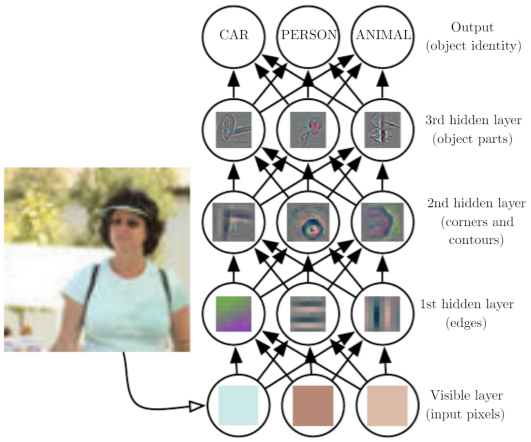


Figure 1.3: hierarchical learning in CNNs, figure taken from (LeCun et al., 2015).

This also enables deeper layers to look at larger portions of the image at a time. For an image classification task, the final layer of a CNN is usually composed of a small multi-layer perceptron that uses the final represented features as input and classifies them into the desired output. Figure 1.4 shows a classical composition of a CNN.

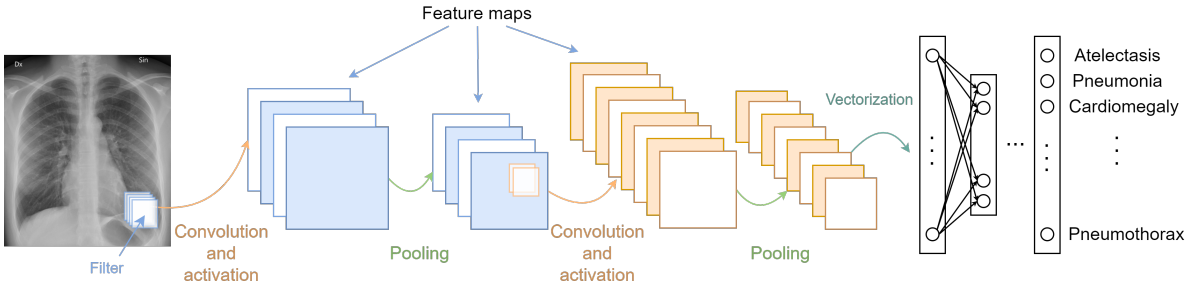


Figure 1.4: Example of a CNN architecture.

As classification CNNs became mainstream thanks to their performances, researchers developed other CNN configurations that are useful for tasks such as object detection or image segmentation. Fully Convolutional Network (FCN) (Long et al., 2015) soon appeared as a variant that replaces the fully connected layer with convolutional operations. It is popular for both object detection and image segmentation. U-shaped FCN (Ronneberger et al., 2015) architectures are the most known models for the latter task. They are composed of both a downsampling and an upsampling path. The downsampling path, called the encoder, encapsulates the most important features for the segmentation using convolutions and pooling operations. And the upsampling path, called the decoder, uses the resulting vector of the previous path to reconstruct the image. The result will be segmented by having its pixels classified into different categories.

1.3.2 Recurrent neural networks

Recurrent Neural Networks (RNN) (Williams and Zipser, 1989) are neural networks capable of analyzing streams of data. This is important in several applications where the output depends on previous computations, such as natural language processing, speech recognition, or stock forecasting. A famous example to illustrate this is the task of predicting the next word in a sentence. When making a prediction, the system doesn't rely only on the last word it has seen but also keeps in mind the previous ones contained in the stream of words. This simulation of memory is possible due to a loop contained in the architecture. And with that, a previous context (called hidden state) can be added to the current input to give more information for the prediction. A general structure of an RNN is illustrated in Figure 1.5, as well as the function composing an RNN block in Figure 1.6.

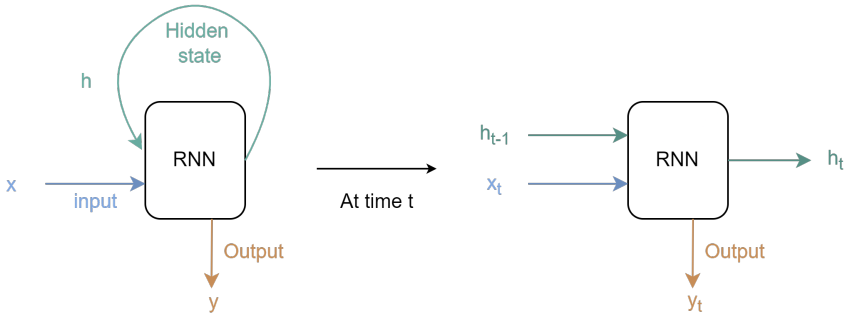


Figure 1.5: RNN architecture.

RNN blocks were soon replaced with more sophisticated blocks such as Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Units (GRU) (Cho et al., 2014) that have a better capacity at retaining information, especially in sequential data where important events can be relatively far from the current input.

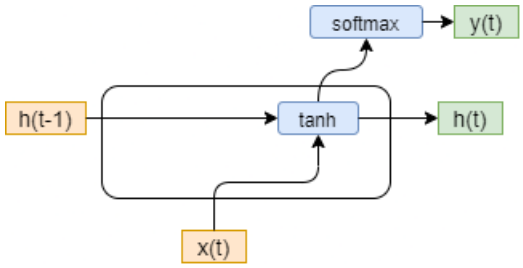


Figure 1.6: composition of an RNN block.

1.3.3 Autoencoders networks

Autoencoders (AE) (Kramer, 1991) are neural networks traditionally used for dimensionality reduction or data compression. The architecture of an AE has an input layer, an output layer of the

same size, and internally, a hidden layer h with a much smaller dimension. The network attempts to compress the data in the encoder part (from the input to the hidden layer h) and then reconstructs it with the decoder (from the hidden layer h to the output). To achieve a good data compression, the goal is to obtain a decoder that recreates a good representation of the input, using only the low-dimensional vector in the hidden layer created by the encoder.

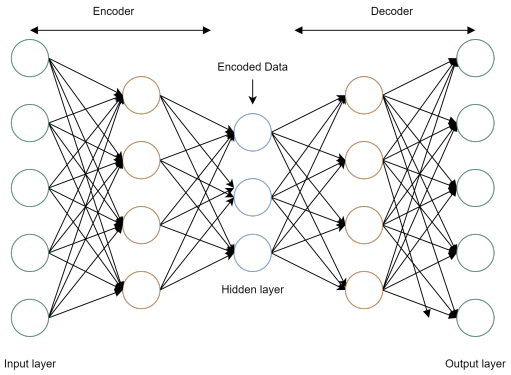


Figure 1.7: Autoencoder.

Many variations of AE exist for tasks other than compression. Denoising AE (Vincent et al., 2008) for example, can be used to extract features from a noisy dataset by keeping the essential information from data and automatically discarding the noise from it. Variational AE (Kingma and Welling, 2014), can recreate the data with variations that can be useful in tasks such as data augmentation.

1.3.4 Restricted Boltzmann machines and Deep belief networks

Restricted Boltzmann Machines (RBM) (Salakhutdinov et al., 2007) are models capable of estimating the probability distribution of the input data in a stochastic way. They are composed of two layers, one layer of visible variables/nodes to represent observable data and one layer of hidden variables to capture dependencies of the visible variables. They are usually used as building blocks for Deep Belief Networks (DBN) (Hinton et al., 2006), which are other known generative models that can be also used in supervised tasks such as classification. Figure 1.8 illustrates a DBN composed of stacked Restricted Boltzmann machines.

1.3.5 Generative adversarial networks

Another generative model that has gained a lot of popularity in recent years is the Generative Adversarial Network (GAN) (Goodfellow et al., 2014). This model uses two networks pitted against each other to create the most convincing data samples. The first network, called generator (G), creates samples that would resemble the most to the real data. They are then fed to the second network called discriminator (D), which tries to classify the data as being real or fake (generated). The networks are trained simultaneously, where G aims to maximize the probability that D makes

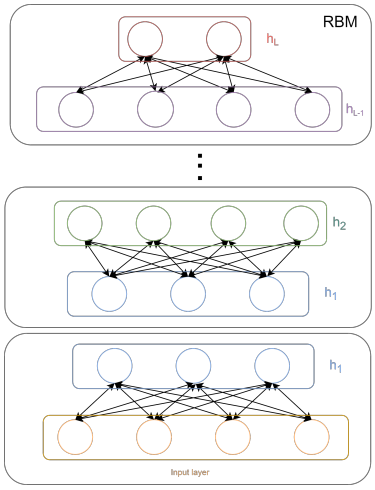


Figure 1.8: DBN architecture.

a mistake while D aims for high classification accuracy (Figure 1.9). GANs are an exciting recent innovation and are behind applications such as DeepFake (Schwartz, 2018), music composition (Elgammal et al., 2017), and art generation (Dong et al., 2018).

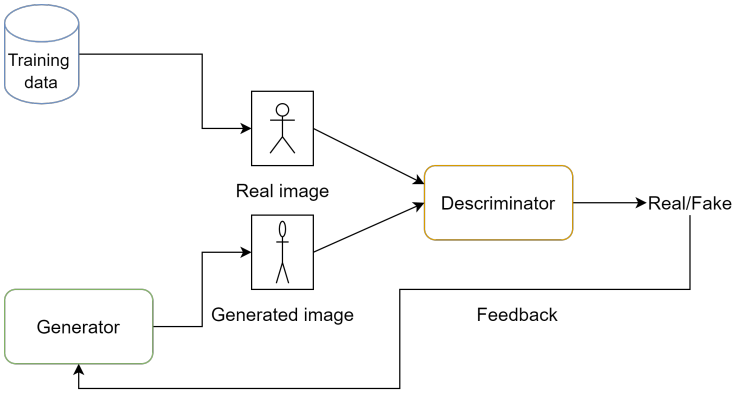


Figure 1.9: GAN representation.

1.3.6 Transformers networks.

Transformers networks, introduced by Vaswani et al. (2017), have become the basic architecture in many state-of-the-art models, particularly in the domain of natural language processing. The architecture includes an attention mechanism, called ”self-attention”, which allows it to focus on different parts of the input data with varying levels of emphasis. This facilitates the model’s ability to understand intricate relations in the data, especially in sequences like text.

The transformer architecture is typically made up of a series of encoder and decoder stacks. Within these stacks are multiple self-attention layers and Feed-Forward Networks (FFN). The at-

tention layers empower the model to weigh the importance of different input data points in relation to each other, rather than in isolation. This is a key divergence from the recurrent or convolutional nature of the architectures mentioned earlier.

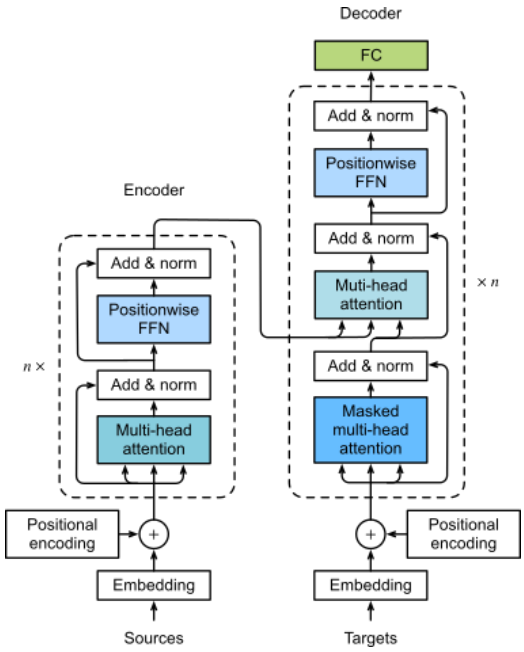


Figure 1.10: Transformer Architecture, figure inspired from (Vaswani et al., 2017).

After its arrival, the transformer architecture produced a series of influential models in NLP. BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2018), GPT (Generative Pre-trained Transformer) (Radford et al., 2018), and T5 (Text-to-Text Transfer Transformer) (Raffel et al., 2020) are among the noteworthy derivatives. Each of these models, based on the transformer backbone, has set benchmarks across diverse tasks, ranging from text classification and translation to generative text tasks.

1.4 Deep learning challenges

While architectures like CNNs, RNNs, and Transformers brought advancements in various domains of deep learning, researchers and practitioners continue to push the boundaries of what these models can achieve. They are often faced with many challenges that need addressing to realize the full potential of deep learning.

Data and privacy

To train a reliable deep learning model, a large set of high-quality data is often required. In many cases, this data also needs to be annotated and balanced to be usable for the given task. In certain fields, these requirements are rarely met. Data can be scarce, imbalanced, and heterogeneous. The

reasons for that are numerous, including the difficulty to get hold of certain data, the differences in instruments and practices used for data collection, or the annotation that can be expensive or time-consuming. To address some of these problems, many techniques were developed to augment the data and improve its quality, ranging from data augmentation and synthesis (Shin et al., 2018) to image de-noising techniques (Gondara, 2016) and preprocessing frameworks (Lin and Haug, 2006). Additionally, domain adaptation and transfer learning techniques have been used to leverage data from related domains to improve model performance when training data is limited.

Another significant challenge, especially in sectors like healthcare, finance, and social platforms, is the concern of data privacy. Handling sensitive information comes with both ethical and regulatory constraints. Privacy and data protection are often not just recommended but mandated by laws like the General Data Protection Regulation (GDPR). This has led to an increasing focus on AI privacy. The field includes the development of techniques to train neural networks without exposing or compromising data integrity. Examples of such techniques are federated learning (Yang et al., 2019), where models are trained across multiple devices or servers without centralizing data, and differential privacy (Abadi et al., 2016), which adds a layer of randomness to data queries, ensuring individual data points cannot be identified.

Computational resources

The time required for training modern DL models represents a significant challenge. Depending on model complexity and dataset size, training can take days or even weeks, leading to higher electricity costs and increased carbon footprints. The prolonged training process slows down research progress and the implementation of models in real-world applications.

In response to these challenges, researchers are exploring solutions to minimize training time. Techniques such as model pruning (Molchanov et al., 2016), where the size of networks is reduced without significantly impacting performance, knowledge distillation (Hinton et al., 2015), where smaller models are trained to mimic larger ones, or new training techniques to reduce computational demands are being developed (Micikevicius et al., 2017). Designing resource-efficient models is also a prominent research field, it includes designing networks that can run on edge devices with very low computational power.

Many efforts are made to decrease the need for computational resources and it is becoming increasingly important with the raising costs of GPUs and the quick widespread of large language models that are specifically resource-demanding.

Explainability

Explainability is another significant challenge in deep learning. The *black box* nature of neural networks makes it nearly impossible to interpret or understand the model's output. This delays the adoption of this technology in areas where interpretability is crucial, such as healthcare, finance, and the judicial system. Practitioners need to understand the decision so they can argue with or against the result of the AI system.

To address this issue, two approaches are currently being explored: the first is to explain existing black box models using different techniques such as linear approximations, saliency maps, class activation functions, etc (Arrieta et al., 2020). The second approach is to design new neural networks that are inherently interpretable (Chen et al., 2018). These systems usually provide an explanation alongside their prediction.

Despite these advances, explainability remains a challenging problem. Striking a balance between model complexity, which often correlates with performance, and model interpretability is an ongoing struggle in the deep learning community.

Architecture design

The architecture of a deep learning model plays a crucial role in its performance. The architecture determines how the network will process input data, how it will represent learned information, and its capacity to generalize to unseen data. A design that works well for one task or dataset might not be optimal for another, and designing an architecture for a specific problem is challenging and resource-intensive. The choice of layers, their connections, and their hyperparameters can significantly impact a model. The number of possibilities is also enormous, even for modest-sized networks. For example, for a simple network of 10 layers, where each layer can take 3 different options (either their type, activation functions, or other hyperparameters) the total number of possibilities reaches almost 60 thousand, which is impractical to sort by exhaustive evaluations.

This task has for a long time been a manual effort guided by trial and error. Architecture engineers can take several months to design a good architecture. Since it is impossible to try all possibilities, they mostly rely on intuition gained through years of experience. The challenge is further amplified by the evolving nature of the field. As new techniques and layers are developed, the potential design space expands, making it even more difficult to find optimal architectures manually. To alleviate this problem, Neural Architecture Search (NAS) is a prominent field that includes techniques to find effective architectures automatically.

1.5 Conclusion

In this chapter, we have laid the foundation of deep learning so the reader can have a comprehensive background for this thesis. We started by presenting the building blocks of neural networks, followed by its training and evaluation process. Moreover, we presented famous types of neural networks and discussed how they function. Lastly, we presented some of the most prominent challenges deep learning faces nowadays. Amongst those challenges, a notable one is presented in architecture design. A task that is getting more impractical in modern DL models. With the increasing size and complexities of the architectures, modern solutions are required to handle it. A research field that tackles this problem is neural architecture search, a solution that promises to alleviate the manual task of architecture design. In the next chapter, we will present in more detail this concept and lay the ground for our future contributions to this field.

Chapter 2

The Tale of Neural Architecture Search

"The future of machine learning is automation."

— Yoshua Bengio

2.1 Introduction

Deep learning has transformed countless domains in recent years, but despite its success in a wide range of applications, using this technology effectively has faced some challenges. As we have seen in the previous chapter, the design of the neural network architecture poses a significant one. This task is crucial in developing deep learning models. It involves choosing a broad spectrum of architectural features and their associated parameters and selecting them to suit a specific task and dataset. Neural network architectures that are tailored to their task and dataset lead to better model performances, as the most suitable architecture for one case might not necessarily be the best fit for another. This process takes up a significant effort given the huge number of possibilities. Researchers and engineers have done this manually for decades. Although it led to state-of-the-art models in many deep learning applications (Alom et al., 2019), this procedure is driven by trial and error and intuition acquired through years of experience. It is both time and resource-consuming and can sometimes lead to sub-optimal architectures that underperform or are computationally expensive.

Neural Architecture Search (NAS) offers an automated solution to network design. It has the potential to reduce the time and cost of developing deep learning models and can help democratize deep learning by allowing non-experts to use and apply them to their problems. NAS belongs to the wider field of Automated Machine Learning or AutoML (He et al., 2021c), a domain that encloses all the techniques that automate the process of machine learning engineering.

In this chapter, we learn more about NAS and its inner workings. We first introduce the concept of AutoML and its connection to our topic, before explaining neural architecture search. We then propose a formulation of NAS in the optimization taxonomy. Moreover, we present in detail the different components defining the NAS framework and its famous benchmarks from the literature.

2.2 Automated machine learning

Building a machine learning model requires going through many steps. Starting from data collection and preprocessing all the way to model deployment and monitoring. This set of tasks requires a variety of specialized skills and resources. AutoML is a set of techniques and tools that automate the process of building and deploying machine learning models. The goal of AutoML is to enable non-experts to build high-quality machine learning models without requiring extensive knowledge of ML algorithms, data preprocessing, or model optimization. AutoML is composed of several subfields, each addressing different aspects of the machine learning pipeline. This pipeline is composed of several stages as seen in Figure 2.1 and usually goes as follows: Initially, relevant data is collected from various sources in the **Data Collection** phase. This data is then cleaned, prepared, and possibly transformed during **Data Processing** to make it suitable for a machine learning model. In the **Feature Engineering** phase, new features are created or extracted from the existing data to improve the model's performance. The model is then designed in the **Model Development** stage, where a suitable ML algorithm or deep learning model is defined and its parameters are chosen. The model's performance is evaluated in the **Model Evaluation** phase, using metrics appropriate to the task. Once it passes this phase, it is deployed into production in the **Deployment** stage. Lastly, the deployed model is continuously monitored to ensure it continues to perform well and to identify when it may need updating or retraining. This process is iterative, often requiring backtracking to previous stages based on ongoing evaluations and changes in data or requirements.

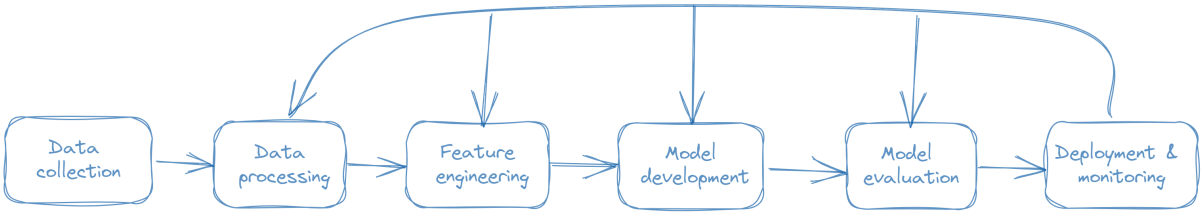


Figure 2.1: The basic machine learning pipeline.

AutoML can interfere in automating one or several stages of this pipeline as well as other peripheral tasks. In the following, we briefly explain the most relevant of its techniques. For a more detailed description of the tools and techniques found in this field, the reader can refer to the survey by He et al. (2021c).

- **Automated data preprocessing:** data preprocessing involves cleaning, transforming, and normalizing raw data to make it suitable for machine learning algorithms. This can include handling missing values, encoding categorical variables, and scaling numerical features. This subfield focuses on automating this process which is often a time-consuming and error-prone process in traditional machine learning workflows.

- **Automated feature engineering:** Feature Engineering involves selecting or creating the most relevant features for the task at hand. This can include extracting text features, creating

new features through mathematical transformations, or using domain knowledge to engineer features. The goal of feature engineering is to improve the performance of machine learning models by providing them with more relevant and informative data. Automated feature engineering techniques can help streamline this process by automatically selecting relevant features, extracting features from text or image data, and transforming the data into a format that is suitable for machine learning.

- **Automated model selection:** Different machine learning algorithms perform differently on different types of data, and selecting the best algorithm for a given task requires expertise and trial and error. Automated model selection techniques can automate this process by exploring a range of machine learning algorithms and selecting the best one for a given task, based on the models' performance metrics.
- **Hyperparameter optimization:** Hyperparameters are the configuration settings for machine learning models that are not learned from the data, such as the learning rate, regularization parameters, and activation functions. Hyperparameter tuning involves selecting the best values for the model hyperparameters, which can significantly impact model performance. Hyperparameter optimization techniques can automate the process of selecting and tuning these parameters, in order to optimize the performance of the model.
- **Meta-learning:** Meta-learning involves using machine learning algorithms to learn how to optimize other machine learning algorithms. For example, a meta-learning algorithm could learn how to select the best hyperparameters for a given dataset and model architecture or learn how to combine different models to achieve better performance.
- **Model Compression:** Machine learning models can be large and complex, which can make them difficult to deploy on resource-constrained devices such as smartphones or embedded systems. Model compression automates the process of reducing the size and complexity of a model, while maintaining its performance, by techniques such as pruning, which removes unimportant connections in a neural network, or quantization, which reduces the precision of the weights and activations in a network.
- **Neural Architecture Search (NAS):** Designing and optimizing neural network architectures is a complex and time-consuming task that requires expertise in deep learning. Neural architecture search techniques can automate this process by exploring a large search space of possible neural network architectures and identifying the optimal architecture for a given task. NAS has a significant overlap with hyperparameter optimization and meta-learning. It is the main focus of our chapter and is presented in more detail in the following sections.

2.3 Neural architecture search

Neural architecture search refers to the automated process of searching for the best neural network architecture for a specific task. Instead of manually designing a neural network, NAS automates

this process by exploring a vast space of possible architectures and identifying the ones that perform best for a given dataset or task. This technique first appeared more than three decades ago (Tenorio and Lee, 1988) but only became popular in the last few years due to its increasing importance and usability. NAS can potentially discover novel architectures that outperform manually designed ones, leading to better performance in modern tasks. NAS is particularly useful for newer tasks or datasets where standard architectures might not be optimal. The process behind NAS can be formulated as an optimization problem, where the goal is to find the optimal neural network architecture that maximizes a performance metric, such as accuracy, while potentially minimizing other factors, such as computational complexity or latency. The size of the feasible solutions can be very large in NAS, reaching an unbounded number of possibilities in certain cases.

2.3.1 NAS as an optimization problem

The optimization problem for NAS can be defined with the following components:

- **Objective function:** The objective function quantifies the performance of a neural network architecture. Commonly, the performance metric is the validation accuracy (or its equivalent) on a given task or dataset. In some cases, additional factors, such as model size or inference time, can be included in the objective function to encourage the discovery of efficient architectures.
- **Decision Variables:** The decision variables describe the possible neural network architectures that can be considered during the search process. They are typically represented as a discrete, combinatorial space encompassing various architectural components, such as layer types, layer sizes, activation functions, and connectivity patterns. In the context of NAS, it is usually referred to as the *search space*.
- **Constraints:** Constraints may be imposed on the optimization problem to ensure that the discovered architectures meet specific requirements, such as computational resource limitations, latency, or model size constraints. These constraints help guide the search process towards architectures that are not only accurate but also efficient and practical for the intended application.
- **Optimization algorithm:** The optimization algorithm is responsible for efficiently exploring the search space and identifying high-performing architectures that satisfy the constraints. Several optimization techniques have been proposed for NAS, including evolutionary algorithms, reinforcement learning, Bayesian optimization, gradient-based methods, and others. In NAS terminology, these techniques are often referred to as *search strategies*

Given these components, the NAS optimization problem can be formally stated as finding an architecture A* in the search space S that maximizes the objective function f(A) subject to the constraints C.

$$A^* = \arg \max_{A \in S \text{ subject to } C} f(A)$$

The main challenge in this optimization problem is foremost to define the search space of possible architectures. This search space is often vast and complex which makes exhaustive search infeasible. Consequently, the second important step is to define the search strategy to efficiently explore the search space and identify high-performing architectures while respecting the given constraints. This process involves sampling and evaluating architectures. Training DNNs for evaluation can be time-consuming. It is considered to be the bottleneck of neural architecture search as it can sometimes take several hours for a single evaluation. For this reason, techniques for estimating the performance of DNN were developed.

With these puzzle pieces in mind, we can highlight the three main components that make up a NAS framework: the search space, the search strategy, and the performance evaluation strategy (see Figure 2.2). Each of these can have a high impact on the results of the framework. We further explain each of these components in the following section.

2.3.2 The building blocks of the NAS framework

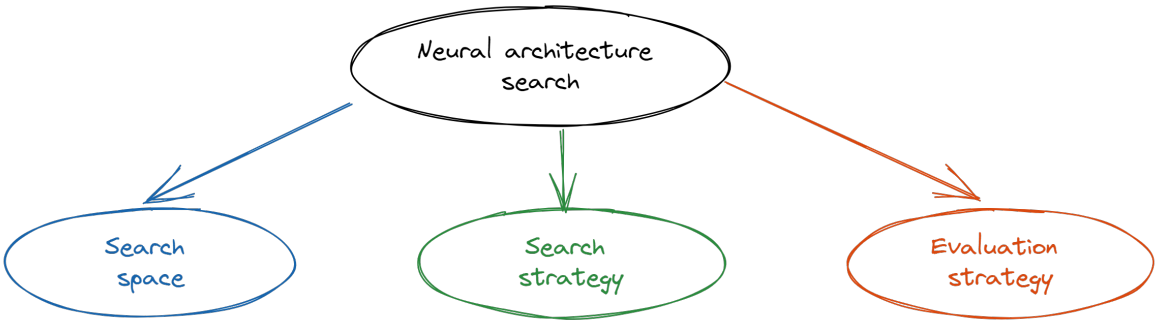


Figure 2.2: The NAS framework

The search space

The NAS search space defines the characteristics of neural architectures that can be considered during the search process. In other words, it characterizes the set of possible solutions. The choice of the search space has a critical impact on the final performance of the NAS algorithm. It directly determines the NAS algorithm's upper-performance limit to some extent. Depending on its design, it can include a wide range of architectural choices, such as the number and size of layers, types of activation functions, and connectivity patterns. The more flexibility is given by the search space, the more freedom the NAS process has in discovering novel networks. However, allowing too much freedom of design dramatically increases the size of the search space, and consequently, the computational time required to find efficient architectures. Therefore, a trade-off needs to be established between size and flexibility.

In the literature, different search spaces have been proposed with various sizes and design principles. Three main types of search spaces can be extracted: the global search space, the cell-based search space, and the hierarchical search space (Wistuba et al., 2019; Elsken et al., 2019).

1. **The global search space:** The less constrained type of search space is the global search space. It allows the NAS process to search for architectures defined in a fine-grained manner. It gives the freedom to choose every operation and connection in the neural network. The chain-structured search space is the most known example of a global search space. Here, the architectures are formed by stacking a number of layers, with the input of a layer being the output of the previous layer. To define such search space, we need to specify certain elements such as the maximum number of layers, the type of possible operations executed by each layer (such as pooling, convolution, etc.), and the associated hyperparameters for each operation (such as the number of filters, kernel size, strides, etc.). It's worth noting that the hyperparameters are dependent on the type of operation specified. For example, in (Baker et al., 2016), the authors use this chain-structured search space to build a convolutional neural network. The search space allows them to choose the type of operations of each layer with its associated hyperparameter settings such as the number of filters, kernel size, stride, and pooling size.

   Other design choices can be added to broaden this type of search space, such as branches or skip connections (He et al., 2016). These allow information to flow from one layer of the network to another layer of the network that does not directly follow it. Figure 2.3 represents two examples of architectures that can result from a global search space. On the left, is an architecture that can result from a standard chain-structured search space. On the right, is an architecture that can result from a search space that allows compositions using branches and skip connections.

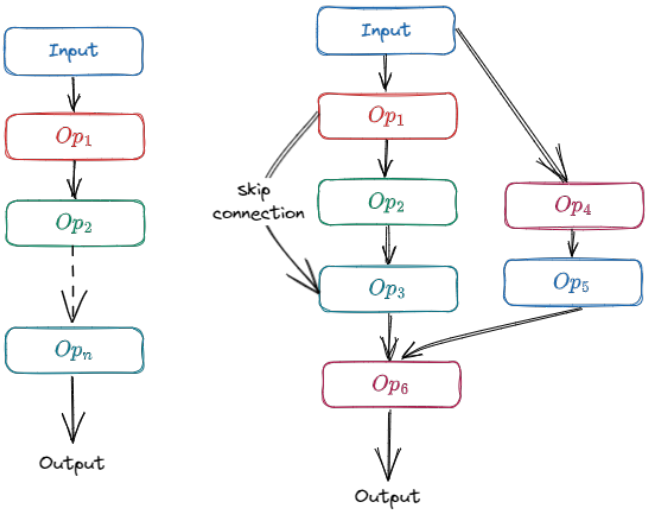


Figure 2.3: Examples of two architectures from a global search space

Global search spaces allow the discovery of novel networks, however, having such freedom can result in longer times to find effective networks. Some constraints are added in practice to make it more manageable. For example, discarding the evaluation of certain networks that are known to perform badly (e.g., removing architectures that start with pooling operations in CNNs), or are too computationally expensive. Similarly, enforcing choices on segments of the network can generate solutions with good starting performances which can lead to shorter search times. Nonetheless, global search spaces are still considered computationally expensive and challenging to optimize due to their high dimensionality. To address this issue, a more structured approach was proposed which is the cell-based search space.

2. **The cell-based search space:** In the cell-based search space, architectures are constructed from smaller sub-structures often called cells or blocks, or modules. They are stacked together in a predetermined manner to form a larger network. This design approach is based on the observation that many well-performing human-designed models are built by repeated instances of fixed structures. The problem of searching for a full neural architecture is simplified into searching for these structures. The search space becomes much smaller since cells typically contain fewer layers than complete architectures. Figure 2.4 illustrates the structure of architectures created from a cell-based search space. In this example, two types of cells are stacked repeatedly, with each of them constructed differently.

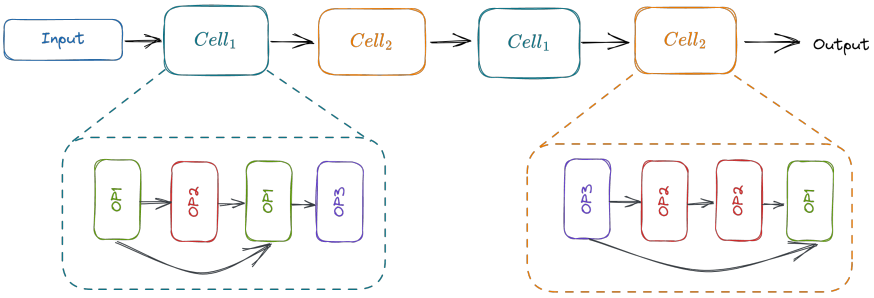


Figure 2.4: Example of a generic architecture in a cell-based search space

An example of a notable early work that explores the cell-based search space can be found in (Zoph et al., 2018). It uses two types of cells in neural architecture search: normal cells and reduction cells. Normal cells extract advanced features and keep spatial resolution unchanged, while reduction cells reduce spatial resolution. The final architecture is formed by connecting several normal cells, followed by a reduction cell, and repeating this pattern. The search process must find the cell structures of each type that makes the most performing architectures. Notable speed-up has been achieved with this type of search space. For instance, Zoph et al. (2018) reported better performance while estimating a seven-fold speed-up compared to their prior work. Along with these improvements, the approach allows for easy generalization across different datasets, as the discovered architectures constructed from cells can be flexibly stacked to build larger or smaller networks. Zoph et al. (2018)

demonstrated this adaptability by transferring cells optimized for CIFAR-10 to ImageNet, adjusting the number of cells and filters within a model, and still achieving state-of-the-art performance.

Although it gained a lot of popularity, the cell-based search space suffers from the limited diversity of the explored architectures. Since it primarily focuses on the cells within the architecture, it overlooks other important factors that can impact the model's performance, such as the overall network topology or the interaction between cells. A more advanced search space design known as the hierarchical search space tries to solve this problem.

3. **The hierarchical search space:** The hierarchical search space is an approach that supports complex topologies in NAS. By breaking down the architecture into different levels of abstraction, it simplifies the search process. Each level contains a set of building blocks, which can be either primitive operations or more intricate structures like cells in a cell-based search space. This organization allows researchers to focus on optimizing structures at each level of abstraction, which are then combined to create the overall architecture. Initially, a small set of primitives, such as convolutional and pooling operations, forms the bottom level of the hierarchy. Higher-level computation graphs or motifs are constructed using lower-level motifs as their building blocks. At the top of the hierarchy, these motifs are stacked multiple times to generate the final neural network. Figure 2.5 illustrates how different motifs can be constructed at each level. In level one, simple operations are assembled to form a first-level motif. In level two, these motifs (or cells) are in their turn assembled to form higher-level structures.

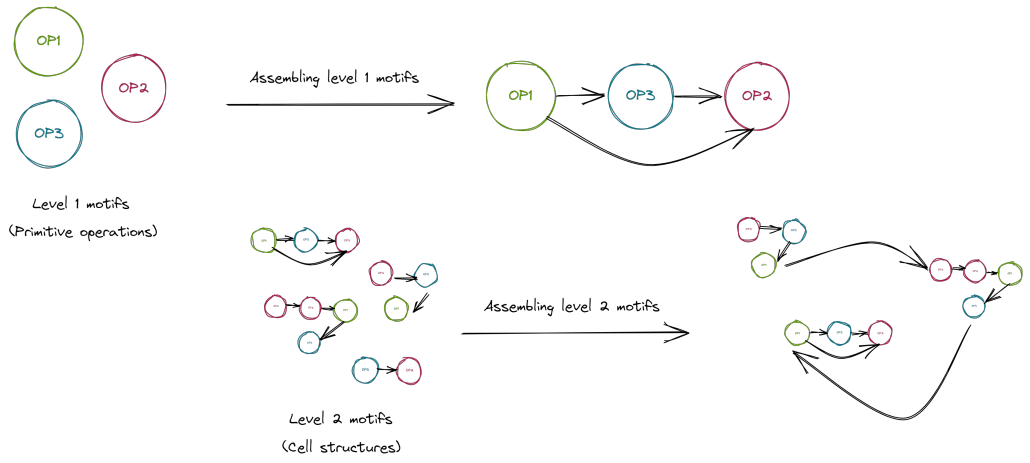


Figure 2.5: Constructing motifs in the hierarchical search space

The hierarchical search space has been employed in various NAS methods demonstrating its potential for discovering high-performance neural network architectures. In (Liu et al., 2017a) for example, authors use an evolutionary algorithm to discover hierarchical archi-

tectures by optimizing the components within each level and combining them to form the overall architecture.

Hierarchical search spaces represent a good trade-off between global and cell-based search spaces. However, this type of search space leads to a more complex search process. To handle this, researchers need to carefully choose the number of levels of abstractions in their search space.

The search strategy

The search strategies employed in NAS determine how the algorithms navigate through the search space and make decisions about which architectures to evaluate. These strategies differ in terms of their exploration, exploitation, and optimization techniques. The most prominent search strategies include:

1. **Evolutionary Algorithms (EAs):** Evolutionary algorithms are a class of optimization techniques inspired by the principles of natural selection and evolution. In the context of NAS, EAs have been employed to discover optimized neural network architectures by evolving a population of architectures over multiple generations. These methods aim to gradually improve the performance of the architectures in the population by favoring those with higher fitness, typically measured as the validation performance (e.g., accuracy, F1 score) on the target task. The main components of an EA-based NAS method are:

    - **The encoding scheme:** Determines how the neural network architectures are represented within the evolutionary algorithm. Encoding translates architectural configurations into genotypic representations that can be manipulated by the genetic operators.
    - **The population:** which represents a set of neural network architectures.
    - **The fitness function:** it quantifies the performance of an architecture, usually based on its performance on the target task.
    - **The selection mechanism:** the method for selecting architectures from the population based on their fitness values (e.g., tournament selection, rank-based selection, or truncation selection).
    - **The genetic operators:** Functions that create new architectures by modifying existing ones, such as mutation (e.g., adding or removing layers, changing activation functions) and crossover (e.g., combining parts of two-parent architectures to create a new architecture).

    In the literature, there are several examples of EA-based NAS methods for improving neural networks. Genetic CNN (Xie and Yuille, 2017) is an early work that applies a genetic algorithm to evolve convolutional neural network (CNN) architectures for image classification, using mutation and crossover operators. It has demonstrated competitive performance on datasets such as CIFAR-10 and a subset of ImageNet from ILSVRC2012 (Deng et al.,

2012). Large-Scale Evolution (Real et al., 2017) is another EA-based NAS method that evolves a population of architectures for image classification tasks, combining tournament selection, mutation, and crossover with techniques like aging and transfer learning to enhance search efficiency and scalability. Regularized Evolution (Real et al., 2019) is another notable work that employs an age-based tournament selection mechanism as an EA-based NAS method. In subsequent research, several works continued to leverage evolutionary algorithms for advancing NAS methods. In "Evolving Deep Neural Networks", Miikkulainen et al. (2019) utilized evolutionary strategies to optimize various aspects of deep neural networks, including their topology, connections, and weights, demonstrating success on tasks like object recognition and playing Atari games. Another work, "NeuroEvolution of Augmented Topologies" (NEAT) (Stanley and Miikkulainen, 2002), employs a genetic algorithm to evolve both the weights and architectures of neural networks, introducing concepts like speciation and genetic encoding to handle different topologies. Moreover, "Hierarchical Representations for Efficient Architecture Search" (Liu et al., 2017a), utilizes hierarchical genetic representations in its evolutionary algorithm to improve the efficiency of the architecture search process.

EA-based NAS methods have proven effective at discovering high-performing neural network architectures for a wide range of tasks. However, they can be computationally expensive due to the need to evaluate and train a large number of architectures.

2. **Reinforcement Learning (RL):** RL-based methods tackle the problem of discovering optimal neural network architectures by formulating it as a sequential decision-making task. In this approach, an RL agent generates architectures by making a series of decisions, interacting with the search space, and receiving feedback in the form of a reward signal (e.g., validation accuracy). The RL agent's policy is then optimized using RL algorithms to maximize the expected performance of the generated architectures. The general framework for RL-based NAS includes:

    - **The action space:** determines the set of possible actions the RL agent can take at each step, such as selecting a specific layer type or operation.
    - **The state space:** represents the current status of the architecture being generated, including previously selected actions.
    - **The controller or agent:** usually a neural network (e.g., a Recurrent Neural Network), responsible for generating architecture configurations.
    - **The agent's policy:** A function that maps states to actions. The RL agent uses it to decide which action to take given the current state.
    - **The reward signal:** measures the performance of the generated architecture on a given task (e.g., validation accuracy) and provides feedback to the RL agent. This function helps guide the agent towards architectures with better performance.

- **The RL algorithm:** An algorithm to train the agent by optimizing its policy. This involves generating architectures, evaluating their performance, and updating the agent's parameters based on the observed reward. Common RL algorithms include policy gradients (e.g., REINFORCE), Q-learning variants (e.g., DQN), and actor-critic methods (e.g., A3C).

A pioneering work that presents an RL-based NAS method can be found in (Zoph and Le, 2016). It utilizes a recurrent neural network controller to generate architectures. The method applies the REINFORCE policy gradient algorithm to optimize the controller, demonstrating competitive performance on datasets such as CIFAR-10 and Penn Treebank. In (Pham et al., 2018), authors build upon the RL-based NAS approach and introduce weight sharing among the generated architectures to reduce the search cost. By sharing weights among similar architectures, this work significantly reduces the training time required for each architecture, leading to faster convergence of the search process. Baker et al. (2016) employed Q-learning, a value-based RL approach, for discovering optimized convolutional neural network architectures. Meanwhile, "Progressive Neural Architecture Search" (PNAS) (Liu et al., 2018a) leverages a sequential model-based optimization strategy, incorporating a predictor model to guide the search process in a sample-efficient manner.

RL-based NAS methods have shown promising results in discovering high-performing neural network architectures for a wide range of tasks. However, they often require significant computational resources to evaluate and train the generated architectures.

3. **Bayesian Optimization (BO):** Bayesian optimization is a popular global optimization technique that has been applied to neural architecture search to efficiently explore the space of possible architectures. BO-based NAS methods typically model the relationship between architectures and their performance using probabilistic models, which provide uncertainty estimates that help guide the search process. BO-based NAS methods are usually composed of:

    - **A surrogate model:** A probabilistic model that captures the relationship between architectures and their performance. Common surrogate models include Gaussian processes and Tree-structured Parzen estimators.
    - **An acquisition function:** A function that quantifies the utility of exploring a particular architecture based on the predictions and uncertainties provided by the surrogate model. Popular acquisition functions include Expected Improvement, Upper Confidence Bound, and Probability of Improvement.
    - **The optimization procedure:** An algorithm for optimizing the acquisition function to identify the next architecture to evaluate.

A few works in the literature employed Bayesian Optimization for NAS. For example, Liu et al. (2018a) propose a method for learning convolutional neural network structures

through a sequential model-based optimization. Their approach is progressive, starting with the simplest models and progressively expanding the search space.

BANANAS (White et al., 2021) is another Bayesian optimization method that employs a graph convolutional network as a surrogate model to predict the performance of architectures. By using graph-based representations, BANANAS captures the structural information of the architectures, leading to improved search efficiency. Additionally, Cai et al. (2018) integrate Bayesian optimization with gradient-based methods, allowing for a flexible, hybrid search strategy that leverages the best of both worlds: the global, probabilistic search of BO and the local, gradient-guided exploration of differentiable methods.

BO-based NAS methods have shown their effectiveness in discovering high-performing architectures with relatively low computational costs compared to other search strategies. However, they can be sensitive to the choice of surrogate models, acquisition functions, and optimization procedures.

4. **Gradient-based Optimization:** Gradient-based optimization methods have been introduced to neural architecture search to enable efficient search for high-performing neural network architectures. These methods reformulate the architecture search problem as a continuous optimization task by relaxing the discrete architectural decisions, which allows the use of gradient-based optimization algorithms. The main components of such methods are:

    - **The continuous relaxation:** To apply gradient-based optimization techniques, the architecture search space must be converted from a discrete to a continuous domain. Methods for continuous relaxations include the use of softmax functions (Goodfellow et al., 2016), mixed-operation representation (Liu et al., 2018b), or Path-level relaxation (Veniat and Denoyer, 2018).
    - **Differentiable architecture representation:** This results from the relaxation of the search space, the differentiable representation of the architecture allows the calculation of gradients with respect to the architecture's parameters.
    - **The objective function:** Similar to the other approaches, this function evaluates the performance of an architecture, usually based on its validation performance (e.g., accuracy, F1 score) on the target task.
    - **Gradient-based optimization algorithm:** an algorithm that updates the architecture parameters using gradient information computed from the objective function. Common algorithms include gradient descent (Ruder, 2016), stochastic gradient descent (Bottou et al., 1991), and adaptive gradient methods (e.g., Adam (Kingma and Ba, 2014), RMSProp (Tieleman et al., 2012)).

    Several gradient-based optimization NAS methods have been proposed in the literature: DARTS (Liu et al., 2018b) introduces a continuous relaxation by using a softmax function to represent the probabilities of selecting different operations (e.g., convolution or pooling)

at each layer. By defining a set of learnable architecture parameters, this work allows the use of gradient descent to optimize the continuous representation of the architecture. In ProxylessNAS (Cai et al., 2018), authors use a differentiable approach to search for hardware-aware neural architectures. The method introduces a path-level binary gating mechanism that allows efficient computation of gradients and enables the direct optimization of architectures on the target hardware (e.g., mobile devices). Authors in (Xie et al., 2018) extend the DARTS framework by incorporating stochastic optimization techniques to enhance search efficiency. Their method uses Gumbel-Softmax relaxation to sample architectures during training. In FBNet (Wu et al., 2019), authors employ a differentiable masking technique and leverage a hardware-aware loss function to guide the search towards architectures that are not only accurate but also adhere to hardware efficiency constraints. Similarly, Stamoulis et al. (2019) employ a one-shot model to facilitate a computationally efficient and memory-friendly differentiable search, utilizing a uniform sampling strategy for pathway selection. In "Latency-aware Differentiable Architecture Search" (Xu et al., 2020a), authors integrate latency considerations into a differentiable search framework, aligning the architecture optimization process with practical deployment constraints.

Gradient-based NAS methods have shown the ability to find high-performing architectures with reduced computational cost compared to other search strategies, such as reinforcement learning or evolutionary algorithms. However, these methods can be sensitive to hyperparameters, initialization, or local optima, which may affect the quality of the discovered architectures.

The evaluation strategy

In neural architecture search, performance evaluation can be time-consuming and computationally expensive. Recent research has developed methods that can reduce the cost of performance evaluation. The most common ones are:

1. **Lower fidelity estimates:** Lower fidelity estimates in neural architecture search are techniques to approximate the performance of candidate neural network architectures using computationally cheaper methods. For example:

    - **Learning curve extrapolation:** In this technique, architectures are trained for only a few epochs, and the learning curves are extrapolated to estimate the final performance. This idea is explored in the work by (Klein et al., 2017). Authors use a Bayesian Neural Network to estimate the learning curve based on the observed performance of architectures trained on a few epochs.
    - **Subset of the dataset:** Some NAS methods evaluate architectures on a smaller subset of the dataset to reduce the training and evaluation time. For example, in the work by Zoph et al. (2018), the authors used a smaller version of the ImageNet dataset, containing only 1000 images per class, to evaluate architectures during the search phase.

- **Downsizing:** Another approach is to downsize the input data by lowering the resolution or using fewer modalities from the input. For example in (Gessert and Schlaefer, 2019b), the authors used lower dimensional data to search for the most effective network for image segmentation.
- **Fewer training epochs:** NAS methods may also reduce the number of training epochs to save time during the architecture search process. At the end of the search, the best networks are fully trained to get the final performances.

2. **Weight inheritance:** Weight inheritance is a technique that reuses or transfers weights from previously trained architectures to newly generated ones during the search process. This method accelerates the training of new architectures, allowing the NAS algorithm to explore the search space more efficiently. Weight inheritance is often applied in combination with other NAS techniques, such as evolutionary algorithms, where new architectures are generated through mutation operations. In these cases, weight inheritance can be leveraged to reduce the training time required for the offspring architectures by initializing their weights from their parent architectures. One example of weight inheritance in NAS is presented in (Real et al., 2019) where authors employed an age-based tournament selection mechanism for evolving a population of architectures. During the search process, the weights of the parent architecture were inherited by the offspring. This weight inheritance allowed the offspring to start training from a better initialization point.

3. **Network Morphism:** The idea behind network morphism is to modify an existing trained neural network architecture into a new architecture while preserving the functionality and performance of the original network as much as possible Wei et al. (2016). This is achieved by applying certain morphing operations that maintain the previously learned knowledge and allow the new network to start its training from a better initialization point using the previous weights, ultimately reducing the training time and computational cost. An example of a NAS method that utilizes network morphism is (Elsken et al., 2018). The method generates child networks through mutation operators, guided by principles of Lamarckian inheritance. It utilizes Network Morphisms (NM) and Approximate Network Morphisms (ANM) for mutation. NM preserves the functionality of the network, ensuring the child has the same initial error as its parent. On the other hand, ANM provides more architectural exploration and the possibility of reducing network size, an important factor in multi-objective search.

4. **One-shot models:** One-shot models in NAS aim to accelerate the architecture search process by training a single, large model containing many candidate architectures.

   One of the most well-known one-shot NAS methods is the Efficient Neural Architecture Search (ENAS) proposed by Pham et al. (2018). In ENAS, a large ”supernet” is constructed, which contains many neural network architectures as its subgraphs. The key idea is that many architectures within the search space can share weights with each other, reducing the training time for each individual architecture. During the search phase, ENAS trains the su-

pernet and samples architectures from it, reusing the weights already learned by the supernet to evaluate the sampled architectures.

5. **Surrogate models:** Surrogate models are used to provide a computationally efficient approximation of the true performance of candidate architectures. They are trained using a limited set of evaluated architectures and their corresponding performance metrics. Based on this training data, the surrogate model can predict the performance of unseen architectures, guiding the search toward promising areas of the search space. There are various types of surrogate models used in NAS, including:

    - **Gaussian Processes (GPs):** Gaussian Processes are non-parametric probabilistic models that provide not only predictions but also uncertainty estimates. In NAS, GPs can be used to model the relationship between architectures and their performance. This type of surrogate model is usually found in conjunction with Bayesian optimization search strategy (Kandasamy et al., 2018).
    - **Random Forests (RFs):** Random Forests are an ensemble learning method that combines multiple decision trees. RFs are used as surrogate models in NAS due to their robustness and ability to handle high-dimensional and noisy data. In (Zela et al., 2018), the authors utilize Random Forests for predicting the joint space of architectures and hyperparameters.
    - **Neural Networks:** Neural networks can also be employed as surrogate models in NAS, learning to predict the performance of candidate architectures based on their structure or encoded representation. A prominent example is found in (Siems et al., 2020), where authors propose a surrogate model based on a feed-forward neural network for predicting architecture performance.

6. **Zero-cost proxies:** It encapsulates all measures that can give an estimate of the performance without any training. Some notable zero-cost proxies include:

    - **Random features:** This approach assumes that randomly initialized networks can capture architectural properties before any training. The performance of a neural architecture is approximated by measuring the similarity between features extracted from the same network but with different random initializations. For example, Saxe et al. (2011) demonstrate that random weights and architectures can often achieve decent performance without training.
    - **Architectural priors:** Architectural priors use the prior knowledge of architecture characteristics and design principles to evaluate the performance of a neural architecture without training. The evaluation is based on how well the architecture adheres to the established architectural patterns. One such measure is presented in (Jiang et al., 2018), where the authors propose a margin-based measure as a proxy for performance. This measure helps assess how well an architecture generalizes by analyzing the distribution of margin values.

2.3.3 NAS benchmarks

The importance of fair comparison, reproducibility, and unbiased evaluation in NAS research is critical to drive progress and innovation in the field. NAS algorithms aim to discover optimal neural network architectures for a given task, but the process can be computationally demanding and time-consuming. Furthermore, variations in training setups, such as learning rate, weight initialization, and random seeds, can lead to inconsistencies in performance and affect the ability to compare different NAS methods. Biases may also arise due to elements such as data augmentation techniques, regularization methods, and dataset handling. These factors can skew the NAS algorithm's ability to choose the optimal model, as the performance comparison may not be based solely on the merits of the architectures themselves.

The purpose of NAS benchmarks is to provide a standardized framework for evaluating and comparing different NAS algorithms. By offering consistent datasets, search spaces, and evaluation metrics, they ensure reproducibility and validation of the results and facilitate fair comparison among NAS algorithms. They also significantly reduce the computational cost and time associated with NAS experiments by providing precomputed performance metrics for a large number of architectures, enabling a broader range of researchers to participate in the field and contribute to its development, even without access to large-scale training systems.

While in many optimization contexts, the word "benchmark" is often synonymous with datasets, in NAS, it not only provides the dataset but also a search space and a structured setting for testing and comparison. Thus, when we refer to a benchmark in NAS, we are referring to both the dataset and the standardized evaluation framework built around it.

Several NAS benchmarks have been proposed in the literature to facilitate the evaluation and comparison of neural architecture search algorithms. A detailed survey about benchmarks can be found in (Chitty-Venkata et al., 2023). Table 2.1 offers a list of prominent NAS benchmarks of different categories. It is followed by a presentation of each of them. Extensive details are given to the benchmarks that we will be using in our studies.

Table 2.1: Summary of famous NAS benchmarks. The highlighted benchmarks will be used in our studies.

Benchmark	**Task**	**Size**	**Dataset(s)**	**Metrics**
NAS-Bench-101	Image classification	423k	CIFAR-10	Val Acc
NAS-Bench-201	Image classification	15k	CIFAR-10, CIFAR-100, ImageNet16-120	Train, Test, Val Loss and Acc
NAS-Bench-301	Image classification	10^{18}	CIFAR-10	Val Acc Train Time Num Params
MacroNAS Benchmark	Image classification	200k	CIFAR-10, CIFAR-100	Val Acc MMACs

NAS-Bench-NLP	Natural language processing	14k	TPT, Wiki-2-Text	Val Perplexity Train Time Num Params
NAS-Bench-ASR	Automatic Speech Recognition	8k	TIMIT	Val, Test Acc Flops. Num Params
NAS-Bench-Graph	Various graph-based applications	26k	9 Graph dataset	Train, Test Acc Latency Num Params

NAS-Bench-101 (Ying et al., 2019): NAS-Bench-101 is the first comprehensive benchmark designed to evaluate and compare neural architecture search algorithms. It provides a fixed search space, a dataset, and performance metrics for the image classification task. Its key components are:

- **Search Space:** NAS-Bench-101 offers a search space of over 420,000 unique convolutional architectures. Each architecture is built upon a fixed backbone starting with a convolution operation and ending with max pooling followed by a fully connected layer. In between, it contains three cells separated by downsampling blocks. Each architecture differs by the choice of cell composition. A cell is comprised of 7 nodes. The first node is the input, and the last node is the output. The remaining five nodes can be either 1×1 convolution, 3×3 convolution, or 3×3 max pooling. The cell can take on any directed acyclic graph structure from the input to the output with at most 9 edges.
- **Dataset:** NAS-Bench-101 focuses on the CIFAR-10 dataset, which consists of 60,000 32x32 color images across 10 classes, with 6,000 images per class. The dataset is divided into a training set of 50,000 images and a test set of 10,000 images. This dataset is widely used for evaluating image classification algorithms due to its manageable size and diverse set of images.
- **Performance metrics:** NAS-Bench-101 provides pre-computed results for all architectures in the search space. The authors trained and evaluated each architecture three times with different seeds and the same training setting (like learning rate) in each run to prevent bias. The networks are trained on the CIFAR-10 dataset for a total of 108 epochs.

NAS-Bench-201 (Dong and Yang, 2020): NAS-Bench-201 is another comprehensive benchmark designed for image classification tasks. It provides a compact search space, multiple datasets, and performance metrics. Its main elements are:

- **Search Space:** The search space of NAS-Bench-201 consists of 15,625 unique convolutional architectures. Each architecture has a fixed backbone comprised of repeated cells. Each cell contains 4 nodes and allows for five possible operations: zero operation (none), skip connection, 1×1 convolution, 3×3 convolution, and average pooling.

- **Dataset:** NAS-Bench-201 provides performance metrics for three image classification datasets, allowing researchers to assess their ability to generalize to different datasets:
  - CIFAR-10: 60,000 32x32 color images in 10 classes, with 6,000 images per class.
  - CIFAR-100: 60,000 32x32 color images in 100 classes, with 600 images per class.
  - ImageNet16-120: A downsampled version of ImageNet (Deng et al., 2012), containing 120 classes and 16x16 resolution images.
- **Performance metrics:** NAS-Bench-201 provides several performance metrics for each architecture in the search space, including training, validation, and test losses/accuracies on the three mentioned datasets after 200 epochs. It also provides the number of parameters of each model.

This benchmark was later extended into NATS-Bench (Dong et al., 2021) with another search space that includes the macro-topology of the architectures. In this search space, the number of channels in each cell of the network can vary, adding another level of complexity and flexibility to the optimization process.

NAS-Bench-301 (Siems et al., 2020): NAS-Bench-301 is another benchmark for evaluating and comparing NAS algorithms in the context of image classification tasks. It was introduced by Siems et al. in 2020 and extends the ideas of NAS-Bench-101 and NAS-Bench-201 by providing an even larger search space, as well as enabling the use of surrogate models to predict architecture performance.

- **Search Space:** It's a convolutional cell-based search space based on DARTS. It consists of two types of cells: convolutional cells, and reduction cells, each type contains six nodes. The architecture stacks k convolutional cells together with one reduction cell. For every cell, the first two nodes are the outputs from the previous two cells in the hyper-architecture. The next four nodes contain two edges as input, creating a DAG. Each edge can take on one of seven operations. Since the DARTS search space is very large (approx 10^{18} architectures) this benchmark only trains a subset of it and uses a surrogate model to predict the remaining architectures.
- **Dataset:** Similar to NAS-Bench-101, this benchmark focuses on the CIFAR-10 dataset.
- **Surrogate Models:** NAS-Bench-301 introduces the use of surrogate models for predicting the performance of architectures without requiring computationally expensive training. By training surrogate models on the precomputed performance metrics of a large number of architectures, it can estimate the performance of unseen architectures during the search process. In this work, authors randomly sampled 60,000 architectures to train various surrogate models. The best-performing surrogates were LGBoost, XGBoost, and Graph Isomorphism Network (GIN).

- **Performance metrics:** NAS-Bench-301 provides training, test, and validation accuracy of each architecture for the CIFAR-10 dataset. It also includes the number of parameters of each architecture. Since only 60k architectures are trained in this benchmark. The accuracy and training time provided are calculated using the surrogate models.

MacroNASBenchmark (Den Ottelander et al., 2021): A benchmark designed specifically for evaluating multi-objective NAS methods.

- **Search Space:** The search space includes over 200,000 unique architectures. The architectures consist of 14 unrepeated cells (x_1 to x_{14}). Each cell x_i can take one of three options (*Identity*; *3_3x3*: MBConv[1] with expansion rate 3 and kernel size 3; *6_5x5*: MBConv with expansion rate 6 and kernel size 5).

- **Dataset:** MacroNASBenchmark focuses on CIFAR-10 and CIFAR-100 datasets presented previously.

- **Performance metrics:** MacroNASBenchmark provides for each architecture precomputed accuracies for the two image classification datasets. It also provides a metric for complexity (the Million Multiply Accumulate Operations MMACs) that can be used as a second objective in multi-objective NAS.

NAS-Bench-NLP (Klyuchnikov et al., 2022): NAS-Bench-NLP is the first tabular NAS benchmark designed specifically for natural language processing tasks. It comprises a combination of standard RNN architectures and modified LSTM and GRU cells. The authors generated from a cell-based search space more than 14,000 architectures and trained it on The Penn Treebank dataset. The best-performing 289 networks were further trainted on the larger WikiText-2 dataset. This benchmark provides the number of parameters, training time, and validation perplexities at several epochs for the used datasets.

NAS-Bench-ASR (Mehrotra et al., 2021): NAS-Bench-ASR is a NAS benchmark designed for the Automatic Speech Recognition (ASR) task. It consists of 8,242 network architectures defined on cell-based search space. The benchmark provides validation accuracy per epoch and the final test accuracy on the TIMIT audio dataset. It also gives the number of parameters, FLOPs, and latencies of these networks on diverse platforms such as desktop (Tesla 1080Ti) and embedded GPUs (Jetson Nano) for different batch sizes. The networks performing very well on the TIMIT dataset are transferred to the larger LibriSpeech dataset.

NAS-Bench-Graph (Qin et al., 2022): NAS-Bench-Graph is a benchmark designed for evaluating Graph Neural Architecture Search (GNAS). It contains 26,206 unique Graph Neural Network (GNN) architectures. The architectures are trained on nine diverse graph datasets. This benchmark provides a variety of metrics for each architecture, including training, validation, and test accuracy for every training epoch, latency, and the number of trainable parameters. Latency information is provided for both Intel CPU and Nvidia GPU hardware platforms.

[1] Inverted Linear BottleNeck layer with Depth-Wise Separable Convolution

NAS-Bench-x11 (Yan et al., 2021): NAS-Bench-x11 is an advanced benchmark suite developed to enhance the granularity of performance tracking in NAS tasks. It was designed to overcome the limitations of previous NAS benchmarks, which typically allowed either single-fidelity or very limited multi-fidelity access to training metrics. Instead of providing performance metrics only at specific epochs or at the final epoch, NAS-Bench-x11 uses a surrogate model to predict the full training information at any random epoch during the training process. This allows for a more detailed analysis of the training progression and can lead to more effective and efficient NAS algorithms. The NAS-Bench-x11 suite includes three surrogate benchmarks—NAS-Bench-111, NAS-Bench-311, and NAS-Bench-NLP11—that build on previous benchmarks like NAS-Bench-101, NAS-Bench-301, and NAS-Bench-NLP. These new benchmarks estimate the full learning curve information, providing a more comprehensive view of the performance of different architectures throughout the training process.

2.4 Conclusion

In this chapter, we presented neural architecture search in detail. We first discussed AutoML, a concept to automate the machine learning pipeline that includes the field of NAS. After that, we introduced NAS in the combinatorial optimization taxonomy before discussing the details of its components. We illustrated each component using examples from the literature and presented the prominent NAS benchmarks.

We have seen how neural architecture search can be a key to the progress of deep learning. It offers different techniques and many elaborate methods to alleviate the manual and time-consuming process of architecture design. This trend forced a growing number of contributions in several directions, especially within the search strategy for NAS. However, most of them demand a great deal of design themselves to offer suitable performances in architecture search. Since we have seen how interconnected this field is with optimization, we want to contribute by elaborating search strategy that can be as efficient as complex state-of-the-art methods using a combinatorial optimization technique. In the next chapter, we introduce our contribution to this field using approaches based on local search.

Chapter 3

A Quest for Efficiency: Building on Local Search for NAS

> "Everything should be made as simple as possible, but not simpler."
>
> — Albert Einstein

This chapter contains the work published in the international conference on machine Learning, Optimization, and Data science (LOD), and the work presented as a poster at the Northern Lights Deep Learning Conference (NLDL) as well as the one presented at the national conference Recherche opérationnelle et aide à la décision en France (ROADEF).

- Meyssa Zouambi, Julie Jacques, Clarisse Dhaenens. LS-PON: a Prediction-based Local Search for Neural Architecture Search. The 8th Annual Conference on machine Learning, Optimization and Data science LOD2022, Sep 2022, Siena, Italy.
- Meyssa Zouambi, Clarisse Dhaenens, Julie Jacques. LS-Weights: Local Search with Weighted Neighborhood Sampling for Neural Architecture Search. NLDL2022, Jan 2022, Tromso, Norway.
- Meyssa Zouambi, Clarisse Dhaenens, Julie Jacques. Improving Local Search for Neural Architecture Search. 23ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, INSA Lyon, Feb 2022, Villeurbanne - Lyon, France.

3.1 Introduction

Neural architecture search has attracted a lot of attention in recent years. Researchers aim to develop the most efficient algorithms to automate the task of architecture design. As a consequence, many complex methods were proposed as we have seen in the previous chapter. However, the necessity of having intricate algorithms is sometimes questioned. Using neural networks as a search

strategy is especially troublesome since they need design and adjusting as well. The same applies to other complex strategies that need tuning before they can be used in the NAS process. On top of this, it has been demonstrated that methods as simple as random search are competitive baselines when it comes to NAS (Yu et al., 2019). So creating efficient methods is possible without excessive complexity. Therefore, it is important to consider the simplicity of a search method so it does not take away the benefit of reducing human intervention. With this in mind, we want to explore the use of local search (LS) in this area. Our choice is based on the many advantages of LS in the realm of NAS, namely:

- **The simplicity of the process:** The local search process is quite straightforward. It involves starting from an initial architecture and then incrementally adjusting it to improve its performance. An easier understanding of the search process can help debug and maintain a clean perspective on the inner workings of the algorithm.
- **Easy solution encoding:** Local search does not require a complex encoding of solutions in the search process. This permits an easy generalization to other tasks and search spaces.
- **Few or no-parameters:** This characteristic of LS helps with a faster and more general usage of this method. Since usually no tuning is required to adjust it, it can be utilized out of the box for new tasks.
- **Locality:** The concept of locality is the key feature of local search. It is embodied by the neighborhoods found in LS. This concept has its advantages in NAS as it permits the use of certain evaluation strategies that speedup the search. For example network morphism (Wei et al., 2016) or weight inheritance (Real et al., 2017).
- **Less computational resources:** Compared to other methods such as evolutionary algorithms or reinforcement learning, local search requires fewer computational resources (both in terms of memory and computational power), making it a more practical option for real-world usage.

Although it is a promising search strategy, very few works investigated the efficiency of local search for NAS. In (White et al., 2020), authors explore the theoretical characterization of the landscape and its effect on the performance of local search. They proved that LS is a very competitive method when the noise in the evaluation pipeline is reduced to a minimum. Within this setting, they demonstrate that hill-climbing -the most known form of LS- can outperform many popular state-of-the-art algorithms on popular NAS benchmark datasets. These results are confirmed in (Den Ottelander et al., 2021), where LS is employed in a multi-objective context. It shows that local search competes with state-of-the-art evolutionary algorithms, even up to thousands of evaluated architectures in the multi-objective setting. LS is therefore a method that is very easy to implement yet yields competitive performances against more complex algorithms.

Local search was also used in conjunction with network morphism as seen in (Elsken et al., 2017; Kwasigroch et al., 2019). Network morphism is a popular method to rapidly search efficient

convolutional neural networks (Wei et al., 2016). This technique allows the expansion of the network using function-preserving operations and prevents training the resulting architectures from scratch. LS is a natural way of exploiting this method since its resulting architecture generates "neighbors" of the current solution.

In this chapter, our goal is to enhance the use of local search for NAS. Our approach aims to make this algorithm even more efficient by investigating new exploration strategies for LS while keeping all the advantages previously mentioned. The remainder of this chapter starts with an explanation of the local search heuristic and introduces how to define its different components (solution encoding, neighborhood, and evaluation) for neural architecture search. It also explains the intuition behind making LS more efficient as a search strategy in this field. In the following sections, we introduce our two contributions LS-Weight and LS-PON both including experiments and results discussions. We finally conclude the chapter with our insights and future directions.

3.2 Local search for NAS

Local search is a popular heuristic used to approximately solve NP-hard optimization problems. It starts from an initial solution s_0, chosen at random or by using another heuristic. It generates neighbors of this solution by applying a *neighborhood* function N. This function applies small changes to the current solution to create neighboring ones that are close to it. Different ways of exploring the neighborhood lead to different variants of local search. The heuristic evaluates these neighbors using a cost function f that assesses their quality. It substitutes the current solution s with one from $N(s)$ to explore the landscape.

The most popular form of LS is called *hill-climbing*. In this form, the search updates the current solution with a better one from the neighborhood. After this update, it reiterates the process until convergence. The search stops when no neighbor is better than the current solution, so we can no longer improve it (the heuristic reaches a local optimum). Several exploration strategies can be chosen: the *first-improve* updates the current solution with the first improving neighbor found. The *best-improve* strategy updates the solution after evaluating all neighbors and picking the one that improves it the most. The *worst-improve* strategy evaluates all neighbors and chooses the one that improves the current solution the least.

In the context of NAS, local search can be used as a search strategy to find the most efficient solutions (i.e., architectures). In our work, we will be using hill-climbing, with the first-improve strategy, which allows exploring only a subset of the neighbors of each solution, as our purpose is to evaluate the least number of architectures for a faster search.

3.2.1 Defining the important components

In order to describe our approach in using local search for NAS, different components must be defined, namely; the solution encoding, the neighborhood function, and the solution evaluation.

Solution encoding

In the context of NAS, a solution s is the architecture of a neural network. To encode it, we must use a representation that can be translated to a neural network. There are several ways to encode an architecture. In our work, we use a list of categorical, discrete, and/or continuous values to represent it. This list can determine the type of operations of each node in the network, the connections between these nodes, the hyperparameters used for each operation, etc.

Figure 5.2, shows an example of a CNN encoding using the **Nas-Bench-201** benchmark (Dong and Yang, 2020). On the top, the encoding is represented, which is the list of operations applied to the data and their hyperparameters (note that other hyperparameters not mentioned in the encoding are fixed and not optimized during the search). On the bottom, the corresponding CNN with its operations, as defined by the benchmark. In this case, our encoding contains a list of operations that define a cell. This cell is then introduced in a predefined architecture backbone and is repeated *N* number of times. The list of values used for the encoding is sufficient to represent any architecture as defined by the search space of this benchmark.

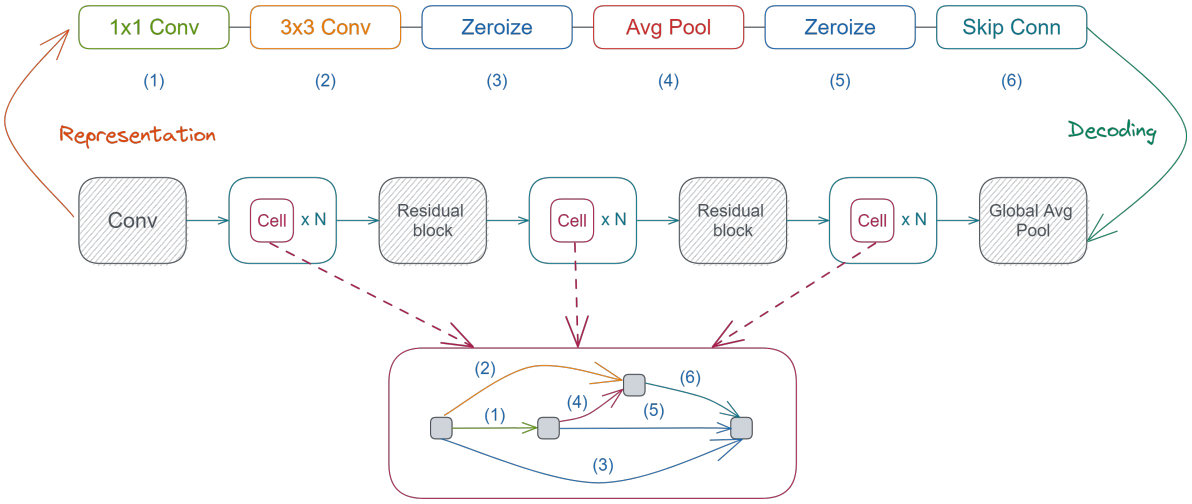


Figure 3.1: Encoding of an architecture from NAS-Bench-201. The top represents the encoding which contains the components of the cell that is present in the architecture backbone in red (best seen in colors). Each cell has the same structure.

Solution evaluation

Evaluating an architecture is the most time-consuming step in NAS. It requires training the architecture using a training set and assessing its performance using a validation set.

Depending on the task, architecture, and hardware used, this step can take several hours for a single evaluation. Therefore, in this part of our work, we will use NAS benchmarks that provide surrogate performance metrics on both the training set and validation set for all possible architectures. Since these benchmarks deal with image classification, their evaluation is based on

classification accuracy and is calculated as the sum of well-classified images divided by the total number of images of the set.

We will be using three different NAS benchmarks, NAS-Bench-201 (Dong and Yang, 2020), MacroNasBenchmark (Den Ottelander et al., 2021), and NAS-Bench-301 (Siems et al., 2020). Each benchmark defines its own set of operations and their corresponding parameters. This gives a different number of possible solutions for each of them, which defines the size of the search space. A more detailed explanation of each of these benchmarks can be found at the end of Chapter 2.

Neighborhood function

The neighborhood function N generates a neighborhood $N(s)$ that contains the architectures close to a given solution s. These architectures can defer by a single element such as an operation or an edge from the original solution. In our work, we use the *one exchange neighborhood* as defined in (Hutter et al., 2011).

In the studied NAS benchmarks, variables are all categorical. Hence, a neighbor is obtained by modifying a single variable. The neighborhood of a solution is the set of solutions obtained by selecting one by one each variable and enumerating all the possible values. The number of neighbors for each solution in NAS-Bench-201, MacroNasBenchmark, and NAS-Bench-301 is respectively 24, 28, and 136 neighbors. The size of the neighborhood is relatively small, but it is important to consider that the evaluation of a solution is extremely costly. So with the size of the neighborhood, it can quickly become impractical during the NAS process.

Figure 5.3 gives an example of a neighbor generation for a solution from NAS-Bench-201. Generating one neighbor consists of randomly choosing one operation and changing it with another, for example, here the Zero operation is replaced with a skip connection.

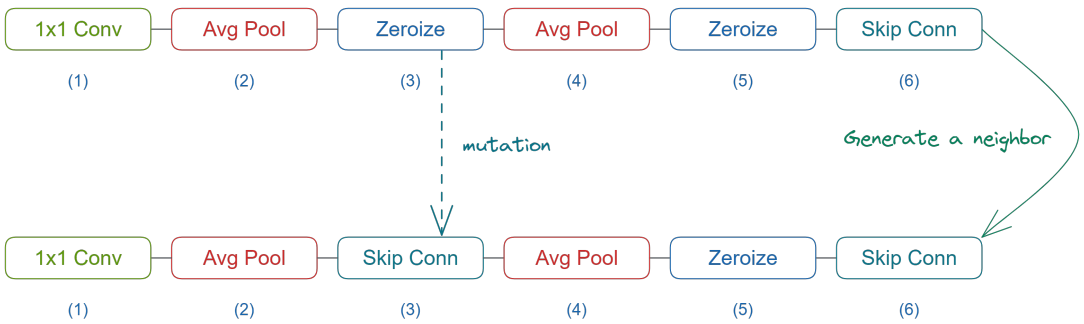


Figure 3.2: Neighbor generation with *One exchange neighborhood* function.

3.2.2 How to make LS more efficient for NAS

Evaluating architectures in NAS is the most time-consuming step of the search process. To be more time-efficient, it is important to visit the least number of solutions possible. As previously mentioned, our work is based on first-improve hill-climbing, which moves to a new solution as

soon as it finds a better one, decreasing the number of evaluated solutions is a synonym in this case to evaluating fewer neighbors. To achieve that, mindfully exploring the neighborhood of a solution can lead to a speed-up of this process. By evaluating the most promising neighbors first, we are most likely to find improving solutions quickly and move on with the search. In the next section, we explore this idea with a first approach that we named *LS-Weights*.

3.3 LS-Weights: Local Search and the Weight of the Parameters

This section initiates our exploration into consciously navigating the neighborhood of a solution. In order to avoid sampling too many neighbors that perform badly during the search, it is important to notice that certain operations/edges are more relevant than others inside an architecture and should appear more often. So focusing on evaluating architectures with more of the most important operations can lead to a speed-up in our hill-climbing algorithm. As we have seen in the previous section, the neighborhood function selects a parameter and then replaces its value with another *randomly* to generate a neighbor. If we give a higher probability for a more important parameter value to be chosen for replacement (i.e. certain operations, edges, etc.), it is more likely that we end up with a better solution.

As an example, let's consider a CNN with just four variables, each of which can take one of three possible operations: Convolution, Pooling, or Identity. We rank these operations by importance and calculate their probabilities for being selected as replacements during the mutation process. Figure 3.3 highlights how the generation of a neighbor is altered when adding the probability factor.

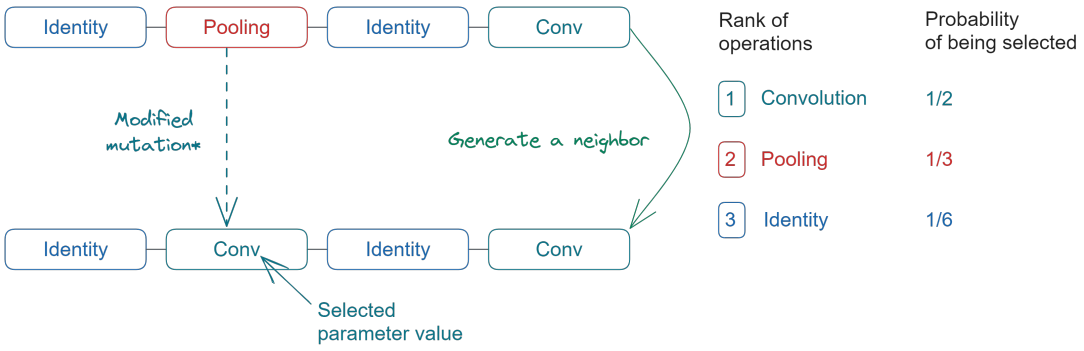


Figure 3.3: Neighbor generation with LS-Weights. * The mutation selects the new values based on the given probabilities.

Architecture	Convolution	Pooling
LeNet-5	2	2
AlexNet	5	3
VGG (VGG16)	16	5
GoogLeNet (Inception v1)	22	5
ResNet (ResNet-50)	49	1
DenseNet (D-121)	121	4
MobileNet V1	13 (Depthwise & Pointwise)	0 (strided convolutions)
EfficientNet (B0)	16	1

Table 3.1: Number of convolution and pooling operations in popular CNN architectures.

3.3.1 Determining the importance of architectural elements

Choosing the importance of an architectural element is an important step in this approach. It can be done by understanding the role of each of them, or by observing state-of-the-art architecture in the literature. We can notice in these architectures, which parts are most prevalent. To do so, we can take several famous architectures for our targeted application and analyze their components.

Since we will be using benchmarks for image classification, the type of architectures defined in our search space are convolutional neural networks (CNNs). In CNNs, the basic operations are convolutional and pooling operations (refer to Chapter 2.2 for more details on CNNs). Table 3.1 gives an example of the number of occurrences of such layers in famous CNN architectures.

From this table, it is clear that, while there is no consistent ratio between the numbers of convolutional and pooling operations, convolutional operations tend to be more numerous. This insight informs our strategy for exploring the solution space, suggesting that convolutions should be more frequent in the architecture. Other operations can also be present in CNNs, but since these are usually very specific to the contributions they appear in, it is difficult to make assumptions about their importance in a general sense. Such operations are for example skip connections. These operations allow to transport of the information as it is without processing. They are usually used to address problems such as the vanishing gradient.

3.3.2 Experiments

To assess our intuition behind the idea of LS-Weights, we conduct experiments using the NAS-Bench-201 benchmark. Our choice of this benchmark is based on its small set of possible operations (around 15 thousand) which makes it easier to rank and experiment with. In the NAS-Bench-201 architectures, each parameter allows for five possible operations: zero operation (none), skip connection, 1×1 convolution, 3×3 convolution, and average pooling. We decide to rank these possibilities based on their importance starting with the highest rank for the two convolutions, followed by the average pooling, then the skip connection, and finally the zero operation.

After assigning the rank, we set the probabilities of selecting them during the creation of a new neighbor accordingly. With the highest rank having the highest probability of selection. The

formula to calculate the probability of the operation i is $\frac{n-\text{Rank}_i+1}{\sum_{j=1}^{n}\text{Rank}_j+1}$ with rank n being the lowest rank (ie. the highest number in the order). The following table summarizes the ranks and the corresponding probability for the given example.

Table 3.2: Parameter values, ranks, and probability of selection

Parameter Value	Rank	Probability of Selection
Conv3x3	1	0.3243
Conv1x1	1	0.3243
Avg Pool	2	0.1622
Zeroize	4	0.0811
Skip Conn	3	0.1081

For the experimentation protocol, NAS-Bench-201 provides us with the validation and test accuracy of each architecture in the search space for three datasets. We use the validation accuracy provided by the benchmark as the search signal. We run the LS and the LS-Weights algorithms 150 times to ensure statistical relevance and compare them. Each run allows 1000 evaluations (the algorithms restart if they reach convergence). We report the test accuracy found at the end of the search as well as the number of evaluations needed to reach it. This number is usually a fair proxy for the speed of the NAS process as the total time of evaluation usually represents the overall time of the search process. The results and comparison are reported in Table 3.3 and Table 3.4, as well as Figure 3.4.

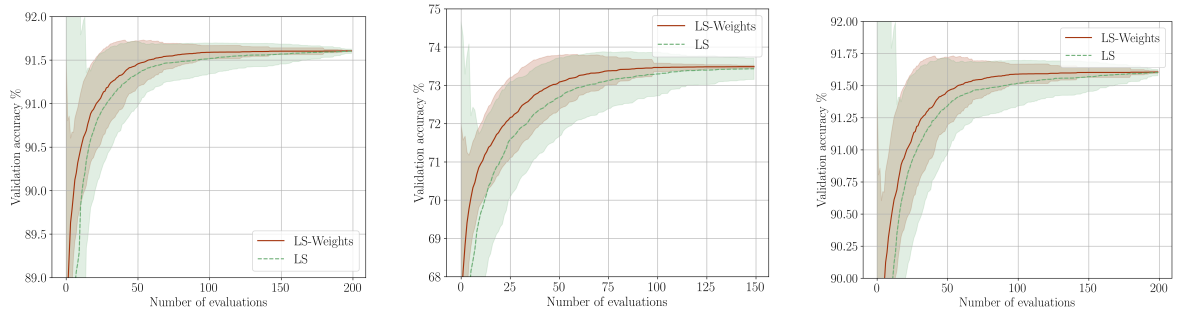
Table 3.3: The table indicates the mean and std of the test accuracy found at the end of the search. The *Optimal* value is given by the benchmark and represents the highest accuracy possible using this search space.

Method	**Cifar10**	**Cifar100**	**ImageNet**
LS	**91.61±0.00**	**73.49±0.00**	46.72±0.05
LS-Weights	**91.61±0.00**	**73.49±0.00**	**46.73±0.02**
Optimal	**91.61**	**73.49**	**46.73**

Table 3.3 shows that both LS and LS-Weights find the optimal solution inside the search space provided by the benchmark. This proves that LS is indeed state-of-the-art in this benchmark as stated in the literature. It also shows that by adding the LS-Weights mechanism we did not degrade the quality of the best solution found during the search.

Table 3.4: The table indicates the mean and std of the number of evaluated architectures before convergence. The *Speedup* represents how fast LS-Weights is compared to LS.

Method	Cifar10	Cifar100	ImageNet
LS	82±56.27	71±48.35	588±408.58
LS-Weights	**55±33.80**	**51±27.73**	**548±353.18**
Speedup	**49%**	**39%**	**7%**



(a) Cifar10 **(b)** Cifar100 **(c)** ImageNet16-120

Figure 3.4: The evolution of the validation accuracy across the number of sampled architectures.

Table 3.4 shows the gains achieved using conscious exploration of the neighborhood. We notice that LS-Weights takes a fewer number of evaluations across the three datasets. It achieves a speedup ranging from 7 to 49 percent. This acceleration can be significant in a non-benchmark setting where each evaluation can take several hours or more.

From this experiment, we conclude that both LS and LS-Weights converge toward the optimal solution, but LS-Weights do so more quickly. This indicates that the modified exploration strategy can improve search time without sacrificing solution quality.

3.3.3 Discussion

The LS-Weights method defines a very simple process to accelerate the search in LS and shows us how the choice of operations has an impact on the speed of convergence. However, there are a few problems that arise from it: First, it still needs a certain level of knowledge and expertise to understand the role of each parameter and its values. This reduces the automation of this pipeline

and does not make it accessible to non-practitioners. Second, it needs to assign new probability values each time the search space changes, requiring new analysis and adjustments every time. More importantly, it does not take into account the other values already chosen in the architecture. It treats each element as a standalone part of it and ignores the importance of certain combinations of parameters. This method also becomes impractical as soon as we have many parameter values to rank and can even lead to suboptimal solutions as it might get trapped in local optima quickly.

To solve all these issues, we thought of another solution where we let our algorithm decide *during* the search which operations and combinations of operations are more likely to be a promising solution. We rely on machine learning to find the appropriate rank of each architecture *as a whole* rather than the values of its operators. The next section explains how this method works and the results we obtained with it.

3.4 LS-PON: Local Search and the Predicted Order of Neighbors

To solve the issues of our previous method LS-Weights, we design a solution that uses machine learning to take care of selecting the architecture's importance. This method will give the rank of *the entire architecture* inside the neighborhood instead of just ranking standalone parameter values. This solution permits taking into account the interaction between the values and alleviates any human intervention or prior expertise for such ranking.

The objective is to still keep all of the benefits that LS offers while being more time-efficient and easily usable. In this case, the speed-up is achieved by ordering the exploration of neighbors of a solution using performance predictors.

Performance predictors help to identify good architectures using their characteristics only. They vary from simple decision trees to deep neural networks (Luo et al., 2020; Liu et al., 2018a; Baker et al., 2017). Recent works, however, opt for using complex models to accurately represent the huge number of possible architectures of the search space (Wei et al., 2020; Li et al., 2020). These methods usually require a pre-sampling step to gather enough architecture-performance pairs for building the prediction model. To evaluate these NAS approaches, it is necessary to consider the training time required to sample and train architectures for the predictor, as well as the design and tuning of its hyperparameters. The latter task can be considered counterproductive in this context, especially if the prediction models used are neural networks.

Wu et al. (2021) offer an interesting perspective on performance predictors. They state that using multiple simple prediction models at different stages of the search can be more efficient than using a single complex predictor to accurately predict the performance of the whole search space. They emphasize that the goal of NAS is to sample the best architectures, and most of the solutions in the search space will not be evaluated at all. So it is not necessary for a predictor to accurately estimate the performance of all of them. Their work iteratively creates weak predictors to determine which architecture is the best in the current subset of the search space. This aspect of locality is important for prediction. As this work demonstrates, solutions close to each other in a

search space are more likely to fit well using a simple predictor.

This observation motivates our work to use multiple simple machine-learning predictors throughout the search to determine the neighborhood order. Unlike most prediction-based NAS methods, we design a method that does not require any pre-sampling or parameter-tuning.

The contributions of this part of our work are summarized as follows:

- We create a simple and parameter-free method for NAS based on a local search and machine learning. This method is easy to implement and requires neither pre-sampling nor parameter-tuning.
- We use three different NAS benchmarks (with small-scale and large-scale search spaces) to validate the performance of our method. We confirm the competitiveness of LS and show that our method gives similar results while being significantly faster in almost all cases.

3.4.1 How it works

The LS-PON is composed of multiple components. Most of these components are in common with the standard LS algorithm. The solution encoding, neighborhood function, and solution evaluation all remain unchanged and are described in section 3.2.1. The last element, i.e., the performance prediction, is the key feature of this method. We will describe it in this section along with the full process in detail.

Performance prediction

In a normal setting, the local search engine evaluates neighbors in a random order. In our proposed approach, we aim to evaluate the most promising ones first by ordering them. To determine the best order in which to evaluate neighborhoods, we use linear regression models. The models do not need to be sophisticated or accurate to yield good performances, but just good enough to provide an approximate ranking of the best neighbors quickly. We choose to work with linear regressions, as they are simple, easy to implement, and do not require parameter tuning. This choice is also based on the work presented in (Wu et al., 2021), which states that using simple predictors is sufficient to estimate the performances of architectures that are close to each other.

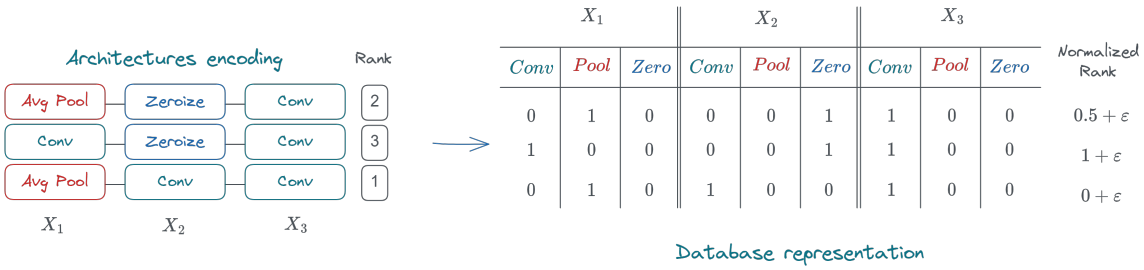
As a reminder, linear regression is a method that assumes a linear relationship between a set of variables $X = (x_1, \ldots, x_p)$ and an output variable y as follows:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

Coefficients $\beta = (\beta_0, \ldots, \beta_p)$ are learned by minimizing the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

The linear models are trained on architecture-performance pairs. The y variable is the performance we want to predict using the x_i variables which are the list of values describing an architecture. Note that in this work, the predicted score corresponds to the ranking of a solution, more

specifically, the normalized value of its rank, i.e., $\frac{\text{rank}-1}{\text{n}-1} + \varepsilon$, with n being the number of architectures in the database. Since the encoding contains categorical data, we use one-hot-encoding to create a binary column for each category and use it as a numerical value for the linear regression. Figure 3.5 presents an example of 3 simplified architectures and their corresponding representation used in the linear regression.



X_1			X_2			X_3		
Conv	*Pool*	*Zero*	*Conv*	*Pool*	*Zero*	*Conv*	*Pool*	*Zero*
0	1	0	0	0	1	1	0	0
1	0	0	0	0	1	1	0	0
0	1	0	1	0	0	1	0	0

Figure 3.5: Representation of a simplified architecture and the corresponding values used for the linear regression model. Smaller rank scores correspond to better architectures

This method does not require any pre-sampling to build the dataset for architecture-performance pairs. The first models created can be random, which is equivalent to using a random order for the neighbors. Each architecture evaluated during the search is added to a *history* database. They will be used in future iterations to create more accurate prediction models. Figure 3.6 illustrates how neighbors are ranked to select the most promising network architecture for the next evaluation.

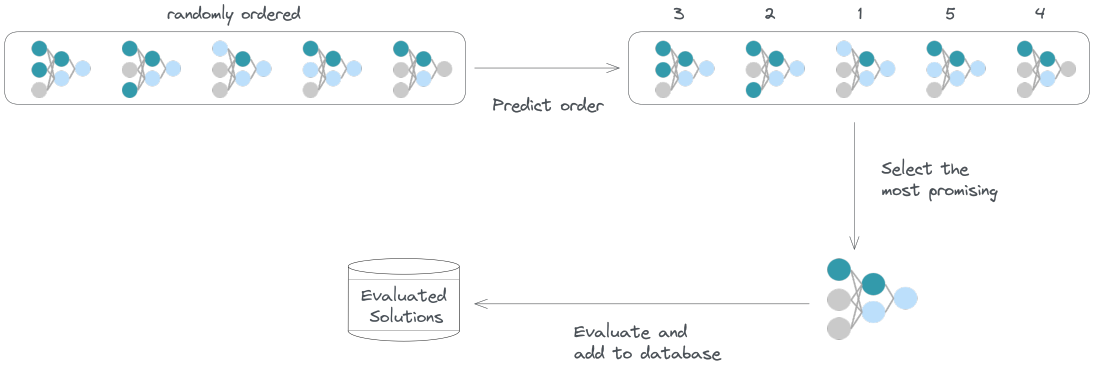


Figure 3.6: LS-PON Ordereing.

The full process

The process of LS-PON is illustrated in Figure 3.7 and works as follows: after choosing an initial solution, the method generates the neighborhood of this architecture in step one (1). In step two (2), the method creates a linear regression model to predict the performance of architectures based on their parameters. It predicts the ranking of the neighbors and evaluates them in that order in the third step (3). Each evaluated architecture gets added to the database of architecture-performance

pairs (step 4). This database will later be used to create a new (more accurate) linear model in the next iteration. If the solution is not better than the current one, it moves to the next one in the neighborhood (step 5a), else, it updates the current solution and reiterates the process (step 5b). If the search can no longer improve the current solution and the max budget of evaluations is not reached, it samples a new random solution and restarts the search. Algorithm 1 gives a detailed description of the proposed method.

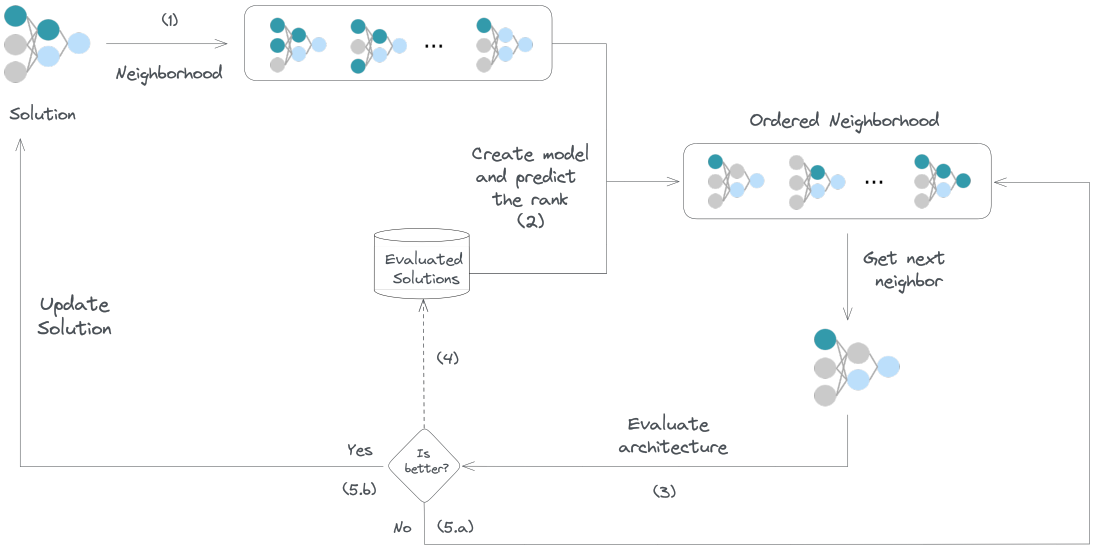


Figure 3.7: Main steps of LS-PON.

3.5 Experiments

To test the performance of the proposed algorithm, three different benchmarks are used: NAS-Bench-201 (Dong and Yang, 2020), NAS-Bench-301 (Siems et al., 2020), and MacroNASBenchmark (Den Ottelander et al., 2021). Their search spaces contain around 15k, 200k, and 10^18 solutions respectively. The choice of the first two benchmarks is to test the algorithm in a cell-based search space, with a small-scale and a large-scale setting. The last benchmark is for assessing the performance in a macro search space.

For NAS-Bench-201 and MacroNasBenchmark, we use validation accuracy as the search metric. Hence, the reference will be the best validation accuracy (Optimal) provided by the benchmark. We compare the algorithm to a standard local search. We also add random search as it is a strong baseline in NAS and can help us assess the speed of our method in a more general framework. For NAS-Bench-201, we further compare the results to Regularized Evolution Algorithm (REA) (Real et al., 2019), which is the best-achieving algorithm reported on the NAS-Bench-201 paper; results on REA are taken as-is from this paper. We set the maximum number of evaluations for these two benchmarks to 1500 evaluated architectures.

Algorithm 1 Local search with a predicted order of neighbors - LS-PON

Input: $\mathcal{A}$: search space
N: neighborhood function
$maxBudget$: maximum number of evaluations
Initialization
Randomly pick an architecture $a \in \mathcal{A}$
$acc\_a, best\_acc \leftarrow Evaluate(a)$
$D \leftarrow \cup\{(a, acc\_a)\}$
$nbEval \leftarrow 1$
while $nbEval \leq maxBudget$ **do**
  Create prediction model M and train it using D
  Predict rank of each $u \in N(a)$ using M
  Order $N(a)$ based on the predicted ranks
  for $u \in$ Ordered $N(a)$ **do**
    $acc \leftarrow Evaluate(u)$
    $D \leftarrow D \cup \{(u, acc)\}$
    $nbEval \leftarrow nbEval + 1$
    if $acc \geq best\_acc$ **then**
      Update current solution: $a \leftarrow u$
      Update best score: $best\_acc \leftarrow acc$
      Exit for loop
    end if
    if $nbEval > maxBudget$ **then**
      return best architecture
    end if
  end for
  if best solution not updated, randomly sample a new one and continue searching
end while
Output: Best architecture found

For NAS-Bench-301, we report the validation accuracy given by the surrogate model provided in the benchmark. In the literature, experiments conducted on this search space suggest that the best architectures perform around 95% of validation accuracy (Siems et al., 2020). As previously, we compare our algorithm against a local search and a random search. Since it has a larger search space than the previous two benchmarks, we set the maximum number of evaluations to 3000 architectures.

For the prediction models, we use the linear regression model from the Scikit-learn library v0.23 ($sklearn.linear\_model.LinearRegression$ with default parameters).

For each benchmark, all experiments are averaged over 150 runs. LS and LS-PON start with the same initial solution in every run. Note that the algorithms will restart after converging as long as they have not exhausted the maximum number of evaluations budget. We compare the algorithms from different standpoints: the mean accuracy, the convergence speed, and the dynamic of each

method.

3.5.1 Results

Table 3.5: Performance evaluation on the three benchmarks. Each algorithm uses the validation accuracy as a search signal. *Optimal* indicates the highest validation accuracy provided by the benchmark. We report the mean and std deviation of 150 runs for Random search, LS, LS-PON. † taken from Dong and Yang (2020).

	NAS-Bench-201			MacroNasBench		NAS-Bench-301
Method	**Cifar10**	**Cifar100**	**ImageNet**	**Cifar10**	**Cifar100**	**Cifar10**
REA†	91.19±0.31	71.81±1.12	45.15±0.89	–	–	–
RS	91.48±0.09	72.93±0.38	46.39±0.23	92.22±0.06	70.22±0.08	94.52±0.08
LS	**91.61±0.00**	**73.49±0.00**	46.72±0.03	92.46±0.04	70.45±0.02	**95.11±0.05**
LS-PON	**91.61±0.00**	**73.49±0.00**	**46.73±0.00**	**92.48±0.02**	**70.47±0.02**	**95.11±0.06**
Optimal	**91.61**	**73.49**	**46.73**	**92.49**	**70.48**	**≈95**

Table 3.5 reports the validation accuracy for random search (RS), local search (LS), and LS-PON on the three benchmarks. It also reports the REA Real et al. (2019) method for NAS-Bench-201 for comparison. We notice that LS and LS-PON give state-of-the-art results in all benchmarks. Finding either optimal or close to optimal accuracy in all cases. With LS-PON slightly surpassing LS in some cases. Both methods significantly surpass random search. For REA, despite being the best-reported algorithm on the NAS-Bench-201 paper, random search still outperforms it in all available datasets. This could suggest that the hyperparameters of REA are too specific to the NAS problem it was designed for and did not generalize well. This also confirms that random search is a strong baseline in NAS (Li and Talwalkar, 2020).

Since both algorithms, LS and LS-PON, give similar performances on data quality, to further compare them, we need to analyze their speed. Indeed, in a non-benchmark setting, each sampled architecture needs to be trained. Training an architecture takes significantly more time than running a local search. For this reason, the number of sampled architectures is a strong indicator of the speed of each method.

Table 3.6 reports the mean and standard deviation of the number of evaluated architectures during the search. Both LS and LS-PON restart after convergence as long as the evaluation budget is not reached. Since random search never surpasses these two methods after exhausting all of its evaluation budget (1500 for the first two benchmarks and 3000 for the last one) it is omitted from

Table 3.6: Speed evaluation on the three benchmarks. The table indicates the mean and std deviation of the number of evaluated architectures before convergence. Values in bold mean statistically better (Wilcoxon's test).

	NAS-Bench-201			MacroNasBench		NAS-Bench-301
Method	**Cifar10**	**Cifar100**	**ImageNet**	**Cifar10**	**Cifar100**	**Cifar10**
LS	82±56	71±48	588±408	622±414	628±393	**1792±833**
LS-PON	**59±45**	**49±24**	**281±168**	**539±403**	**483±417**	1880±835
Speedup	**38%**	**44%**	**109%**	**15%**	**30%**	**-4%**

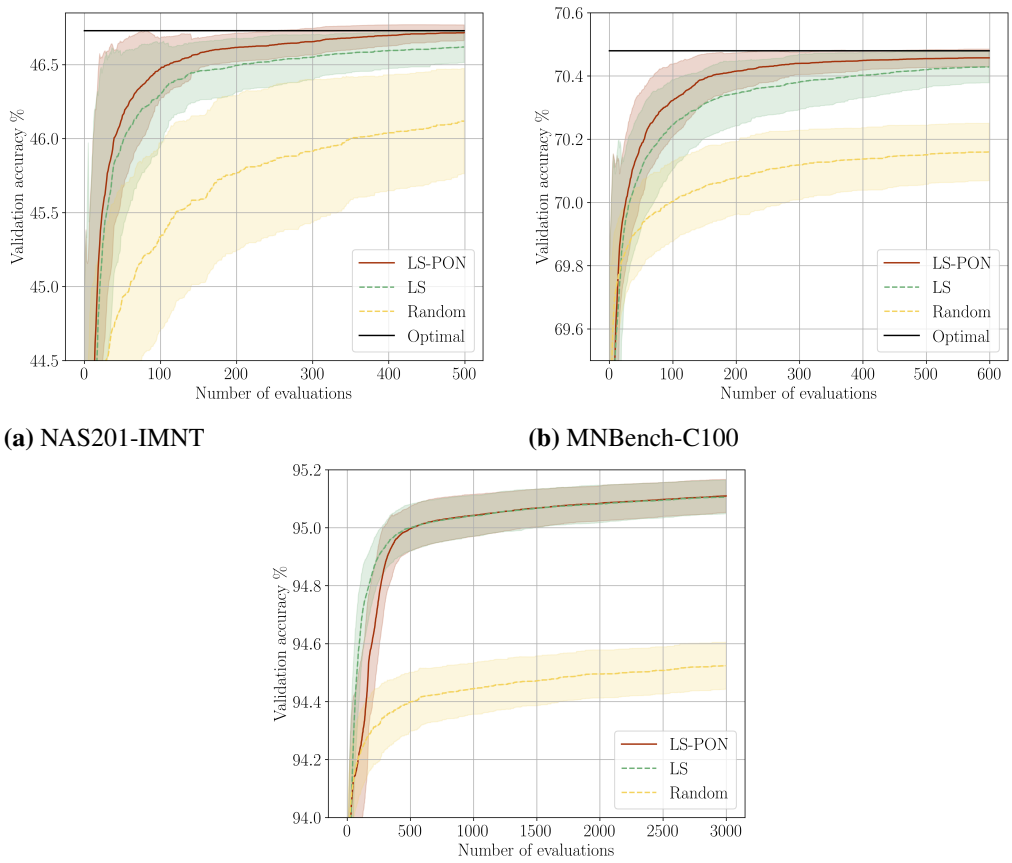
this table. The acceleration obtained using LS-PON is reported in the last line of the table. It shows that in both NAS-Bench-201 and MacroNasBenchmark, there is a significant speedup ranging from 15% to up to 109%. In NAS-Bench-301, however, LS-PON shows less efficiency with a slower convergence time. Figure 3.8 gives examples that illustrate the evolution of the validation accuracy across the number of evaluated architectures for a different dataset in each benchmark. It shows how LS-PON is faster than LS in NAS-Bench-201 and MacroNasBenchmark for ImageNet16-120 (IMNT) and Cifar100 respectively, and how it is slightly less efficient in NAS-Bench-301 for its Cifar10 dataset.

Table 3.7: Results on all benchmarks. The table indicates the mean and std deviation of the number of required evaluations before improving on the current solution during the search. Values in bold mean statistically better (Wilcoxon's test)

.

	NAS-Bench-201			MacroNasBench		NAS-Bench-301
Method	**Cifar10**	**Cifar100**	**ImageNet**	**Cifar10**	**Cifar100**	**Cifar10**
LS	4.66±4.38	4.69±4.43	4.55±4.34	5.45±5.70	5.49±5.70	**18.16±26.08**
LS-PON	**2.22±3.06**	**2.05±2.86**	**1.94±2.62**	**2.50±3.12**	**2.48±2.90**	18.66±26.78

Table 3.7 further shows the dynamic of these methods. It reports the number of evaluated neighbors before improving on the current solution. In NAS-Bench-201 and MacroNasBenchmark, it requires from 2 times to almost 3 times the number of evaluations for LS to find a better neighbor compared to LS-PON. For this reason, LS-PON progresses more quickly during the search.



(a) NAS201-IMNT

(b) MNBench-C100

(c) NAS301-C10

Figure 3.8: Example of the evolution of the validation accuracy across the number of evaluated architectures for a dataset of each benchmark. NAS201-IMNT, MNBench-C100, and NAS301-C10 correspond to NAS-Bench-201 for ImageNet16-120, MacroNasBenchmark for Cifar100, and NAS-Bench-301 for Cifar10. Results are averaged for 150 runs for all algorithms. The shaded region indicates the standard deviation of each search method.

In NAS-Bench-301, however, it takes around 2% more neighbor evaluation for LS-PON to find a better neighbor compared to LS, which makes LS-PON slightly slower in this case.

To investigate the reason behind this, and understand the effectiveness of this method in the different benchmarks/datasets, an analysis of the distance between the current solution's accuracy and its neighbors' accuracy is conducted. This analysis seeks to determine if the difference between the neighbors' accuracies affects the method's effectiveness. To do this, the mean absolute error between the current solution accuracy, and its neighbors' accuracy is calculated for a sample of solutions. Results of this are represented in Figure 3.9. We see that for the three datasets in NAS-Bench-201, there is a notable difference between the neighbors' performances. Hence, the predictor can more easily classify them and identify the best ones. This difference is less visible on MacroNasBenchmark but the benchmark still has a certain number of outliers (represented by

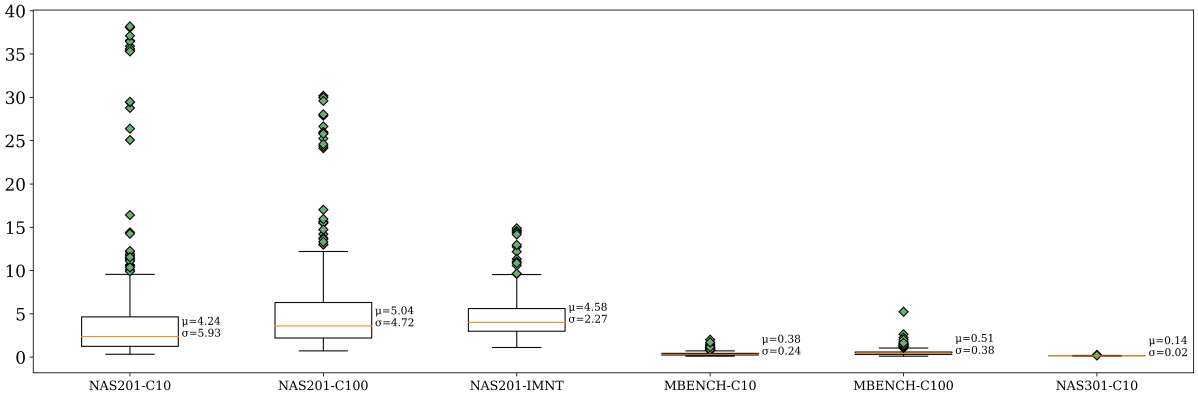


Figure 3.9: Box plots represent the distribution of distances between the validation accuracy of a solution and its neighbors, and diamonds represent outliers. NAS201, MNBench, NAS301 stand for NAS-Bench-201, MacroNasBenchmark and NAS-Bench-301. C10, C100 and IMNT are respectively Cifar10, Cifar100 and ImageNet16-120.

diamonds in the Figure) that can be easily recognized by the predictor.

On the other hand, in NAS-Bench-301's dataset, the difference between the accuracy of neighbors is negligible (there is a mean of 0.14% difference in their accuracy) and there are also not many noticeable outliers to recognize during the search. The search space of this benchmark is mostly composed of good architectures with very close performances as presented in their paper Siems et al. (2020).

This shows that the method is more efficient if the improvement to make is relatively observable. This is expected in problems where the search space contains a diverse set of solutions.

LS-PON proved that it can be more than twice as fast as the LS without the ordering mechanism. Its effectiveness relies on the diversity of solutions in the search space and works better if there is an observable difference in terms of accuracy between neighbors.

3.6 Conclusion

In this chapter, we introduced two approaches to improve the time efficiency of local search. The first LS-Weights, is based on ranking the importance of operations inside a network to effectively guide the neighbor generation. It reduces the number of required evaluations without compromising on the quality of the final solution. However, this method still requires human expertise and intervention. To improve on this, we have designed a second approach, called LS-PON, that is based on performance predictors for neighborhood order. This method, on top of being also more time efficient than the vanilla local search (from 30% to 109% faster in some datasets), is easy to implement, does not require any hyper-parameter tuning, and yields state-of-the-art results on three popular NAS benchmarks.

In the future, we will test other types of predictors and analyze their impact on the results. We also aim to make this method more robust to search spaces that mostly contain solutions with very

close performances.

Now that we have implemented an efficient method in the single objective context, an interesting next step is to find a similar parameter-free method that can compete with other works in the multi-objective context, and see if it scales well with the growing objectives of architecture design, such as the size of the network, the energy consumption, the inference times, etc. To do that, we will study in the next chapter the problem of multi-objective NAS and its challenges.

Chapter 4

Multi-objectively, NAS

> "The art of life lies in a constant readjustment to our surroundings."
>
> — Kakuzo Okakura

4.1 Introduction

Balancing several competing factors is a crucial aspect of neural network design. While achieving high performances on a given task is important, other aspects are also essential in real-world applications. Depending on the chosen task and targeted applications, several characteristics need to be considered. For example, considering model complexity in the design process can lead to neural networks that are less prone to overfitting. Similarly, considering the memory footprint or energy consumption is critical for resource-constrained environments. Likewise, being mindful of the inference time can be important for applications that have time-constrained predictions. These examples illustrate the multifaceted nature of neural network architecture design. It highlights the importance of a multi-objective (MO) approach that balances competing factors according to the specific use case and constraints. For this reason, researchers have been exploring alternative optimization approaches that incorporate more elements into network design. Multi-objective optimization in NAS addresses this challenge by balancing trade-offs between predictive performance and other metrics, leading to more practical models. By focusing on multiple objectives, NAS can discover diverse architectures that cater to different needs, allowing practitioners to select the most suitable architecture for their problem.

In this chapter, we explore how multi-objective optimization is used in neural architecture search. To do that, we begin by briefly explaining multi-objective optimization and its key features. Next, we present a classification of the approaches used in MO NAS with the most notable works found in the literature. Lastly, we conclude with the challenges we identified that are left to address in this area.

4.2 Multi-objective optimization in a nutshell

Multi-objective optimization is a branch of optimization that deals with problems involving several, often conflicting, objectives. In these problems, the goal is to find a set of solutions that achieve an optimal balance between the various objectives, rather than optimizing a single objective function. The primary challenge in multi-objective optimization is navigating the trade-offs between the objectives, as improving one objective might lead to the degradation of others.

4.2.1 Problem formulation

The problem formulation of multi-objective optimization involves defining the objectives, decision variables, and the feasible region determined by constraints on the decision variables. Let us denote the decision variables as $\boldsymbol{x} \in \mathbb{R}^n$ and the k objective functions as $f_i(\boldsymbol{x})$, where $i = 1, 2, \ldots, k$. The multi-objective optimization (minimization) problem can be formulated as:

$$\text{minimize} \quad \boldsymbol{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x})) \quad \text{subject to} \quad \boldsymbol{x} \in \mathcal{X}, \tag{4.1}$$

Where $\boldsymbol{F}(\boldsymbol{x})$ is a vector of the k objective functions, and $\mathcal{X}$ is the feasible region determined by constraints on the decision variables.

4.2.2 Pareto dominance and Pareto optimality

In Pareto dominance, a solution $\boldsymbol{x}_1$ is said to dominate another solution $\boldsymbol{x}_2$ if $\boldsymbol{x}_1$ is at least as good as $\boldsymbol{x}_2$ in all objectives, i.e. (in a minimization context), $f_i(\boldsymbol{x}_1) \leq f_i(\boldsymbol{x}_2) \quad \forall i = 1, 2, \ldots, k.$ and $\boldsymbol{x}_1$ is strictly better than $\boldsymbol{x}_2$ in at least one objective, i.e., $f_i(\boldsymbol{x}_1) < f_i(\boldsymbol{x}_2)$ for at least one i (Deb, 2011).

Pareto optimality is a state where it is impossible to improve one objective function without worsening at least one of the other objective functions. A solution is said to be Pareto optimal if no other feasible solution dominates it.

The Pareto front is the set of all Pareto-optimal solutions in the solution space. These solutions represent the trade-offs between the competing objectives, and decision-makers can choose a solution from the Pareto front based on their preferences or other criteria. Figure 5.6 gives a representation of a Pareto-front for a bi-objective problem with f_1 and f_2 the objectives to minimize.

4.2.3 Techniques in multi-objective optimization

Multi-objective optimization algorithms aim to discover a diverse set of solutions that lie on or close to the Pareto front, representing the optimal trade-offs among the conflicting objectives. In this section, we introduce a set of exact approaches, the scalarization approach, as well as a selection of classical metaheuristic methods, each recognized for its usefulness in addressing multi-objective optimization problems.

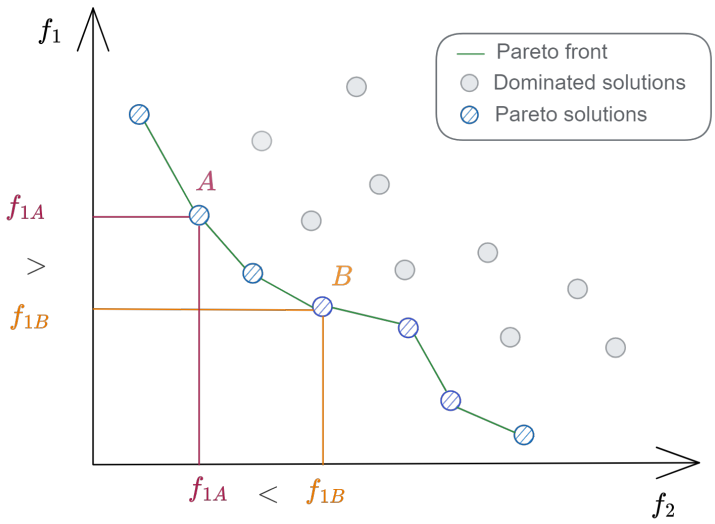


Figure 4.1: Example of a Pareto-front for a bi-objective problem.

The Scalarization approach (Marler and Arora, 2004):

Scalarization allows the transformation of multi-objective problems into single-objective problems by combining the objectives into a scalar function (generally using weighted sum or product). This enables the use of traditional single-objective optimization algorithms offering a straightforward way to handle multiple objectives. In this scenario, exploring the Pareto-front requires adjusting weights to discover a new solution for each setting.

Exact method:

Exact methods in multi-objective optimization serve as a solution for finding solutions that lie on the Pareto front with a guarantee of optimality. Unlike heuristic and metaheuristic methods, which seek approximations of the optimal solution, exact methods provide precise and accurate results, ensuring that all the solutions are truly non-dominated. Prominent examples of exact methods include:

- **Epsilon-constraint (Haimes, 1971):** The epsilon-constraint method is used to find Pareto solutions by optimizing one objective while setting specified limits, or "epsilon constraints," on the other objectives. The process involves systematically varying the constraints on one objective, while optimizing another, to generate a set of solutions that form a part of the Pareto front.
- **The two-phase method (Ulungu and Teghem, 1995):** The two-phase method is particularly employed in bi-objective optimization. In the first phase, it identifies the two extreme points of the Pareto front by optimizing each objective separately. In the second phase, it explores between these extremes, utilizing scalarization techniques to find additional sup-

ported Pareto optimal solutions. A generalization to this method accounts for more than two objectives and was proposed by Tenfelde-Podehl (2003)

- **Parallel Partitioning Method (PPM) Lemesre et al. (2007):** Another exact method designed for bi-objective optimization. The method has three stages to determine the entire Pareto front. Initially, it computes the Ideal and Nadir points, using extreme solutions in the bi-objective case, to constrain the search space. Next, it seeks well-distributed efficient solutions to segment the search space. Finally, it finds other efficient solutions by further narrowing the search space using solutions from the second stage. PPM was later generalized to K-PPM to account for more than two objectives in (Dhaenens et al., 2010)

Metaheuristic:

Metaheuristics offer an alternative approach to solving multi-objective optimization problems. They are particularly useful when dealing with large-scale or computationally intensive scenarios. These methods, while not guaranteeing optimality, provide a practical and efficient means to approximate the Pareto front, balancing solution quality and computational effort in complex optimization landscapes. The following is a selection of well-known algorithms in this category:

- **The Non-dominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al., 2002):** NSGA-II is an evolutionary multi-objective optimization algorithm that iteratively generates and refines a population of candidate solutions. It ranks solutions based on non-domination and crowding distance, using these measures for selection. Genetic operators are applied to create offspring, which are combined with parents and sorted to form the next generation. The result is a diverse set of non-dominated solutions (an approximation of the Pareto optimal solutions), each representing a unique trade-off among objectives.

- **The Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) (Zhang and Li, 2007):** In MOEA/D, the multi-objective optimization problem is decomposed into a number of scalar optimization subproblems that are solved simultaneously. Each subproblem focuses on optimizing a specific region of the Pareto front. Solutions are evolved by considering their own subproblem and those of neighboring subproblems, encouraging diversity. Like NSGA-II, this algorithm also provides an approximation of the Pareto optimal solutions representing trade-offs among the objectives.

- **The Strength Pareto Evolutionary Algorithm 2 (SPEA2) (Zitzler et al., 2001):** A multi-objective optimization algorithm that works through an iterative process of evolving a population of candidate solutions. It assigns fitness to solutions based on their degree of domination and density estimates. It employs binary tournament selection, crossover, and mutation operators to generate offspring. Solutions from the combined offspring and parent populations are selected to update an external archive of non-dominated solutions, ensuring the

maintenance of quality and diversity. Similar to the previous approaches, the algorithm outputs an approximation of the set of non-dominated solutions upon meeting a termination criterion.

- **Dominance-based Multi-objective Local Search (DMLS) (Liefooghe et al., 2012):** This approach presents a comprehensive model that contains multi-objective strategies that iteratively improve a set of non-dominated solutions through local explorations. Its main elements include a neighborhood function, a selection mechanism, an archive, and an exploration strategy. The most notable example of DMLS is the Pareto Local Search (Paquete et al., 2004). As its name suggests, this method also returns an approximation of the set of non-dominated solutions at the end of the search.

4.2.4 Performance metrics

To evaluate the quality of the obtained solutions and compare different multi-objective optimization algorithms, various performance metrics can be used. Widely used performance metrics are:

- **Spacing:** Spacing is a performance metric used in multi-objective optimization to evaluate the diversity of an approximation set of solutions. It provides a measure of how uniformly the solutions are distributed along the Pareto front. A well-distributed set of solutions is desirable, as it allows decision-makers to choose from a diverse set of alternatives that span the trade-offs between objectives.

- **Generational Distance (GD) and Inverted Generational Distance (IGD):** These two metrics are distance-based performance indicators. GD quantifies the average distance from each non-dominated solution in the approximation set to the closest point on the reference Pareto front. On the other hand, IGD calculates the average distance from each point on the reference Pareto front to the nearest non-dominated solution in the approximation set. In both cases, smaller values signify better convergence towards the reference front. IGD, in particular, accounts for both convergence and diversity in the approximation set, as it considers the distances from the entire reference Pareto front to the obtained non-dominated solutions.

- **Hypervolume Indicator:** Also known as the S-metric, it measures the volume in the objective space that is dominated by the approximation set and is bounded by a reference point, which is typically chosen to be worse than the worst values for each objective. A larger hypervolume indicates a better approximation of the Pareto front. The hypervolume captures two aspects of multi-objective optimization: how close the approximation set is to the true Pareto front, and how evenly the solutions are distributed along the Pareto front. By considering the volume dominated by the approximation set, the hypervolume indicator provides a single scalar value that accounts for both of these aspects.

4.3 The state of the literature in multi-objective NAS

Traditional NAS approaches generally focus on optimizing a single objective, such as minimizing the error rate or maximizing the accuracy. However, real-world applications often require balancing multiple objectives. For example, it may be desirable to minimize the resources consumed by the neural network or to have a neural network with reduced inference time. Multi-objective NAS is an advanced approach in the field of neural architecture search that aims to optimize multiple objectives simultaneously. The most common objectives considered when designing a neural network are presented in the next section.

4.3.1 Common objectives in neural network design

- **Accuracy:** The primary objective is often to maximize the accuracy of the model on a given task, such as classification or regression. This can be measured using metrics like top-1 accuracy, top-5 accuracy, F1-score, etc.
- **Computational complexity:** Another common objective is to minimize the computational complexity of the architecture. This can include reducing the number of parameters, FLOPs (floating-point operations per second), or total model size (memory footprint).
- **Latency:** Minimizing the inference time, or latency, is crucial for many real-time applications. This can involve optimizing the architecture to reduce the number of layers, convolutions, or other operations that impact the model's latency.
- **Energy efficiency:** For edge devices and mobile applications, energy efficiency is an important consideration. This objective aims to minimize the energy consumption of the neural network during training and inference, often by reducing the number of operations or simplifying the architecture.
- **Robustness:** Ensuring that a model is robust to adversarial attacks, noise, or other perturbations is another objective in multi-objective NAS. This can involve optimizing the architecture to improve the model's generalization capability and resilience to various perturbations.
- **Transferability:** Some NAS methods aim to optimize architectures that can be easily transferred to other tasks or domains. This can involve searching for architectures with good performance across a range of tasks or datasets, or optimizing for architectures that can be fine-tuned with minimal additional training.
- **Fairness:** In applications where fairness is a concern, NAS can be used to optimize architectures that mitigate biases and ensure equitable performance across different demographic groups. This may involve minimizing the disparity in performance metrics across different subgroups of the data.

4.3.2 Approaches for multi-objective NAS

Many approaches have been used in the literature to tackle the problem of multi-objective NAS. They can be found using different search strategies to fetch the best architectures. In this section, we will give an overview of the literature for the most notable works found in each approach.

Evolutionary algorithms

The broad field of multi-objective NAS has seen a variety of evolutionary algorithm-based approaches. Each of these approaches offers unique strategies and techniques to optimize multiple objectives simultaneously. For instance, LEMONADE (Elsken et al., 2018), a multi-objective NAS method, optimizes for predictive performance, inference time, and the number of parameters. It maintains a population of trained networks that form an approximate Pareto front. The method generates child networks through mutation operators, guided by principles of Lamarckian inheritance. It utilizes Network Morphisms (NM) and Approximate Network Morphisms (ANM) for mutation (Kwasigroch et al., 2019). An alternative evolutionary algorithm-based approach is implemented by NSGA-Net (Lu et al., 2019). It leverages the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) for maintaining a diverse set of solutions across objectives. This technique concurrently optimizes for classification error and computational complexity, quantified by the number of floating-point operations during a forward pass. NSGA-Net applies a three-stage process: it initializes a population based on hand-crafted architectures, explores through genetic operations (crossover and mutation), and exploits the knowledge from previously evaluated architectures using a Bayesian Network. Further, Lu et al. (2020) present another example of these types of algorithms. This method considers both classification performance and computational cost in the architecture design process. To progressively modify architectural components, it uses genetic operations for recombination and mutation. To improve computational efficiency, it employs two strategies: down-scaling of architectures during the search phase and reinforcement of patterns common among successful past architectures through Bayesian model learning.

Evolutionary algorithm-based approaches exhibit a shared principle: evolving populations of network architectures to achieve optimal trade-offs between objectives. These methods make use of mutation operators and genetic operations, introducing variability in architecture design. This variability, coupled with methodologies for retaining desirable traits (e.g., Lamarckian inheritance, Bayesian model learning), results in an efficient exploration-exploitation balance across the architecture search space. However, it is important to consider the potentially high computational cost associated with these methods due to the necessity of training a multitude of child networks.

Swarm optimization

In the context of multi-objective NAS, swarm optimization-based approaches have also been explored, with different methods presenting unique strategies to optimize various objectives concurrently. One such example is MOCS-Net (Zhang et al., 2020), this method applies a multi-objective cuckoo search algorithm to NAS, with a specific focus on mobile device network architecture. It

operates within a compact and flexible search space using block-wise efficient mobile CNNs. It leverages Lévy flights as mutations to enhance the search for optimal architectures. MOCS-Net aims to balance classification accuracy and computational resources. The parameters it considers include inference latency, the number of parameters, and floating-point operations (FLOPs). Another method, MOCNN (Wang et al., 2019), employs Multi-Objective Particle Swarm Optimization (MOPSO) to evolve deep convolutional neural networks. It targets the trade-off between classification accuracy and computational cost, with the latter quantified by Floating Point Operations (FLOPs). This measure reflects both training and inference costs. MOCNN automates the tuning of hyperparameters of DenseNet, focusing on parameters such as the number of dense blocks, growth rate, and the number of layers in each dense block. By applying an optimized MOPSO algorithm to these encoded dense blocks, MOCNN optimizes the two objectives. The algorithm retrieves non-dominant solutions, which represent a Pareto front of optimized CNN architectures. Users can then select the most suitable architecture from these non-dominant solutions, considering their specific needs and resource availability.

With their unique mutation mechanisms, swarm optimization-based methods present an interesting approach to multi-objective NAS. However, the complexity of their operators may introduce challenges in tuning the search process, and, similar to evolutionary algorithms, they may result in high computational costs.

Reinforcement learning

Reinforcement learning approaches have also been popular in multi-objective NAS. For instance, MONAS (Hsu et al., 2018) utilizes a reinforcement learning framework guided by an RNN controller. The controller generates a sequence of tokens, with each representing a different aspect of the architecture and hyperparameters of a convolutional neural network. After the controller generates an architecture, it is trained and its performance on the validation set provides the reward signal, which is a combination of multiple reward functions. The latter may include a weighted sum of accuracy and energy cost or a hard constraint on either. The same paper also introduces MONAS-S, which uses a weight-sharing technique. This technique allows architectures sampled by the controller to share weights, reducing the computational cost of training each sampled architecture individually.

MnasNet (Tan et al., 2019) is another approach that uses a reinforcement learning framework, incorporating real-world latency into the main objective of the architecture search. It aims to optimize the trade-off between accuracy and latency. This method is composed of an RNN-based controller and a mobile phone-based inference engine. The controller generates new neural network architectures which are then trained and executed on mobile phones to measure their actual latency. A reward, defined by a customized weighted product between accuracy and latency, is computed for each model and used to update the parameters of the controller via Proximal Policy Optimization. MnasNet also employs a novel factorized hierarchical search space to balance the flexibility and size of the search space while promoting layer diversity.

Reinforcement learning is a powerful method that has also been tailored for multi-objective

NAS. It uses reward signals, often based on the scalarization of objectives, allowing the account of objective trade-offs. However, such an approach is considered sub-optimal in multi-objective optimization (Deb, 2011).

Gradient-based optimization

Gradient-based optimization is another approach that has been used in multi-objective NAS. It enables efficient and direct optimization of the architecture. This approach utilizes gradient information to guide the search, optimizing architecture parameters in a continuous space. An example is ProxylessNAS (Cai et al., 2018), a gradient-based algorithm aimed at optimizing network architectures for specific tasks and target hardware without relying on proxy tasks. The design of ProxylessNAS includes path-level binarization to manage memory consumption during the architecture search process. It also incorporates a latency regularization loss to consider hardware-specific constraints in the optimization process. ProxylessNAS has been evaluated on large-scale datasets, like CIFAR-10 and ImageNet, across different hardware platforms, including GPUs, CPUs, and mobile devices.

Another instance is FBNet (Wu et al., 2019), which employs a differentiable neural architecture search framework for the design of convolutional neural networks. By using gradient-based methods to optimize the architecture, it eliminates the need to enumerate and train each individual architecture separately. Its loss function includes the latency of the architecture on the target hardware, using a weighted product with the cross-entropy. Furthermore, an exponent coefficient is added to modulate the magnitude of the latency term, allowing for a more flexible and hardware-aware network design.

Gradient-based methods for multi-objective NAS provide significant advantages, such as computational efficiency, flexibility, and scalability, making them suitable for large-scale tasks. However, these methods have their limitations. They can fall into local optima due to the complexity of NAS search spaces, and the requirement of discretizing the continuous optimization space can lead to sub-optimal solutions. Additionally, designing suitable regularization terms or loss functions to balance multiple objectives can be challenging, adding to the complexity of the problem.

Bayesian optimization

Bayesian optimization is another approach that was used to efficiently navigate the search space and optimize objectives. We find it in SpArSe (Fedorov et al., 2019) which utilizes this strategy to optimize convolutional neural networks for microcontrollers with limited resources (often used in Internet of Things (IoT) devices). The method in SpArSe combines neural architecture search and pruning in a unified framework to balance three competing objectives: inverse validation accuracy, model size in bits, and maximum working memory across all CNN layers. The process includes sampling a set of configurations, which are either generated by a multi-objective Bayesian optimizer (MOBO) or selected randomly, to ensure broad coverage of the search space. MOBO evaluates changes to previously sampled configurations, yielding both new and reference configu-

rations. Subsequently, the new configuration inherits parameters from the reference, followed by retraining and pruning. This approach provides an efficient and flexible method to optimize CNNs, even under tight resource constraints, making it especially suited for IoT applications.

Bayesian optimization gives a different approach to multi-objective NAS and can exploit previous information during the search process. Nonetheless, this approach is challenged by the high dimensionality of the NAS search spaces.

Table 4.1: Illustrative work from the literature

Name	**Paper**	**Search Strategy**	**MO Approach**	**Objectives**
LEMONADE	(Elsken et al., 2018)	Evolutionary Algorithm	Scalarization Pareto-based	Accuracy, Inference time, Num parameters, Num Operations
NSGA-Net	(Lu et al., 2019)	Evolutionary Algorithm	Pareto-based	Classification error, floating-point operations (FLOPs)
MOCNN	(Wang et al., 2019)	Swarm Optimization	Pareto-based	Accuracy, floating-point operations (FLOPs)
MOCS-Net	(Zhang et al., 2020)	Swarm Optimization	Pareto-based	Accuracy, inference latency, number of parameters, and floating-point operations (FLOPs)
ProxylessNAS	(Cai et al., 2018)	Differentiable NAS	Scalarization	Accuracy, Latency on different hardware
MONAS	(Hsu et al., 2018)	Reinforcement Learning	Scalarization	Accuracy, Energy
MnasNet	(Tan et al., 2019)	Reinforcement Learning	Scalarization (weighted product)	Accuracy, Latency
FBnet	(Wu et al., 2019)	Differentiable Strategy	Scalarization (weighted product)	Cross-entroypy, Latency
SpArSe	(Wu et al., 2019)	Bayesian Optimization	Scalarization	Validation Accuracy, Model size in bits, maximum working memory

4.4 Conclusion

The landscape of MO NAS methods is diverse, it employs various strategies such as evolutionary algorithms, reinforcement learning, or gradient-based optimization to balance critical factors in neural network design. Each method offers a distinctive trade-off between performance, computational efficiency, and resource utilization, and presents a set of strengths and potential challenges. For example, evolutionary and swarm optimization-based approaches excel at exploring a wide range of architectures and optimizing multiple objectives concurrently. However, they often require substantial computational resources and time due to the necessity of training numerous networks.

Reinforcement learning and gradient-based optimization methods introduce efficient ways of exploring the search space and directly optimizing network architectures, reducing the computational load compared to their evolutionary and swarm optimization counterparts. However, these methods can face challenges with the scalarization of objectives and maintaining a broad exploration of the search space.

Bayesian optimization also provides a unique approach to the problem. However, the high-dimensional nature of the search space in NAS may pose a challenge, as it traditionally works best in lower-dimensional spaces.

We summarize this with a set of illustrative works in Table 4.1. This table presents the search strategy used, its multi-objective approach to optimization, and the objectives.

One particular problem that is shared between these different approaches is the complex design required for these approaches to transfer to the NAS problem. This challenge is particularly counter-productive if our goal is to minimize the manual labor required in designing neural networks.

In the next chapter, we will present a different approach that tries to solve the above issues with a dominance-based multi-objective local search. This chapter will include our reasoning behind this choice as well as a detailed explanation of how to adapt this method to neural architecture search.

Chapter 5

An Alternative Pareto-based Approach for Multi-objective NAS

"The whole is greater than the sum of its parts."

— Aristotle

This chapter contains the work presented as a workshop in the International Joint Conference on Artificial Intelligence (IJCAI) and published as a conference paper at the Congress on Evolutionary Computation (CEC).

- Meyssa Zouambi, Julie Jacques, Clarisse Dhaenens. Dominance-Based Local Search for Neural Architecture Search, DSO Workshop, IJCAI 2022, Vienna, Austria.
- Meyssa Zouambi, Clarisse Dhaenens, Julie Jacques. An Alternative Pareto-based Approach to Multi-objective Neural Architecture Search. IEEE 2023 Congress on Evolutionary Computation, Jul 2023, Chicago, United States.

5.1 Introduction

The challenge of balancing competing objectives in neural architecture search continues to require innovative solutions. As we have seen in the previous chapter, approaches to multi-objective optimization in NAS have primarily centered around two main approaches. The first is the use of population-based methods such as evolutionary algorithms or swarm optimization. These methods, although effective, require substantial computational resources and often complex mechanisms. The second is to introduce a scalarization approach to the search process that permits the account of multiple objectives simultaneously. This approach simplifies the transition to a multi-objective context in NAS using traditional search strategies. However, in multi-objective optimization, the scalarization of objectives is considered suboptimal. The need for less complex algorithms that consider a Pareto-based approach is still present.

In this chapter, we will look at a new way of handling these competing objectives that will address the previous challenges. Our focus will be on the use of Dominance-based Multi-objective Local Search (DMLS) in NAS. This approach is an extension of the well-established Local Search (LS) and presents itself as a promising solution to these challenges. It capitalizes on the strengths of LS while incorporating the flexibility to accommodate multiple objectives. It's easy to implement, efficient (Liefooghe et al., 2012), and similar to LS, capable of leveraging advanced techniques like weight inheritance and network morphism which reduce the training time for architecture evaluation. It has also been effective in other machine-learning contexts as seen in (Jacques et al., 2015) which makes it a great candidate for our study.

In this chapter, we will go into detail about how DMLS works and explain its benefits for NAS. We will propose a formulation of the NAS problem for a resolution with DMLS and back our discussion with practical examples and evidence from experiments. We will show that DMLS can effectively navigate the complex world of multi-objective neural architecture search.

5.2 Introducing dominance-based multi-objective local search

As seen in Chapter 2, local search methods are known to provide good-quality solutions for many hard combinatorial optimization problems. Dominance-based multi-objective local search (DMLS) is an adaptation of the single-objective local search for multi-objective optimization problems (Liefooghe et al., 2012). It defines a unified model that includes all multi-objective approaches that iteratively improve a set of non-dominated solutions based on local explorations. Such methods include Pareto Archived Evolution Strategy (PAES) (Knowles and Corne, 1999) or Pareto Local Search (Paquete et al., 2004) (PLS). The main components that form a DMLS approach are:

- **Problem Representation:** This includes the objectives to be optimized, the constraints, and the decision variables that form a solution.
- **Initialization Strategy:** The algorithm can start off with a single solution or a set of solutions, generated randomly or through a problem-specific heuristic.
- **Neighborhood Function:** This defines which solutions are considered "neighbors" of a given solution. The definition of a neighbor depends on the problem representation. For instance, in a binary decision variable problem, a neighbor could be obtained by flipping one bit.
- **Dominance relation:** Different dominance relations can be used to create a DMLS method (Pareto, weak, strict, etc.). The most common one is Pareto dominance.
- **Evaluation Process:** This involves the objective functions and the dominance comparison. The algorithm evaluates the objective functions and checks if the solution dominates or not the solutions from the archive.

- **Selection Mechanism:** This is the process of choosing the next solution in the search. In PLS for example, if a neighbor dominates the current solution, it becomes the new current solution.
- **Exploration Strategy:** Can be similar to the exploration strategies found in single-objective local search. It defines how the neighborhood of a solution is explored and how the next solution can be selected.
- **Archive Management Strategy:** This involves the methods used to maintain the archive (data structure used to store the non-dominated solutions found during the search). For instance, when the archive becomes too large, some solutions might need to be removed. This could be done based on crowding distance, hypervolume contribution, clustering, or other criteria.
- **Stopping Criterion:** This is the condition that determines when the algorithm stops. It could be based on a maximum number of iterations, a time limit, or a condition on the quality or diversity of the solutions in the archive.

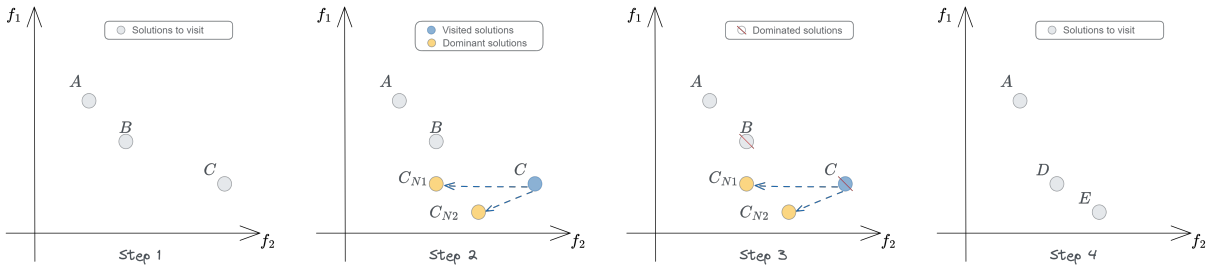


Figure 5.1: DMLS iteration

Algorithm 2 describes the full process of a DMLS method. It starts with an archive of one or several initial solutions that are non-dominated by one another. The algorithm then proceeds to explore this archive by iteratively evaluating the neighbors of unvisited solutions and adding the non-dominating ones. The archive gets pruned automatically of dominated solutions each time. The process ends once the stopping criterion is met. Figure 5.1 describes a standard iteration of a DMLS method with f_1 and f_2 representing the two objectives to minimize. In step 1, the iteration starts by selecting one or more unvisited solutions from the archive (in this case solution C). In step 2, the algorithm explores the neighborhood of the selected solution(s). This typically involves generating a set of solutions using the neighborhood function. In this case, two neighboring solutions, C_{N1} and C_{N2}, are generated from C. In step 3, if a neighboring solution dominates a solution in the archive, the dominated solution is removed from the archive, and the dominating solution is added. Finally, the algorithm starts a new iteration, selecting one or more unvisited solutions from the updated archive for exploration. If there are no more unvisited solutions in the archive, the algorithm stops.

Algorithm 2 Dominance-based Multi-objective Local Search

```
DMLS
solution ← initial solution
archive ← initial set (possibly empty)
while not stopping criterion and unvisited solutions available do
  neighbors ← generate neighbors(solution)
  for neighbor ∈ neighbors do
    if neighbor is not dominated by any solution in archive then
      archive ← archive ∪ neighbor
      remove dominated solutions from archive
      if archive exceeds its maximum capacity then
        prune archive with archive management strategy
      end if
      Mark solution as visited
      solution ← select new solution from archive
      Break
    end if
  end for
  Mark solution as visited
  solution ← select new solution from archive
end while
```

Output: *archive*

5.3 DMLS for NAS

Considering the importance of multi-objective neural network design, we explore the use of local search for multi-objective NAS. Our goal is to design a search algorithm that requires the least amount of human intervention that comes through tuning and parameter selection. With the LS advantages that we highlighted in the previous chapters, we want to explore how this plays out in this context. While a previous work has explored the use of LS in this multi-objective NAS (Den Ottelander et al., 2021), their method is based on the scalarization of objectives. However, as this can give suboptimal solutions, we want our approach to use Pareto dominance and generate a set of non-dominated solutions.

Since evaluating the quality of a neural architecture is very costly (can even take hours for a single evaluation in certain tasks), we base our work on a pre-defined benchmark to simplify experimentation and assess the quality of our approach. We focus on using the **MacroNASBenchmark** (Den Ottelander et al., 2021), a benchmark designed specifically for evaluating multi-objective NAS methods. It includes a search space of over 200,000 unique architectures, along with their predictive performance on image classification for two popular datasets: CIFAR10 and CIFAR100. In addition, each architecture is assigned a metric for complexity. Unlike the other benchmarks, this one also provides the optimal Pareto front for the two objectives (complexity and accuracy) which will help assess the performance of our multi-objective method.

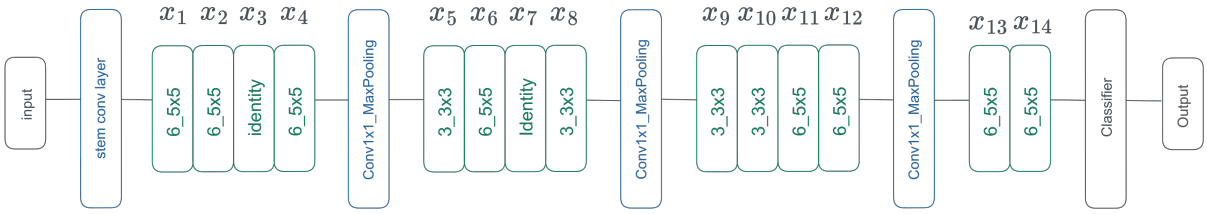


Figure 5.2: Solution encoding

.

In the following, we explain the different components of the DMLS for NAS and give a detailed description of our proposed approach. The different components are adapted to MacroNasBenchmark, but our approach can nonetheless be adapted for other search spaces, datasets, or metrics by following the same encoding and procedures.

5.3.1 Solution encoding

The encoding used in our method is a list of values. This list contains the different parameters chosen that uniquely define an architecture in the search space. For the **MacroNASBenchmark** (Den Ottelander et al., 2021), as we have presented it in Chapter 2, each solution consists of 14 unrepeated cells (x_1 to x_{14}). Each cell x_i can take one of three options (*Identity*; *3_3x3*: MBConv[1] with expansion rate 3 and kernel size 3; *6_5x5*: MBConv with expansion rate 6 and kernel size 5). Figure 5.2 shows an example of a solution from this search space. The vector x_1 to x_{14} contains the values of each selected option. The components in blue are immutable and only the options in green can change from one solution to another. They represent the encoding of the solution. In total, there are more than 200,000 unique solutions available (after removing duplicates).

5.3.2 DMLS components

Neighborhood function

Similar to what we have presented in the previous chapters, in our work, we define the neighborhood of a solution as the set of solutions that differs by exactly one cell from the current one. The set of neighbors is obtained by selecting each cell and enumerating all the possible values. For the used benchmark, there are 28 possible neighbors for each solution (2 other alternative values for the 14 cells). Figure 5.3 shows an example of a neighbor generation of a solution in this benchmark. For example, here a neighbor is obtained by modifying the operation in cell x_2 (6_5x5) by the *Identity* operation.

[1]Inverted Linear BottleNeck layer with Depth-Wise Separable Convolution

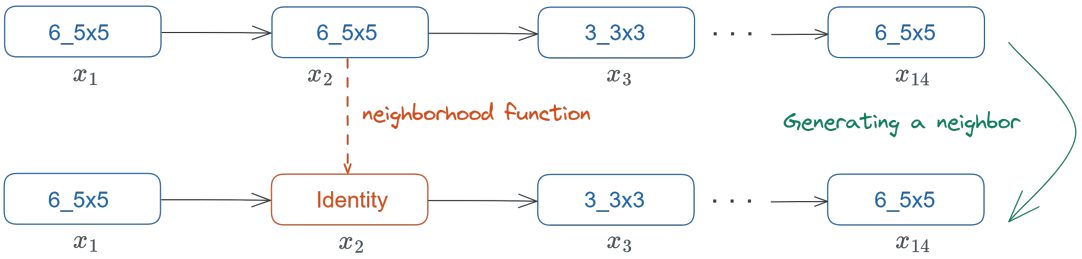


Figure 5.3: Neighbor generation with the neighborhood function

Solution evaluation

To evaluate an architecture, we chose two objectives, one representing the predictive performance of the model (to maximize) and one for the model's complexity (to minimize). Optimizing the model's complexity along with the predictive performance is usually useful when we want a lightweight model or a model that is easier to interpret, it also makes it less prone to overfitting. MacroNASBenchmark provides for each architecture precomputed accuracies (sum of well-classified images divided by the total number of images) for two image classification datasets (CIFAR10 and CIFAR100). It also provides a metric for complexity (the Million Multiply Accumulate Operations MMACs) that is used as the second objective. To compare between solutions, we use the Pareto-based dominance as defined in the previous chapter. In this case, the objectives to minimize are $f_1 = 1 - accuracy$ and $f_2 = MMACs$. We store the non-dominated solutions in an archive that will represent the approximate Pareto-front throughout the search.

Solution initialization

We initialize the archive with a trivial solution (containing all *Identity* operations) which is the optimal solution for the MMACs metric. We sample different solutions uniformly at random from the search space. When meeting a solution that is not dominated by other solutions from the archive, we add it until we reach the desired initial size. Any solution that becomes dominated is automatically removed.

The initial archive size is the only parameter that needs to be tuned for this approach, the impact of its choice will be studied in the experiment section. We propose to set the maximum size of the archive is set to infinity (unbounded).

Selection strategy

During the search, solutions from the archive get selected for exploration. Different approaches for selecting these solutions are available. Examples include selecting multiple solutions to explore simultaneously or using ranking methods to select which ones to start with. In our approach, we explore *one* solution at a time. This solution is chosen randomly from the archive. This way we reduce the number of evaluations required during the search process and favor diversification by using randomness during selection.

Exploration strategy

For the exploration strategy, we opt for the *first improve* approach. Once a neighboring solution is found to dominate the current one, the search immediately uses that solution without exploring the rest of the neighbors. This strategy offers a balance between intensive and extensive search. While it might not guarantee that the best neighbor is always selected, it considerably reduces the computational overhead by cutting down the number of evaluations.

5.4 Experimentation protocol

To assess the performance of the proposed algorithm, we run it for the CIFAR10 and CIFAR100 image classification datasets provided by the MacroNASBenchmark (Den Ottelander et al., 2021). We chose this benchmark because it is specifically designed for evaluating bi-objective NAS, as stated in their paper. CIFAR10 and CIFAR100 are also the two most popular datasets in NAS benchmarks for this task (Ying et al., 2019; Dong and Yang, 2020; Siems et al., 2020).

We compare our work with two other methods previously tested in (Den Ottelander et al., 2021). Both of these use different approaches: a population-based method with NSGA-II and a scalarization-based approach using a customized local search[2]. The two methods are described as state-of-the-art for this benchmark (Den Ottelander et al., 2021).

We use accuracy and MMACs as the two objectives for the architecture search. Both of them are provided by the benchmark. We limit the number of evaluations to 6000 evaluated architectures (approx 3% of the size of the search space). We fix this number because in a non-benchmark setting evaluating architectures is very costly. If an algorithm stops before the evaluation budget is reached, it restarts the search. The best solutions are kept for each restart.

The benchmark provides the globally optimal solutions (Pareto optimal). It is the set of solutions that are non-dominated by the whole search space. They are obtained by exhaustive search and will be a point of comparison to the approximate Pareto-front obtained by other methods.

The setup of each method is as follows:

- For the NSGA-II algorithm, it uses a 2-point crossover, a single-variable mutation with probability $p_m = 1/l$, tournament size of 2, and population size n_{pop} = 100. As suggested in (Den Ottelander et al., 2021) for this problem.

- For the local search algorithm, a random scalarization of the objectives is used in this single-objective search algorithm, meaning that random weights are assigned for each objective. This compensates for having to choose weights for each objective a-priori. An archive is maintained to keep the non-dominated solutions from each iteration.

- For our method (DMLS), the only parameter to choose is the initial size of the archive. We run our tests for multiple archive sizes ranging from 2 to 10 and compare the results.

[2]Two more algorithms are presented for this benchmark. Unfortunately, we were not able to run them for comparison using the provided code.

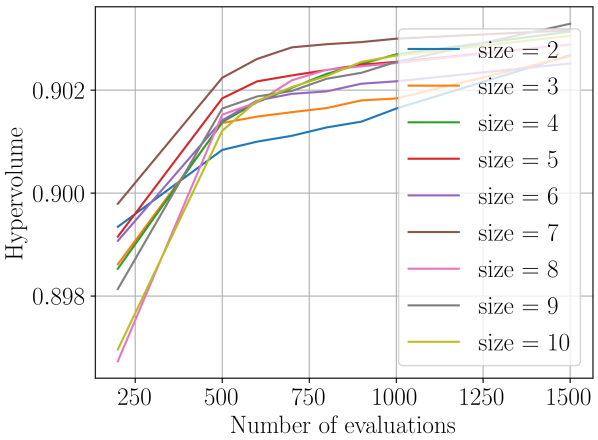


Figure 5.4: Hypervolume using different initial sizes for the archive

Figure 5.4 shows that all archive sizes seem to reach the same point after a certain number of iterations. However, bigger sizes of the archive tend to give the method a slower start. We choose to work with an archive size of 7 as it gives a larger hypervolume in the early stages of the search.

- Following the protocol in (Den Ottelander et al., 2021), we also add the trivial solution to the archive (or population) to each method before starting the search. This solution contains all identity cells and represents the lower bound for the second objective.

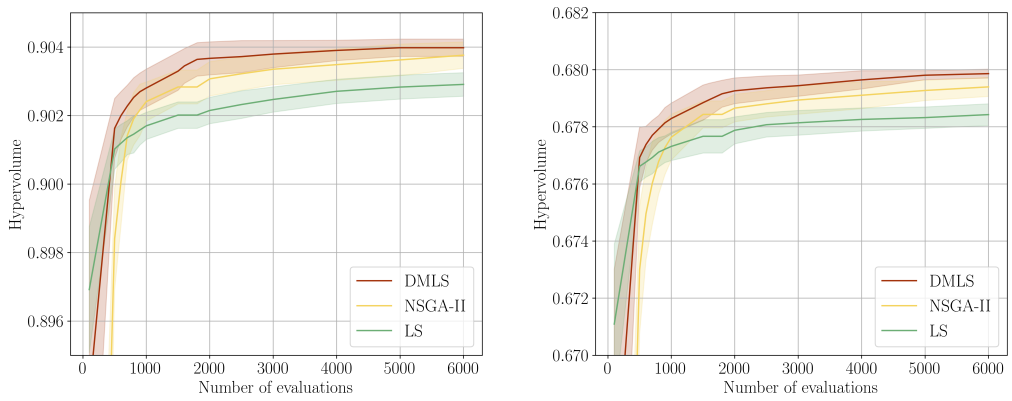
For each method, all experiments are averaged over 30 runs. We compare the algorithm from several standpoints: the hypervolume that should be maximized (we normalized the objectives, turned them into a minimization problem, and calculated the hypervolume with (1,1) as a reference point) and the percentage of exact Pareto optimal solutions found.

We use the Mann-Whitney U rank test to assess the statistical significance of our results.

5.5 Results and discussion

5.5.1 Hypervolume results

Table 5.1 reports the hypervolume of each method for CIFAR10 and CIFAR100 at different evaluation steps (500, 1000, 1500, 3000, and 6000). We notice the three approaches showing very close values at each step, with DMLS seemingly taking the lead. In the early stage of the search, at 500 evaluations, the NSGA-II seems to give a slightly lower hypervolume than LS before quickly catching in the 1000 evaluations mark. Figures 5.5.a and 5.5.b show the quick evolution of the hypervolume across the number of evaluations for CIFAR10 and CIFAR100, respectively. Here we see that in the first evaluations, LS briefly gives the highest hypervolume. This can be explained by the time it takes to initialize the archive of DMLS and the population for NSGA-II. But since NSGA-II requires a larger population, it takes it more time to surpass LS.



(a) Hypervolume (CIFAR10)

(b) Hypervolume (CIFAR100)

Figure 5.5: Evolution of the hypervolume across the number of evaluated architectures for CIFAR10 and CIFAR 100. Results are averaged for 30 runs for each algorithm.

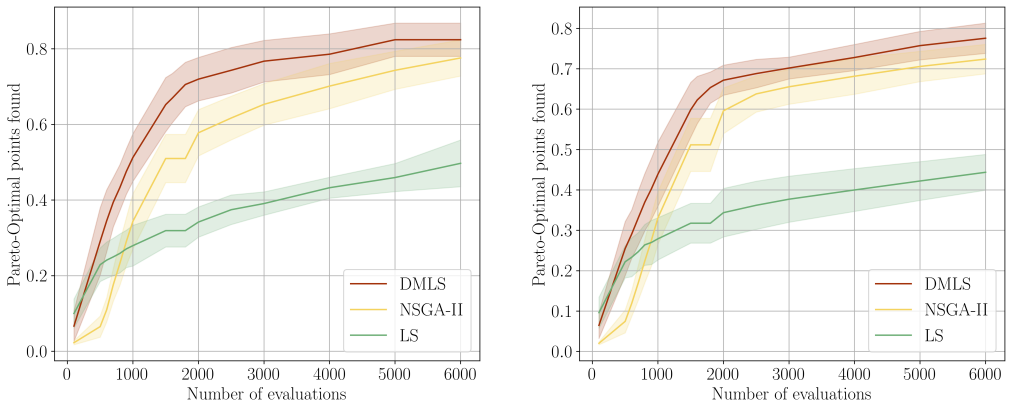
	EVALUATIONS	DMLS	NSGA-II	LS
CIFAR10	500	**90.16±0.08**	89.84±0.14	90.10±0.05
	1000	**90.28±0.05**	90.24±0.05	90.17±0.03
	3000	**90.37±0.039**	90.33±0.04	90.24±0.03
	6000	**90.39±0.02**	90.37±0.03	90.29±0.03
CIFAR100	500	**67.69±0.10**	67.30±0.22	67.66±0.05
	1000	**67.82±0.05**	67.76±0.08	67.73±0.04
	3000	**67.94±0.03**	67.89±0.04	67.81±0.04
	6000	**67.98±0.01**	67.93±0.03	67.84±0.03

Table 5.1: Table reporting the hypervolume (scaled for better readability) for each method on CIFAR10 and CIFAR100. Results at presented for 500, 1000, 3000, and 6000 evaluations. Values in bold are statistically better (Mann-Whitney U rank test).

The very close hypervolume results suggest that the solutions composing the approximate Pareto-front are spread in an equivalent manner. For this reason, another metric should be used to better compare these methods. Here we will use the number of optimal Pareto points found for each of them.

5.5.2 Analysis of Pareto points found

Table 5.2 presents the percentage of Pareto optimal solutions found for each method on CIFAR10 and CIFAR100. For this metric, there is a more significant difference between each method, and this is true for both datasets as well. When the search budget is reached, DMLS finds statistically more Pareto optimal solutions. In CIFAR10, for example, DMLS ends the search with more than



(a) % of Pareto optimal solutions found (CIFAR10)

(b) % of Pareto optimal solutions found (CIFAR100)

Figure 5.6: Evolution of the percentage of Pareto optimal solutions found across the number of evaluated architectures for CIFAR10 and CIFAR 100. Results are averaged for 30 runs for each algorithm.

	EVALUATIONS	DMLS	NSGA-II	LS
CIFAR10	500	**29.07±8.89**	6.52±2.79	22.90±4.45
	1000	**51.27±6.29**	34.39±7.31	27.94±5.37
	3000	**76.73±5.49**	65.31±5.49	39.07±3.08
	6000	**82.41±4.39**	77.58±4.78	49.71±6.17
CIFAR100	500	**25.49±6.67**	7.45±2.83	22.22±3.99
	1000	**43.92±7.93**	33.07±6.49	27.90±5.22
	3000	**70.19±2.69**	65.55±4.31	37.71±5.69
	6000	**77.58±3.74**	72.41±3.64	44.37±4.42

Table 5.2: Table reporting the percentage of Pareto optimal solutions found for each method on CIFAR10 and CIFAR100. The results at presented for 500, 1000, 3000, and 6000 evaluations. Values in bold are statistically better (Mann-Whitney U rank test)

82% of the Pareto optimal solutions, followed by NSGA-II with 77.6% of solutions, and lastly, LS with only 49.7% of optimal solutions. Similarly, for CIFAR100, DMLS takes the lead with 77.6% Pareto optimal solutions found at the end of the search, against 74.6% and 44.4% for NSGA-II and LS, respectively. It is clearly shown in Figure 5.6 as well, which represents the percentage of Pareto points found across the number of evaluations. Here we can see that DMLS has performed much better since the beginning of the search, even if it has a lower hypervolume than LS, as seen previously.

5.5.3 Discussion

To sum up, results show that even if our proposed DMLS for NAS is only slightly higher in hypervolume than other methods, it can find significantly (and statistically) more Pareto optimal solutions compared to them. Which gives more architecture choices at the end of the search for a limited evaluation budget. On top of that, DMLS is easy to implement and requires only one parameter to adjust (the size of the archive). This method has also the potential to exploit local properties to speed up network training (which is the most time-consuming task), such as weight inheritance (Real et al., 2017) or network morphism (Wei et al., 2016). This would further reduce the time needed to search for architectures in a non-benchmark setting.

5.6 Conclusion and future directions

The goal of this chapter is to propose an alternative multi-objective resolution method for NAS using dominance-based multi-objective local search (solution encoding, neighborhood, and evaluation function). Unlike other multi-objective NAS strategies, it does not require defining weighted preferences between objectives like scalarization methods, nor is it complex to tune like other Pareto-front-based methods. It requires only one parameter to tune (the initial archive size), which makes it easily reusable for new tasks without much tuning. Since it is based on local search, it has the potential to exploit the local property-based evaluation strategies for NAS that will speed up the global search time. The experiments on a specialized MO benchmark show that our approach obtains statistically better Hypervolume and finds significantly more Pareto optimal solutions than NSGA-II and LS. As a perspective, we will compare this method using datasets for other types of tasks, such as natural language processing or image segmentation. Another interesting approach would be to study the effect of using it in conjunction with the weight inheritance strategy to assess the speed up it can obtain.

Chapter 6

Bridging Barriers: from the Benchmarks to the Real World

> "You never know ahead of time what something's really going to be like"
>
> — Katherine Paterson, Bridge to Terabithia

6.1 Introduction

While benchmarks offer standardized datasets and tasks to evaluate the performance of NAS methods, using neural architecture search in real-world scenarios presents several challenges that benchmarks dismiss. In order to carry out an end-to-end NAS process, multiple elements need to be regarded and carefully crafted to get the best out of neural architecture search. The goal of this part is to step out of the benchmark zone and identify the different issues that arise from it. In this study, we take a step toward real-world applications using healthcare data. Our aim is also to validate the method proposed in Chapter 3 while navigating new properties such as search spaces and training protocols.

The chapter is organized as follows: it first starts by providing a list of considerations that need to be handled when moving to real data. Next, it presents the design choices and experimental protocol used in our study. Finally, it presents the results obtained and discusses new perspectives.

6.2 The challenges of NAS in the real world

When using neural architecture search outside the familiarity of benchmarks, many issues can arise from different sources, most commonly from data, training pipelines, and search space design. The following explains different challenges and how they can affect the NAS process:

- **Data preparation:** Real-world data is often not as cleanly distributed as the datasets contained in NAS benchmarks. The real-world data might have class imbalances, noise, and

other irregularities. This directly affects the assessment of an architecture's true performance during evaluation. This problem is more critical during NAS evaluations as it could lead the search to non-efficient regions of the search space. On top of this, data volume poses another significant challenge. While benchmark datasets have fixed sizes, real-world datasets can vary widely in volume. Small datasets might not provide enough information for a NAS method to discern between architectures, while very large datasets might be computationally prohibitive. It is important to notice the specificities of the real-world dataset used and process it accordingly. A pre-step of data cleaning and preparation is essential before moving to the NAS process. Some solutions in data processing include normalization, denoising, data augmentation, automated feature selection, using specialized loss functions, etc.

- **Network capacity:** Benchmarks usually provide a search space containing architectures that are mostly suitable for their specific datasets, in terms of their type (e.g., CNNs, RNNs, etc.) and their capacity. The network capacity refers to the ability of a neural network to fit a given function, and it is typically associated with the complexity of the network. High capacity within the search space translates to more parameters and deeper or wider architectures. Evaluating these architectures requires more computational resources, both in terms of memory and processing power. A search space containing high-capacity architectures is also more prone to overfitting. Conversely, a search space with architectures of low capacity might underfit, causing poor performances for the given dataset. Searching in a space with lower capacity networks might be faster but could miss out on potentially more performant architectures. It is possible to include both types in the search space, but this makes the NAS process computationally more intensive and time-consuming. Designing the right search space that fits the datasets is essential and the results of the NAS highly depend on it.

- **Noisy evaluation:** An architecture that seems slightly better in one evaluation might be worse in another due to random fluctuations. This noise during evaluation poses significant challenges when comparing and ranking various architectures. It can also lead to the early termination of potentially good architectures or the continuation of poor-performing ones, especially when incorporating techniques like early stopping or population-based methods. If architectures are ranked based on noisy validation performance, the NAS process might prioritize architectures that just happened to perform well due to random fluctuations but might not generalize well to new data.

  Several factors contribute to this variability. For instance, model initialization. Different initial weights can set off diverse convergence paths, leading to fluctuating validation performances, especially during the early stages of training. Additionally, regularization techniques, including dropout, data augmentation, and stochastic depth, introduce intentional randomness into the training process. On top of this, using a high learning rate or training data in a network with low capacity can cause such fluctuations.

  To meet these challenges, it is beneficial to adopt certain strategies. Conducting multiple evaluations of the same architecture and averaging the results can offer a more trustworthy

performance estimate.

- **Computational resources:** Running neural architecture search outside of benchmarks requires training the architecture for evaluation, this step can take hours for a single evaluation. On top of this, real-world datasets might be large and complex, making the search process more computationally intensive and requiring more resources and time. To overcome the noise and obtain more reliable estimates of architecture performance, we might also need to perform multiple evaluations or longer training, increasing the computational cost. This is one of the most significant problems and the main reason why benchmarks were crafted. Many methods that reduce the computational cost are available, they are commonly known in NAS as evaluation strategies. For detailed examples of this, you can refer to Chapter 2.

6.3 Diving in with medical image classification

Now that we are familiar with some of the prominent challenges in applying NAS to real use cases, we will try to use our previous contribution LS-PON for a further study. In this section, we walk through the steps needed to prepare the experiments. We start by presenting the dataset, and how we tailored different NAS components to it. We will also explain our experimental protocol before presenting the results and their discussion.

Disclaimer: The experiments of this chapter were conducted on limited resources (using a single RTX 2080 GPU). Although it lays out a good foundation on using NAS on real data, a more throughout study could be carried out to provide even more insights and investigations for this niche.

6.3.1 Datasets

For our experiments, we will use the MedMNIST (Yang et al., 2023) as our primary data source for experimentation. MedMNIST is a large-scale repository of standardized biomedical images. It consists of a total of 708,069 2D images and 9,998 3D images, distributed across 12 datasets for 2D images and 6 datasets for 3D images. The images are pre-processed and are formatted as 28 x 28 pixels for 2D images, and 28 x 28 x 28 voxels for 3D images. Each image is labeled with the corresponding classification. All the datasets have the same pre-processed format, with standard train-validation-test splits. This repository also offers the performance obtained for a set of models for each of the provided datasets. The models available are the well-known and state-of-the-art ResNet models, models automatically designed with the open-source solutions Auto-sklearn and AutoKeras, and finally, a model created from the commercial Google AutoML Vision tool.

Our experiment limits its focus to two particular datasets, PneumoniaMNIST and DermaMNIST. These datasets were chosen because they are lightweight and represent two different classification tasks that are prominent in healthcare. Both of these datasets offer specific challenges and opportunities relevant to our study, from the binary nature of diagnosing pneumonia in Pneumoni-

aMNIST to the more complex multi-class categorization in DermaMNIST. Details of each dataset are as follows:

- **PneumoniaMNIST:** This dataset consists of Chest X-ray images with a binary classification task, labeling images as either indicative or non-indicative of pneumonia. It contains a total of 5,856 samples, distributed across training (4,708), validation (524), and test (624) sets.
- **DermaMNIST:** This dataset is composed of dermatoscopy images with a multi-class classification task involving seven categories. The dataset comprises 10,015 samples, subdivided into training (7,007), validation (1,003), and test (2,005) sets.

6.3.2 Behind the scenes

Before running experiments, multiple aspects must be considered. These aspects, detailed hereafter, are often disregarded when using benchmarks as they are usually included in the framework that created the benchmark. Nonetheless, these elements are important and heavily affect the efficiency of any NAS framework. Every element in this should be tailored to the task and dataset that it will be used for. For time efficiency, we choose to tune these elements using PneumoniaMNIST, which is the smallest of the two selected datasets. We will use the same parameters selected here for DermaMNIST and see if they will transfer well.

Selecting the appropriate search space

One of the first steps when using NAS is picking the right search space. It defines the range of architectural possibilities that the search can investigate. This choice should be informed by the specific requirements of the domain and the computational feasibility of exploring the space. While the search space should be expansive enough to capture potential optimal architectures, it shouldn't be so vast that it becomes computationally prohibitive.

Designing a search space takes significant expertise and a considerable amount of time. To avoid this process, researchers usually use search spaces previously defined either in benchmarks or in prominent works such as DARTS (Liu et al., 2018b). For our work, we test two different search space options provided in NAS-Bench-201 and NAS-Bench-301. Both of these search spaces were used in NAS for general domain image datasets. Picking either one for our experiments should be validated first. For this reason, we trained several randomly sampled architectures from each search space on the PneumoniaMNIST dataset and observed the training plots for the train set, test set, and validation set. Figure 6.1 gives a representative sample of the plots obtained from architectures of the NAS-Bench-201 search space. We observed a lot of fluctuations during training which indicates a mismatch between the complexity of the architectures and the dataset. It is worth mentioning that we fixed a small learning rate (0.0001) to rule out the possibility of the fluctuations being caused by big weight updates. In contrast, and using the exact same training protocol, the plots obtained from architectures of the NAS-Bench-301 behaved differently. As we

can see in Figure 6.2, the train curve is much smoother and we can see notable progress across epochs. Going forward we will use this search space.

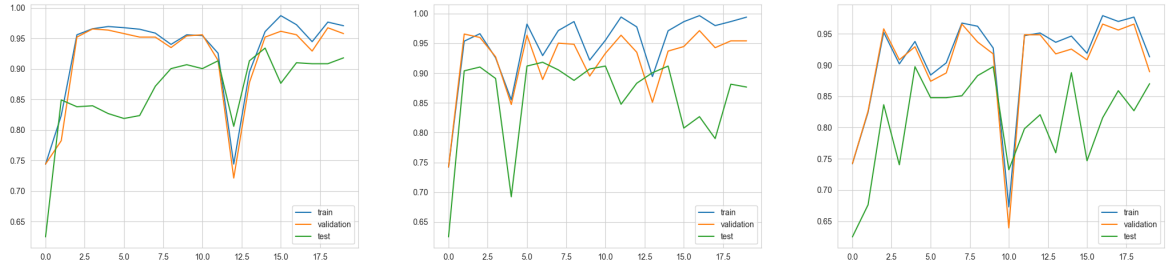


Figure 6.1: Training curve of a sample of architectures from the NAS-Bench-201 search space. The x-axis represents the number of epochs and the y-axis the accuracy.

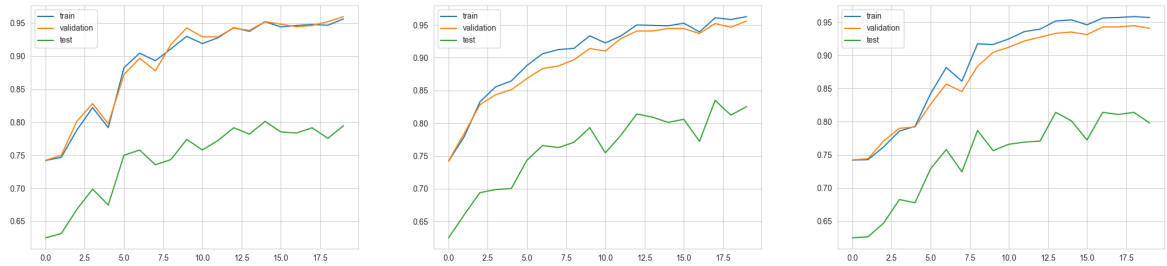


Figure 6.2: Training curve of a sample of architectures from the NAS-Bench-301 search space. The x-axis represents the number of epochs and the y-axis the accuracy.

Optimizing the learning rate

With the search space defined, the next consideration is setting the right Learning Rate (lr). This factor influences the training dynamic and heavily affects both the time of the search and the ability to rank architectures. A learning rate that is too high causes big noisy evaluations and makes it difficult to measure the true potential of various architectures. On the other hand, a learning rate that is too low could lead to slow convergence or to the model becoming trapped in local minima. To choose the right learning rate, we sample random architectures from our selected search space and plot their training curves for various lr values. The values we picked for the learning rate range from 0.01 to $1e-5$, as they are the most found in the literature. Figure 6.3 gives a representative sample of the architectures trained with different lr values. As it is shown, low values of the learning rate ($1e-4$ and $1e-5$) either trap the performance in a local minimum or barely progress for several epochs. On the other hand, the highest value of the learning rate (0.01) creates fluctuations in the training curve which might create biases in ranking architectures during the search process. The smoothest training curve with appropriate progress of performances is found with the lr value of 0.001. It is this value that we will set for training during our work.

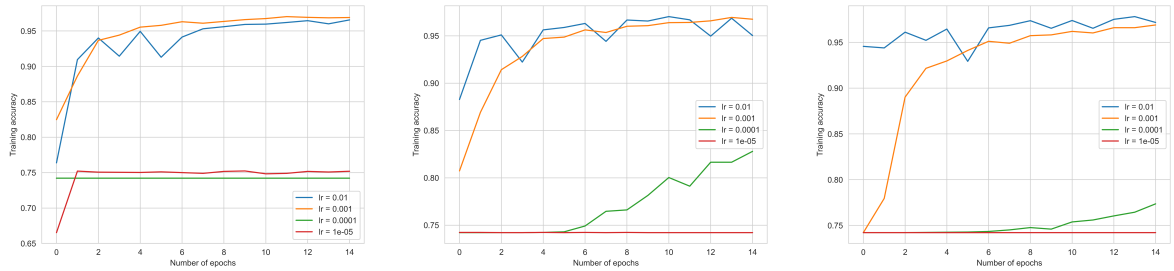


Figure 6.3: Training curve of a sample of architectures from the NAS-Bench-301 search space using different learning rates. The best learning rate is the one in orange as it gives the smoothest curve with appropriate progress.

Choosing the right epoch for a significant proxy

When evaluating architectures during the NAS process, it is usually very computationally prohibitive to fully train each of them (for example, training one architecture for 200 epochs on dermaMNIST takes around 4 hours on our GPU). As we have seen in the first chapter, many techniques are available to reduce the time required for evaluation. The most straightforward approach is using a small number of epochs to estimate the performance. This shortened training offers a quick estimate of an architecture's potential. The choice of the right number of epochs for this evaluation is important. It should be long enough to offer a meaningful insight into the architecture's potential, yet brief enough to maintain efficiency. Using strategies like monitoring validation performance trends can aid in making this decision. To find this value, we also ran several randomly sampled architectures and monitored their trainings. We have set to use the value of 5 epochs for each evaluation. The result obtained after this shortened training will help us determine the potential performance of the architectures during the search.

6.3.3 Experimental protocol

The last part is to run a NAS search algorithm with the previously set components. In this section, we outline our experimental protocol. It describes the methodology we adopted, the steps we took for each evaluation, and the criteria we used to compare and assess the performance during our NAS process. The protocol (summarized in Figure 6.4) includes the following key steps:

- **Runs & Evaluations:**
  - Ten runs are conducted to achieve statistical significance.
  - 1,000 architecture evaluations are performed during each run.
  - For each evaluation, the network is trained for five epochs using the training set.
  - The accuracy of the validation set is recorded after training. This accuracy serves as an indicator of potential full-training performance and is used by the search strategy.

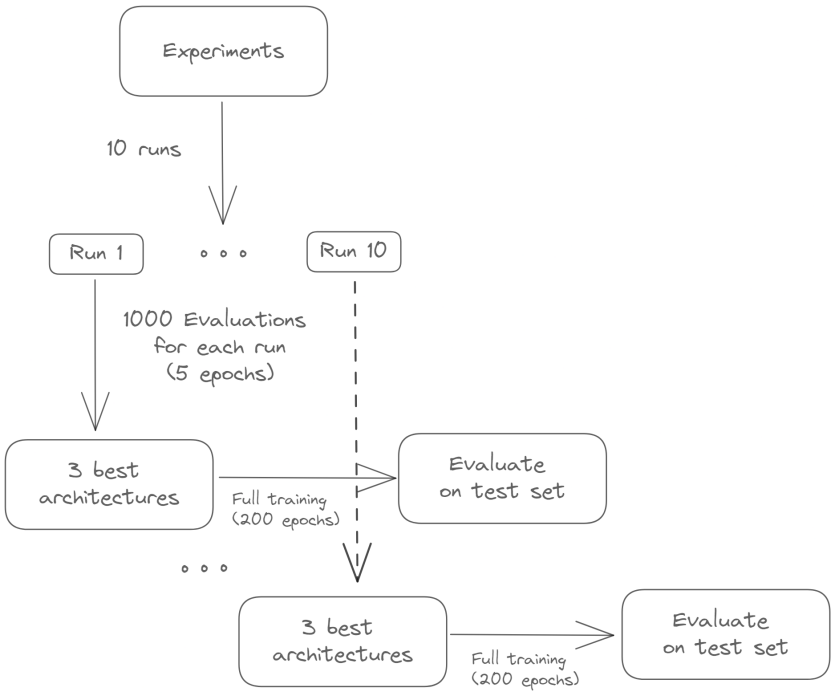


Figure 6.4: Experimental protocol

- **Post-Evaluation Training:**
  - After the algorithm stops, the top three architectures are selected and trained for 200 epochs. After that, they are assessed on the test set and the best of the three is kept at the end.
  - These architectures undergo full training at four evaluation milestones: 200, 500, 750, and 1,000 evaluations. This permits an insight on how the test performance progresses on the fully trained networks.
  - A mean Accuracy (ACC) and Area Under the Curve (AUC) [1] is calculated using the best architecture of each run at 1000 milestone. These metrics are compared to the ones of other models provided by the MedMNIST repository.

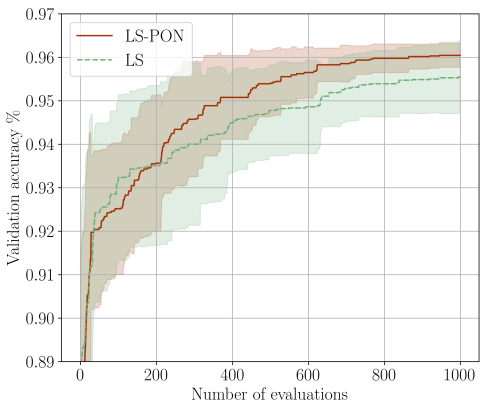
6.3.4 Results

PneumoniaMNIST dataset

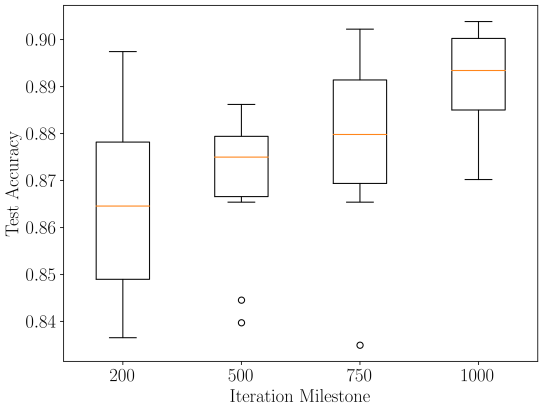
For the PneumoniaMNIST dataset, we run two NAS algorithms using the protocol mentioned above (For time efficiency, we will only do the Post-Evaluation step for the best algorithm). The first is a simple local search (the same used in the comparison in Chapter 2), and the second is our

[1] This metric quantifies the overall ability of a model to distinguish between positive and negative classes, with a value of 1 indicating perfect classification and 0.5 representing a random classifier

proposed approach LS-PON presented in Chapter 2. We compare these two algorithms to validate the efficiency of our approach outside the scope of benchmarks.



(a) Evolution of the validation accuracy of the best architecture found during the NAS process (with 5 epochs of training)



(b) Results on the test accuracy after full training and selecting the best architecture at different milestones during the LS-PON

Figure 6.5: Results on PneumoniaMNIST dataset

Figure 6.5 shows the results of this study. On the left (Figure 6.5.a), we plotted the progress of the validation accuracy (after 5 epochs training) across the number of evaluations for both LS-PON and LS, it represents the mean and standard deviation of the 10 runs of each method. We see in the beginning that the two methods are similar, with LS slightly surpassing LS-PON in the first 200 evaluated architectures. After that, LS-PON seems to be finding better architectures throughout its search compared to LS. With this result, we validate that LS-PON can be more efficient than LS even when using real-world data. We take the top three architectures from the LS-PON algorithm at different milestones and fully train them. After assessing their performance on the test set, we keep the best architecture of the three. The boxplot in Figure 6.5.b shows how the values of test accuracy of the best architectures (fully trained) progress at different milestones. We use boxplots to represent the results from the 10 runs. These results further confirms that our search is indeed finding better architectures with time. Finally, we compare the best architecture at the last milestone with the provided models from the MedMNIST dataset. Table 6.1 presents this finding.

Table 6.1: The table indicates the Area Under the Curve and the Accuracy on the Pneumonia test set for each model. (28) or (224) corresponds to the size of the input image. The images are resized to 224 x 224 by PIL.Image.NEAREST option using the Pillow package. Results of the other methods are found in (Yang et al., 2023).

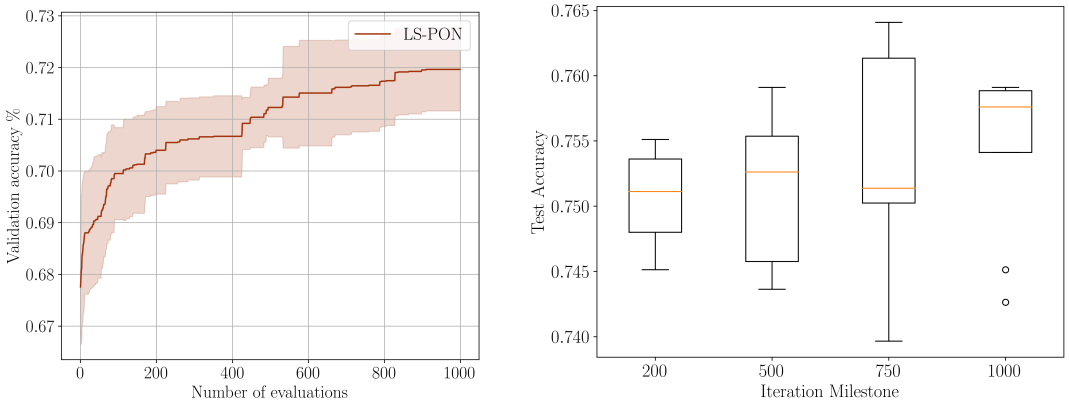
Method	AUC	ACC
ResNet-18 (28)	0.944	0.854
ResNet-18 (224)	0.956	0.864
ResNet-50 (28)	0.948	0.854
ResNet-50 (224)	0.962	0.884
Auto-sklearn	0.942	0.855
AutoKeras	0.947	0.878
Google AutoML Vision	**0.991**	**0.946**
Our best architecture	**0.968**	**0.891**

Aside from the commercial solution Google AutoML Vision which is superior, the best architecture found surpasses all the other models in both AUC and Accuracy. Whether they were carefully handcrafted by experts such as ResNet architectures, or automatically created from libraries such as Auto-sklearn and AutoKeras. This proves that even using a pre-defined search space from a benchmark, we were able to achieve good results using our method and protocol for this dataset.

DermaMNIST dataset

In this part, we investigate the result obtained on a new dataset without changing the search space and other metrics we tailored for the previous one. For time efficiency, we only run the LS-PON algorithm and assess the performance of the obtained architectures. The goal of this experiment is to see if the elements adjusted for pneumoniaMNIST can be generalized to dermaMNIST, with good convergence and results in line with the literature. This will help us determine the transferability of certain components when we change the nature of the medical images in terms of modality and associated tasks. Figure 6.6 shows the results of our experiment. On the left (Figure 6.6.a),

presents the progress of the validation accuracy (after 5 epochs of training) across the number of evaluations for LS-PON. This plot shows an increase in the validation accuracy across the evaluations. On the right (Figure 6.6.b) we see the test accuracies slowly increasing at each milestone in the fully trained architectures. As seen in Table 6.2, we were able to obtain comparable results to the literature. This shows that there is still some level of transferability for this framework. A better adaptation could be possible, but due to the limited computing power we had, tailoring for this larger and more difficult task is time-consuming.



(a) Evolution of the validation accuracy of the best architecture found during the NAS process (with 5 epochs of training)

(b) Results on the test accuracy after full training and selecting the best architecture at different milestones during the LS-PON

Figure 6.6: Results on DermaMNIST dataset

Table 6.2: The table indicates the Area Under the Curve and the Accuracy on the DermaMNIST test set for each model. (28) or (224) corresponds to the size of the input image of the model. The images are resized to 224 x 224 by PIL.Image.NEAREST option using the Pillow package. Results of the other methods are found in (Yang et al., 2023).

.

Method	AUC	ACC
ResNet-18 (28)	0.917	0.735
ResNet-18 (224)	**0.920**	0.754
ResNet-50 (28)	0.913	0.735
ResNet-50 (224)	0.912	0.731
Auto-sklearn	0.902	0.719
AutoKeras	0.915	0.749
Google AutoML Vision	0.914	**0.768**
Our best architecture	**0.918**	**0.754**

6.4 Conclusion

In this chapter, the challenge was to extend our contribution to real-world applications, specifically in the realm of healthcare data. The transition from benchmark scenarios to real applications required new considerations and did not allow the use of pre-calculated results. And these elements have been proven as crucial determinants in the efficiency of any NAS framework. The computational resources when making this chapter were very limited. For this reason, a more extensive study is possible to try to tailor the different components to other datasets and extract any patterns noticed in the nature of the task or dataset. Our method, LS-PON, was validated as more efficient than LS for the context of PneumoniaMNIST, and was able to find results that are comparable to state-of-the-art models. When moving to another real-world dataset with different modalities and tasks, the NAS framework was also found efficient in moving to better areas in the search space. This highlights a certain transferability of the fine-tuned elements of the NAS framework. A perspective to this work is to create a meta-learning approach that automatically selects the appropriate parts, starting from a dynamic search space to an adaptive learning rate and epochs setting.

Chapter 7

Neural Architecture Search in Healthcare: An Introductory Survey to Unlocking Opportunities

"Study the past if you would define the future."

— Confucius

This chapter contains the work under review in the journal *Artificial Intelligence Review*

- Meyssa Zouambi, Clarisse Dhaenens, Louis Rousselet, Julie Jacques. Neural Architecture Search for Healthcare: a Comprehensive Survey. Journal paper submitted to Artificial Intelligence Review.

7.1 Introduction

After discussing the approach of using NAS in real-world scenarios, the best strategy to contribute to a new field using NAS is to study its literature for domain applications. In this thesis, we chose to focus on the healthcare sector. The unique nature of healthcare data as well as its prominent use created the need for tailored deep learning models. This makes it a high-potential domain that can greatly benefit from the use of NAS. The purpose of this chapter is to provide a structured starting point for those new to the field or considering leveraging NAS for healthcare applications. We highlight not only the potential of NAS in healthcare but also underscore the current gaps, setting the stage for future exploration and contribution.

This chapter delves into the literature, drawing insights from over 40 different contributions that have used automatically designed neural networks in healthcare applications. Using the comprehensive paper database of neural architecture search found in (Freiburg-Hannover, 2021a), we made a thorough review process. We started by selecting studies related to healthcare and examined their titles and abstracts. This allowed us to narrow down our focus and ensure that the

contributions we discuss are most relevant to the field.

7.2 Context

The growth of data in the healthcare sector, driven by electronic health records, wearable devices, medical imaging, and various other sources, has made utilizing advanced machine learning techniques a promising path. It has the potential to improve patient care, prognosis, and medical research. Traditional deep learning models, which have achieved significant successes in tasks such as medical image classification, segmentation, and anomaly detection, are generally hand-crafted by human experts. However, these manual designs often do not optimize for domain-specific characteristics found in medical data, which may differ significantly from other data types. The potential of neural architecture search in modern machine learning applications is vast, but its exploration within healthcare holds particular promise. Given the domain's sensitive nature, tailoring neural networks to the challenges of healthcare datasets is important.

To structure our literature review, the following sections group three main application areas: lesion detection and classification, image segmentation, and healthcare-supporting tasks. This structure helps readers quickly understand the primary avenues of NAS application in healthcare and identify specific areas they might be interested in.

7.3 Lesion detection and classification

The application of neural architecture search has made significant progress in various healthcare challenges, most notably in lesion detection and classification. Starting with skin conditions, Kwasigroch et al. (2020) used a hill-climbing algorithm for the detection of malignant melanoma from dermoscopic images. Their approach starts from a base architecture and then builds on it using function-preserving transformations. This specific NAS algorithm was later applied to a Chest-X ray classification model in (Radiuk and Kutucu, 2020), leveraging the CheXpert (Irvin et al., 2019) benchmark. In a similar domain, for skin disease classification, He et al. (2021b) introduced a NAS method anchored in an evolutionary algorithm. Their approach was tested on a real-world skin disease image dataset to verify its effectiveness.

Branching out to other medical imaging challenges, we found that several works have focused on pulmonary nodule classifications. Jiang et al. (2021) refined an initial neural network with the Partial Order Pruning method and integrated an Angular Softmax loss function to enable better learning of discriminative representations (Liu et al., 2017b). The authors also used Convolutional Block Attention Modules (CBAM) (Woo et al., 2018) for explainability. Similarly, in (Hu et al., 2021), differentiable NAS was combined with an attention mechanism in CNNs for improved feature representation in pulmonary classification.

With the onset of the Covid-19 pandemic, there was increased attention to diagnosing infections. He et al. (2020) tackled this challenge by searching for an effective 3D CNN model using differentiable NAS, further adding *Class Activation Map* for explainability. Building on their prior

work, they later integrated a multi-objective evolutionary method for enhanced efficiency (He et al., 2021a).

Some NAS research has dived into more intricate medical challenges. In (Jiang et al., 2020), the detection and delimitation of lesions in CT scans were tackled using a differentiable NAS strategy. This task is similar to object detection but is much more complex in the context of medical imagery. The authors' approach was augmented by modules to address data imbalance and enhance network generalization.

Additionally, NAS has been used to find networks for diagnosing neurological and genetic conditions. Lee et al. (2021) employed a genetic algorithm to determine the best CNN architecture for Alzheimer's disease diagnosis in amyloid brain images. Meanwhile, in (Miahi et al., 2019), a genetic algorithm was adopted to find the best network for sperm abnormality detection.

Heart-related anomalies have also been an important research application. In (Lv et al., 2021), the Darts algorithm was utilized for irregular heartbeat classification. Similarly, Odema et al. (2021) used a NAS algorithm to detect myocardial infarction on wearable devices.

Lastly, some unique applications include the work of Wang et al. (2020) which introduced a teleoperated ultrasound system for Hepatic Echinococcosis diagnosis, enhancing it with Darknet53 for lesion detection.

7.3.1 Our takeaway

The previously presented works illustrate the versatility of NAS across a wide variety of lesion detection and classification tasks. These studies include a broad range of data types, ranging from image data from scans and microscopic snapshots, to sequential data formats like ECG signals and rsfMRI time series. Specific challenges related to class imbalance and data quality are tackled in certain contributions (Lin et al., 2017; Miahi et al., 2019). Notably, a few researchers (Jiang et al., 2021; He et al., 2021a) have also emphasized the importance of explainability, a crucial factor in high-stakes domains like healthcare.

To further illustrate the extent of NAS applications in healthcare, we summarized the tasks within this category in Table 7.1. This table extracts important information about the data utilized and additionally notes the availability of the code for potential adopters. It showcases a broad spectrum of healthcare tasks and the data types associated with them.

Meanwhile, Table 7.2 dives deeper into the technical aspects of NAS. It outlines the deep learning tasks and corresponding architectures, in addition to detailing the search strategies, optimized parameters, and NAS execution times where available. A significant observation from this table reveals the predominance of CNNs, confirming the image-centric nature of most data in this category. Most researchers also seem to gravitate towards evolutionary algorithms or gradient descent as their primary search strategy. On top of that, several works incorporated objectives like energy consumption, model size, or inference time (Odema et al., 2021; He et al., 2021a; Jiang et al., 2021).

Table 7.1: Applications of lesion detection and classification papers

Paper	Healthcare Task	Type of Data	Dataset	Dataset Specifications	Code available
Kwasigroch et al. (2020)	Skin lesion classification (malignant melanoma)	Dermoscopic images	International Skin Imaging Collaboration	12500 benign instances; 1100 malignant instances; Images annotated by experts.	No
Radiuk and Kutucu (2020)	Chest X-Ray Classification (various diseases)	Chest X-Ray	CheXpert benchmark	224,316 chest Xrays, 65,240 patients; Size 320×320 pixels; Images labeled for 14 diseases as negative, positive, or uncertain.	Yes (final architecture only)
Dai et al. (2020)	Classification of fMRI signals	rsfMRI time series	Autism Brain Imaging Data Exchange (ABIDE II)	MRI Scan of 58 subjects	No
He et al. (2021b)	Skin disease classification	Clinical images	XiangyaDerm Dataset	150,223 clinical images, 543 categories of skin diseases	No
Jiang et al. (2021) Hu et al. (2021)	Classification of pulmonary nodules	Thoracic CT scans	LIDC-IDRI database	CT scans with marked-up annotated lesions containing 1018 cases	Yes
He et al. (2020) He et al. (2021a)	COVID-19 Detection	3D CT Chest Scans	Clean-CC-CCII MosMedData COVID-CTset	526 scans of 377 patients with 244 COVID-19 positive scans; 1110 scans of 1110 patients with 856 COVID19 positive scans; 4356 scans of 2778 patients with 1578 COVID19 positive scans	Yes
Jiang et al. (2020)	Lesion Detection	CT scans	DeepLesion Kits19	876,934 axial CT slices (size 512 × 512) from 10,594 CT studies of 4,427 patients; 32,120 axial slices with bounding boxes annotation; 1–3 lesions in each axial slice; total 32,735 lesions. 71,423 CT slices of 210 unique patients	No
Lee et al. (2021)	Alzheimer's disease diagnosis	PET/CT amyloid brain images	Private dataset	Images of 414 patients diagnosed as Normal Control, Mild Cognitive Impairment, and Alzheimer's disease.	No
Miahi et al. (2019)	Sperm abnormality detection	microscope images	MHSMA dataset	1, 540 sperm images from 235 patients with infertility problems	No

Lv et al. (2021)	Arrhythmia detection	ECG signals	MIT-BIH arrhythmia database and INCART and QT databases	Heartbeats are classified into five classes: Normal beats, Supraventricular ectopic beats, Ventricular ectopic beats, Fusion beats, and Unclassified beats.	No
Odema et al. (2021)	Myocardial Infarction Detection	ECG signals	PTB diagnostic ECG database	200 subjects, 148 with Myocardial Infarction.	No
Wang et al. (2020)	Hepatic Echinococcosis diagnosis	ultrasound	Private dataset	9112 Hepatic Echinococcosis ultrasound images from 5028 patients	No

Table 7.2: NAS details for Lesion detection and classification papers

Paper	**Task**	**Network Searched**	**Parameters searched**	**Search strategy**	**Objective(s)**	**Search time**
Kwasigroch et al. (2020) Radiuk and Kutucu (2020)	Image classification	CNN	Number and type of layers, number of filters in Conv layers, and network structure (add and concatenation schemes and skip connections).	Hill climbing	Validation Loss	18 GPU hours and 28 GPU hours
Dai et al. (2020)	1D classification	CNN	Number of layers, type of operation in each layer	Gradient descent optimization	Accuracy	-
He et al. (2021b)	Image classification	CNN	Parameters of three types of layers (convolutional, pooling, and fully connected layers)	Evolutionary algorithm	Classification error rate	-
Jiang et al. (2021)	Image classification	3D CNN	Number of residual blocks and number of channels in each of them	Partial Order Pruning	Accuracy inference time	-
Hu et al. (2021)	Image classification	CNN	Operations composing normal and reduction cells	Gradient descent optimization	Validation Loss	-
He et al. (2020)	Image classification	3D CNN	Kernel size and expansion ratio of convolution modules, inside each of the cells composing the backbone architecture.	Gradient descent optimization	Validation Loss	8 GPU hours
He et al. (2021a)	Image classification	3D CNN	Cell structure	Evolutionary algorithm	Accuracy Model size	8 GPU hours

Jiang et al. (2020)	Object detection	CNN	Operations composing normal and reduction cells	Gradient descent optimization	Accuracy and L1	-
Lee et al. (2021)	Image classification	CNN	Network structure and hyperparameters	Evolutionary algorithm	Accuracy	-
Miahi et al. (2019)	Image classification	CNN	Number of filters, filters height and width, stride size of each convolutional cell	Evolutionary algorithm	Cross-entropy	310, 162, and 240 GPU hours
Lv et al. (2021)	Classification	CNN	Composition of normal and reduction cells	Gradient descent optimization	Validation Loss	-
Odema et al. (2021)	Classification	CNN	Number of blocks, number of filters, kernel length and stride	Gradient descent optimization	Detection error Energy consumption	-
Wang et al. (2020)	Image classification	CNN	Composition of normal and reduction cells	Gradient descent optimization	Accuracy	-

7.4 Medical image segmentation

Medical image segmentation is the process of highlighting regions or structures of interest within an image, such as tumors, organs, or vessels. Given the unique characteristics of medical images, including the subtle distinction between the region of interest and its background, the task becomes especially complex. This complexity is amplified when the data extends beyond two dimensions. Researchers increasingly leverage NAS to discover architectures tailored for 2D and 3D medical segmentation.

7.4.1 2D segmentation

Many architectures have been developed for 2D medical image segmentation using NAS. In a pioneering study by Gessert and Schlaefer (2019a), the authors developed a network for 2D depth OCT retinal segmentation using 1D depth data. Their objective was to demonstrate that NAS on lower-dimensional data could produce architectures that generalize well on higher-dimensional data. Their findings revealed that networks optimized using 1D depth OCT scans could match the performance of those developed using 2D depth scans, but with an 87.5% speed advantage. The ENAS framework Pham et al. (2018) underpinned their search strategy, aiming to discover optimal building blocks for a U-shaped architecture.

Several other works have contributed to the domain of retinal segmentation. For instance, Fan et al. (2020) used an evolutionary-based NAS paired with a focal loss (Lin et al., 2017) to address data imbalance. The resulting architecture was evaluated on the DRIVE, STARE, and CHASEDB_1 datasets. In another notable work, Gheshlaghi et al. (2020) employed Proxyless NAS

(Cai et al., 2018) to determine the optimal U-shaped architecture for 2D OCT retinal segmentation. This was preceded by a data enhancement phase where OCT images underwent super-resolution transformations, leveraging algorithms like Enhanced Deep Residual Super Resolution (Lim et al., 2017), among others.

Outside the area of retinal imaging, Mortazi and Bagci (2018) proposed a NAS approach optimized by a policy gradient reinforcement learning for cardiac MR image segmentation. Their framework refined a densely connected encoder-decoder structure (Huang et al., 2017) and was tested using the Automated Cardiac Diagnosis Challenge (ACDC) MICCAI 2017 dataset.

In the realm of electron microscopy segmentation, Xu et al. (2020b) proposed a NAS framework. This approach sought to explore the best architecture using a recurrent neural network-based NAS, taking inspiration from ENAS (Pham et al., 2018). As part of their search space considerations, the authors included operations that make up the upsampling, downsampling, and bridge cells of the network as well as intra and inter connections between cells.

Another intriguing work by Dong et al. (2019) delved into the task of chest organ segmentation using adversarial networks. This technique hinged on discriminating segmentation maps derived from either the ground truth or the segmentation network. This iterative comparison propelled improvements in the segmentation process at every step. To enhance efficiency, the process of designing the discriminator's architecture was automated using differentiable NAS.

Moving further, Liu et al. (2021) introduced a generalized segmentation architecture tailored for multi-domain medical data, integrating multiple small-scale datasets. Such an approach was envisaged to counteract the challenge of data scarcity and to craft a versatile segmentation network, aptly aligning with the broad variations inherent in multi-domain data. After curating this mixed dataset, the architecture search was bifurcated for both the macro and micro structures of the network, with differentiable NAS being the method of choice. The outcomes were remarkable as the resulting network outperformed the established U-Net network across various tests, spanning dermoscopic images, gastroscopy images, and CT scans.

More 2D segmentation networks are proposed and tested in diverse datasets, for example, Weng et al. (2019) uses ProxylessNAS algorithm to search for a segmentation architecture using the PASCAL VOC 2012 dataset and evaluates it on various medical data types: CT scans, MRI, and ultrasounds. In (Yan et al., 2020), authors use a multi-scale search space to include both the architecture backbone and cell operations in their search, including connections to fuse features with different sizes. This work also tests their resulting model across with CT and MRI scans of PROMISE12 Litjens et al. (2014), SILVER07, and CHAOS Kavur et al. (2021) datasets.

Our takeaway

Researchers show that there are many challenges when searching for optimal segmentation networks. Data volume was addressed in Gessert and Schlaefer (2019a) by working with lower dimensional data. In Gheshlaghi et al. (2020), authors pre-processed the scans with multiple techniques to enhance the data quality of medical images. In Liu et al. (2021), authors deal with data scarcity using multi-domain medical data. This shows that in healthcare, segmentation is challenging and

requires additional efforts compared to segmentation for regular images.

Table 7.3 and Table 7.4 summarize the same details mentioned in the previous tables of this section. Table 7.3 covers application details for 2D segmentation papers, underscoring that organ segmentation remains a prominent application focus. Table 7.4 gives more technical details regarding NAS. It confirms that U-shaped CNNs are the most used network for segmentation. It also shows that gradient descent is extensively used as a search strategy.

Table 7.3: Applications of 2D segmentation papers

Paper	**Healthcare Task**	**Type of Data**	**Dataset**	**Dataset Specifications**	**Code available**
Gessert and Schlaefer (2019a)	OCT Image Segmentation	SD-OCT scans	(Farsiu et al., 2014)	Individual SD-OCT images and marking: 38400 BScans from 269 AMD patients and 115 normal subjects. Layer boundaries provided for the inner limiting membrane, retinal pigment epithelium drusen complex and Bruchs membrane.	No
Fan et al. (2020)	Retinal Vessel Segmentation	OCT scans	Drive Stare Chase_DB1	40 fundus images with 768 x 584 pixels resolution 20 color fundus images of 700 × 605 pixels 28 eye fundus images with 1280 x 960 pixels resolution.	No
Gheshlaghi et al. (2020)	Retinal layer segmentation	OCT scans	Private dataset	19 OCT scans of 45 subjects	No
Mortazi and Bagci (2018)	Segmentation of cine cardiac MR images	Cine Cardiac MR	Automated Cardiac Diagnosis Challenge MICCAI 2017	150 cine-MR images: 30 normal cases, 30 myocardium infarction, 30 dilated cardiomyopathies, 30 hypertrophic cardiomyopathies, and 30 abnormal right ventricles.	No
Xu et al. (2020b)	Segmentation of neuronal processes in EM images	Serial section Transmission Electron Microscopy data	Dataset of larva ventral nerve cord from the ISBI Challenge	30 sections from a serial section Transmission Electron Microscopy (ssTEM). Microcube measures 2 x 2 x 1.5 microns approx., with a resolution of 4x4x50 nm/pixel. The labels given are fully annotated binary masks.	No
Dong et al. (2019)	Chest organ segmentation	Chest X-ray	CX-SEG	259 Chest X-ray images; pixel-wise ground truth labels of left lung	No

			JSRT-DB	247 Chest X-ray images; pixel-wise ground truth labels of left lung, right lung and heart; 2048 × 2048 pixels resolution	
Liu et al. (2021)	Segmentation of multiple types of medical images	Dermoscopic images	ISIC	2,594 dermoscopic images	Yes
		gastroscopy images	CVC Bernal et al. (2017)	612 still images from 29 different sequences	
		CT scans	CHAOS-CT	–	
Weng et al. (2019)	Organs and nerves segmentation	MRI Scans	Promise12	50 cases; include a transversal T2-weighed MR images of the prostate.	Yes
			Chaos-MR	Liver, kidneys and spleen CT scan; 1270 images; 20 training cases	
		Ultrasound	Ultrasound nerve dataset ner (2016)	11270 images with binary mask showing the Brachial Plexus segmentations.	
Yan et al. (2020)	Organ segmentation	CT scans	Sliver07	8318 Liver CT scans	No
		MRI scans	Promise12	1377 Prostate MRI scans	
			Chaos-MR	1270 images of: Liver, kidneys and spleen MRI scans	

Table 7.4: NAS details for 2D segmentation papers

Paper	**Task**	**Network Searched**	**Parameters searched**	**Search strategy**	**Objective(s)**	**Search time**
Gessert and Schlaefer (2019a)	Image segmentation	U-shaped CNN	Properties of 2 cells each containing 2 subcells (operations and connections)	Reinforcement learning	Dice score	1.5 GPU hours
Fan et al. (2020)	Semantic image segmentation	U-shaped CNN	Operations of the upsampling and down sampling cells, activation functions, shortcut connections inside the cells, and the skip connections with the preceding cells	Evolutionary algorithm	F1-score	-
Gheshlaghi et al. (2020)	Image segmentation	U-shaped CNN	Operations of the upsampling and down sampling cells	Gradient descent optimization	Dice Similarity Coefficient	24 GPU hours

Mortazi and Bagci (2018)	Image Segmentation	Densely connected encoder-decoder CNN	Number of filters, filter height, and filter width for each layer, type of pooling layer.	policy gradient reinforcement learning	Dice index	3600 GPU hours
Xu et al. (2020b)	Image segmentation	U-shaped CNN	Operations of upsampling and downsampling and bridge cells and intra-cell and inter-cell skip connections	Recurrent neural network	IoU score	8.75 GPU hours
Dong et al. (2019)	Image segmentation	Adversarial network	Structure of the cells composing the discriminator	Gradient descent optimization	Cross entropy loss/ adversarial loss	144 GPU hours
Liu et al. (2021)	Image segmentation	U-shaped CNN	Operations of upsampling, downsampling and normal cells and aggregations	Gradient descent optimization	Weighted sum of cross entropy loss and Dice loss	45.6 GPU hours
Weng et al. (2019)	Image segmentation	U-shaped CNN	Operations of the upsampling and down sampling cells	Gradient descent optimization	Dice Similarity Coefficient	8 GPU hours
Yan et al. (2020)	Image segmentation	U-shaped CNN	Network structure and cells composition	Gradient descent optimization	Accuracy	48 GPU hours

7.4.2 3D segmentation

In deep learning, working on the segmentation of volumetric data is often problematic. Because of their size, they require a lot more computational power for processing. Most approaches for segmentation either use 2D FCN, which does not fully exploit the 3D context of the data, or use 3D FCN, which is more appropriate but requires a lot more computational resources.

Addressing this, Zhu et al. (2019) introduced a NAS method that dynamically selects between 2D, 3D, or Pseudo-3D (Qiu et al., 2017) convolutions for each layer. By doing so, the network combines the strengths of both 2D and 3D architectures. The algorithm relies on differentiable neural architecture to search for the best encoder and decoder blocks for the network architecture. The result was validated on lung and pancreas segmentation tasks, focusing on organs and tumors.

Taking a different approach, Baldeon Calisto and Lai-Yuen (2020) revealed an adaptive 2D-3D multi-objective evolutionary algorithm. It tailors its segmentation strategy according to the nature of input data (2D or 3D). It also optimizes both accuracy and network efficiency. The framework is tested on the task of prostate segmentation and cardiac segmentation using the PROMISE12 Litjens et al. (2014) and MICCAI ACDC datasets. A subsequent iteration of this work (Baldeon

Calisto and Lai-Yuen, 2021) introduced a Random Forest surrogate function to enhance the search process.

In the area of cardiac interventions, Zeng et al. (2020) designed a strategy that finds the most suitable architecture for real-time 3D cardiac segmentation. The resulting architecture must yield good performances while avoiding any visual lag during the real-time segmentation. This was achieved using a differentiable NAS that incorporated a latency-driven regularization component.

In Wang (2020), the focus is on glioma segmentation using a patching strategy for multi-model MRI scans, leveraging a back-propagation algorithm on the BraTS 2019 dataset. Building on segmentation themes, Guo et al. (2020) addresses organ at risk in the head and neck. By adopting the clinical organ at risk (OAR) delineation approach, they segment organs into anchor, mid-level, and small and hard (S&H) categories, optimizing their transitions with differentiable NAS and 2D, 3D, or P3D convolutions.

Highlighting the importance of understanding data specifics, Bae et al. (2019) pinpoints variations in 3D shapes across medical images. Their NAS approach, influenced by ENAS Pham et al. (2018), permits dynamic input selection and downsampling, ensuring efficiency as child networks inherit parent weights. Testing this, they benchmarked their results on the medical segmentation decathlon challenge.

Further exploring this challenge, Ji et al. (2020) delves into brain, heart, and prostate datasets. Their innovative approach uses multi-level feature aggregation in a U-shaped network combined with gradient-descent optimization to maximize data insights. On the other hand, Yu et al. (2020) introduces a fresh perspective with a two-stage search. First, the network maps out the overall structure, aiming beyond the ubiquitous U-shape, then refines the internal cellular structure. This duality uses an evolutionary strategy and a single-path one-shot NAS, with findings validated across several datasets.

In a quest for flexibility, He et al. (2021d) pursues a 3D segmentation design that deviates from the conventional U-shape. They integrate topology loss for optimization and ensure GPU adaptability. Concluding this discourse, Kim et al. (2019) present a scalable gradient descent method that zeroes in on neural connectivity in both encoders and decoders, a technique further validated on tasks like the MSD challenge Freiburg-Hannover (2021b).

Our takeaway

These works emphasize the distinct challenges presented by 3D healthcare segmentation. Most significantly its computational demands. To deal with this, most authors use NAS for building hybrid models containing 2D, 3D, and Pseudo-3D convolutions. Although some scans can be of the same nature, the difficulty of segmentation differs depending on the target organ/tumor. Some work mimics the practices of health professionals for segmentation and translates that into a DL model. Some authors also use multi-objective approaches to optimize memory usage or inference time which are important in certain tasks and settings in segmentation (He et al., 2021d; Zeng et al., 2020).

Tables 7.5 and 7.6 encapsulate the nuances of the discussed papers, reproducing the earlier ta-

bles. From the 3D segmentation research, it is evident that organ segmentation remains a prominent area of focus. The Medical Segmentation Decathlon (MSD) challenge is a recurring benchmark, with most models incorporating Fully Convolutional Networks (FCN) featuring an amalgamation of 2D, 3D, and Pseudo-3D modules.

Table 7.5: Applications of 3D segmentation papers

Paper	**Healthcare Task**	**Type of Data**	**Dataset**	**Dataset Specifications**	**Code available**
Zhu et al. (2019)	Organ and tumor segmentation	3D CT scans	NIH Pancreas Dataset	82 normal abdominal CT volumes (size 512 x 512 x D); D ranging in [181, 466]	No
			MSD Lung Tumors	96 3D CT volumes from the MSD challenge of small tumors segmentation task.	
			MSD Pancreas Tumors	420 3D Portal venous phase CT volumes	
Baldeon Calisto and Lai-Yuen (2020) Baldeon Calisto and Lai-Yuen (2021)	Segmentation of cine cardiac MRI and Prostate segmentation	MRI scans	Cardiac Diagnosis MICCAI 2017 PROMISE12	150 cine-MR images: 30 normal cases, 30 myocardium infarction, 30 dilated cardiomyopathies, 30 hypertrophic cardiomyopathies, 30 abnormal right ventricle (RV). 50 transverse T2-weighted MR images of the prostate and their corresponding reference segmentations	No
Zeng et al. (2020)	Real-time MRI segmentation	Cine Cardiac MRI	Cardiac Diagnosis MICCAI 2017	150 cine-MR images: 30 normal cases, 30 myocardium infarction, 30 dilated and 30 hypertrophic, 30 abnormal right ventricle (RV).	No
Wang (2020)	Brain tumor segmentation (Gliomas)	Multimodal volumetric MRI scans.	BraTS 2019 dataset	4D Scans containing high grade glioma (HGG) or low grade glioma (LGG) from 19 institutions. MRI modalities: T1-weighted, T2-weighted, FLAIR and T1Gd. Ground truth labels of 259 HGG and 76 LGG	Yes
Guo et al. (2020)	Organ at risk segmentation in head and neck	CT scans	-	142 RTCT images with 42 annotated OARs	No
Bae et al. (2019) Ji et al. (2020)	Brain tumor, heart and prostate segmentation	MRI scans	MSD Brain	750 4D volume with modalities: FLAIR, T1w, T1gd,T2w	No
			MSD Heart	30 3D volumes	
			MSD prostate	48 4D volumes: Prostate central gland and peripheral zone	

Yu et al. (2020) He et al. (2021d)	Medical image segmentation	MRI and CT scans	All tasks of the MSD Challenge	Refer to (Freiburg-Hannover, 2021b) for full details	No
Kim et al. (2019)	Brain tumor, heart and lung segmentation	MRI and CT scans	MSD Brain MSD Heart MSD Lung	484 labeled images, 3 classes 20 labeled images, 1 class 20 labeled images, 1 class	No

Table 7.6: NAS details for 3D segmentation papers

Paper	**Task**	**Network Searched**	**Parameters searched**	**Search strategy**	**Objective(s)**	**Search time**
Zhu et al. (2019)	Volumetric Image Segmentation	Hybrid 2D, 3D and Pseudo-3D (P3D) FCN	Type of encoder (resp. decoder): 2D, 3D or Pseudo 3D inside a fixed architecture	Gradient descent optimization	Dice-Sørensen Coefficient	28.8 GPU hours
Baldeon Calisto and Lai-Yuen (2020)	Volumetric Image Segmentation	2D or 3D FCN	Number of cells, number of filters, type of operation, and connections, learning rate, dropout probability, merge operations, activation functions.	Multiobjective evolutionary-based algorithm	Weighted sum of multiple indicators	117.6 for 2D and 208.8 for 3D (GPU hours)
Baldeon Calisto and Lai-Yuen (2021)	Volumetric Image Segmentation	2D or 3D FCN	Number of encoder-decoder cells, number of filters, type of operation, and connections in each cell, and learning rate.	Multiobjective evolutionary based algorithm		273.6 GPU hours
Zeng et al. (2020)	Real-time 3D segmentation	3D FCN	Cell operations and connections and architecture backbone search	Gradient descent optimization	Latency Dice score	-
Wang (2020)	Volumetric semantic segmentation	U-shaped CNN	Structure of upsampling and downsampling cells	Gradient descent optimization	multi-class Dice loss	-
Guo et al. (2020)	Multi-stage segmentation	Hybrid 2D, 3D and Pseudo-3D (P3D) FCN	Layers structure within a fixed backbone	Gradient descent optimization	Dice score	-
Bae et al. (2019)	Volumetric Image Segmentation	3D FCN	Input patch size, size of 3D pooling and 3D conv operations, skip connections, activation functions	Reinforcement learning	Dice loss	33.36 GPU hours

Ji et al. (2020)	Volumetric Image Segmentation	U-shaped CNN	Operations of each cell type and multi-stage aggregations	Gradient descent optimization	Dice loss	74.4 h
Yu et al. (2020)	Volumetric Image Segmentation	Hybrid 2D, 3D and Pseudo-3D (P3D) FCN	Network structure, and cell operations (2D, 3D or Pseudo 3D)	Evolutionary + Gradient descent optimization	Dice loss Cross entropy loss	7952 GPU hours
He et al. (2021d)	Volumetric Image Segmentation	FCN	Operations of each cell	Gradient descent optimization	Dice-Sørensen score Topology loss	139.2 GPU hours
Kim et al. (2019)	Volumetric Image Segmentation	U-shaped CNN	Operations of each cell	Gradient descent optimization	Jaccard distance	-

7.5 Other medical-related tasks

In the vast expanse of medical research, Neural Architecture Search NAS has been employed in addressing an array of complex challenges and niche domains. In terms of medical prediction, the use of NAS has been explored. Balaprakash et al. (2019) focus on cancer research by proposing a method that takes into account characteristics specific to cancer data. Their algorithm is based on reinforcement learning to search for the best architecture to predict drug response based on molecular features of tumor cells and drug descriptors. They use three benchmarks from the CANcer Distributed Learning Environment (CANDLE) project (can, 2021).

In (Lin et al., 2021), authors work on the 3D radiotherapy dose prediction. This is a complex task because of the conflicting objectives it carries: the target dose prescription, and normal tissue sparing. Their method integrates neural architecture search with knowledge distillation to reduce inference time.

Beyond predictive models, tasks supporting healthcare applications, such as feature extraction, are also important. A notable study by Peng et al. (2020) employed NAS to develop a model to automatically perform this task on multi-modality medical data. This requires a lot of effort in selecting the relevant features and combining them. Their work focuses on the fusion of PET-CT and CT radiomics using an automatically searched network that provides more options for finding and integrating the complementary PET and CT features.

Further expanding on data modeling, Qiang et al. (2020) introduced a deep belief network tailored for modeling task-based fMRI data. Their NAS approach uses particle swarm optimization. The model they developed categorizes seven classes of behavioral tasks, including emotions and language, among others. In parallel, research by Li et al. (2020) utilized an evolutionary algorithm to create a deep sparse recurrent auto-encoder (DSEAE) for a similar purpose.

Li et al. (2021) tackled the challenge of decomposing the brain's spatial/temporal function networks from 4D MRI scans. They incorporated a Recurrent Neural Network (RNN) whose ar-

chitecture was refined using differentiable NAS. They also embedded an early stopping mechanism to address overfitting challenges.

Quality of medical images is also very important, and Liu et al. (2019) acknowledged this in their work on medical image denoising. Their unique NAS-based approach automatically identifies the optimal model for denoising. Their method leans on a genetic algorithm, augmented with an experience-based greedy strategy. Transfer learning is also interwoven to hasten the optimization process, tested specifically for computed tomography perfusion denoising.

Moreover, Huang et al. (2020) ventured into the realm of MRI data reconstruction from undersampled inputs. Their methodology, grounded in differentiable NAS, was meticulously tested on both brain and cardiac MRI datasets.

Lastly, computer-assisted interventions represent a promising frontier in medical technology. Kügler et al. (2020) explored pose estimation for surgical instruments during minimally invasive temporal bone surgery. They employed NAS to optimize an auto-encoder's architecture blocks using gradient ascent. The scarcity of real-world data led them to develop a synthetic dataset, leaving real-world data for validation only. Their findings are based on the i3PosNet Dataset Kügler et al. (2018), which offers real radiographs of surgical instruments.

7.5.1 Our takeaway

The diverse landscape of healthcare data, ranging from gene signatures to radiomics, demands customized architectures. This section underscores that NAS has been used outside traditional data types, adapting unconventional ones like drug descriptors. Whether it is in data modeling, feature extraction, or predictive analytics.

Tables 7.7 and 7.8 provide detailed summaries of the discussed applications and their respective NAS methodologies. These tables highlight the versatility of NAS approaches across varied healthcare tasks, underscoring the wide range of network types.

Table 7.7: Applications of healthcare supporting tasks papers

Paper	Healthcare Task	Type of Data	Dataset	Dataset Specifications	Code available
Balaprakash et al. (2019)	Predicting tumor cell line response	Cell line molecular features and drug descriptors	COMBO benchmark	-	No
	Predicting tumor dose response	Cancer drug screening data	Uno benchmark	-	
	Classifying RNA-seq gene expressions	Gene-expression-level tumor signatures	NT3 benchmark	-	

Lin et al. (2021)	Radiotherapy Dose Prediction	CT scans	OpenKBP dataset	340 CT scans with 7 organs at risk and 3 planning target volumes: gross disease, intermediate-risk, and elective	No
Peng et al. (2020)	Fuse and derive the optimal PET-CT image features	PET-CT radiomics	PET-CT dataset from the cancer imaging archive	51 multi-modality PET-CT scans derived from 51 patients with pathology-proven STSs	No
Qiang et al. (2020); Li et al. (2020, 2021)	Modeling task-based fMRI data	fMRI	Human Connectome Project Q3 tfMRI dataset	1,678,100 volumes of 865 patients with 7 behavioral tasks	No
Liu et al. (2019)	Medical image denoising	CT scans	-	10,775 cerebral perfusion CT images (512×512 gray-scale images) .	No
Huang et al. (2020)	Medical image reconstruction	MRI scans	Brain and Cardiac datasets	4480 cardiac real-valued MR images from 33 subjects; T1-weighted Brain MR of 35 fully-sampled subjects. (images of size 256×256 for both datasets)	No
Kügler et al. (2020)	Instrument pose estimation	X-ray	i3PosNet Dataset	10000 synthetic images and 540 real X-ray images of medical screws on a phantom head; captured with a Ziehm c-arm machine	No

Table 7.8: NAS details for healthcare supporting tasks papers

Paper	**Task**	**Network Searched**	**Parameters searched**	**Search strategy**	**Objective(s)**	**Search time**
Balaprakash et al. (2019)	Prediction	DNN	DNN components that take into account characteristics specific to cancer data	Reinforcement Learning	Accuracy	-
Lin et al. (2021)	Prediction	U-shaped CNN	Operations of 3 types of cells and connections	Gradient descent optimization	Dose error	-
Peng et al. (2020)	Feature extraction and fusion	3D CNN	Cell operations	Gradient descent optimization	Cross-entropy loss	6.16 GPU hours

Qiang et al. (2020)	Data modeling	DBN	Number of layers and number of neurons in each layer	Particle swarm optimization	Mean Squared Error	30 GPU hours
Li et al. (2020)	Data modeling	DSRAE	Number of layers and number of neurons in each layer	Evolutionary algorithm	Mean Squared Error	-
Li et al. (2021)	Data modeling	RNN	Number of layers and number of neurons in each layer	Gradient descent optimization	Mean Squared Error	2-6 GPU hours
Liu et al. (2019)	Image denoising	CNN	Number of layers, number of neurons in each layer, activation function and optimizers	Genetic Algorithm	Peak Signal-to-Noise Ratio	540 GPU hours
Huang et al. (2020)	Image reconstruction	FCN	Operations and connections in each cell	Gradient descent optimization	L2 loss	12 GPU hours
Kügler et al. (2020)	Pose estimation	AE	Layer operations and typology	Gradient descent optimization	Cross-Entropy and MSE	100 GPU hours

7.6 Global discussion and future research directions

This chapter discusses the different trends in the area of NAS in healthcare. It includes the advances, shortcomings, and possible contributions in this field.

In terms of healthcare tasks, the most addressed ones are medical image applications. This reflects the trend of the general research on NAS that is mostly applied to images. More specifically, it is shown that medical image segmentation (in 2D or 3D form) is the most prevalent here.

The most searched architectures are U-shaped fully convolutional networks. This type of architecture has many parameters and needs to be well-adjusted. Optimizing this type of architecture often depends on the data type as well as the task itself, which can get notably complex in some areas such as 3D segmentation.

Most search strategies use differentiable NAS for gradient descent optimization, which is often reported as less time-consuming than evolutionary approaches. A lot of contributions rely on already existing frameworks, such as ENAS (Pham et al., 2018) and ADANET (Cortes et al., 2017), and others propose their own search strategies. It is also shown that the data used for these contributions are mainly from public datasets such as the Automated Cardiac Diagnosis Challenge (ACDC) dataset, Autism Brain Imaging Data Exchange (ABIDE) dataset (Di Martino et al., 2014), CheXpert benchmark dataset (Irvin et al., 2019), etc.

In the papers presented in this review, some shortcomings are identified. Explainability and interpretability are both very important in healthcare but very few works include them in their research (Jiang et al., 2021) (He et al., 2021a).

There are also very few contributions for applications involving sequential data like data ex-

tractions in medical reports and other NLP tasks. Only some of them use this type of data as seen in (Lv et al., 2021) for ECG signals, and (Qiang et al., 2020; Li et al., 2020, 2021) for fMRI signals.

Finally, contributions to medical predictions using NAS are rare. This category of tasks usually involves tabular data or sequential data that is also lacking as a type of data generally used in NAS research.

In light of these observations, interesting directions in this research would include more applications related to non-image data. Medical report analysis, medical prognosis, and tasks related to public health are all important areas that could benefit from NAS. Moreover, employing interpretability or explainability when developing healthcare applications can help practitioners in their decision-making.

Another interesting direction is using federated learning with NAS. This technique enforces privacy by allowing model training without revealing data to unauthorized users (Zhu et al., 2021). This could give more access to private datasets and allow the widespread of healthcare applications.

General Conclusion

> "Second star to the right and straight on 'til morning."
>
> — J.M. Barrie, Peter Pan

In this thesis, the exploration of Neural Architecture Search (NAS) has brought several insights and contributions across different aspects of neural network design and application. The journey transited from understanding the complex mechanisms of NAS, simplifying and optimizing its processes through local search heuristics, addressing the multi-objective nature of neural network design, and finally to the application of these principles in real-world scenarios, particularly in healthcare. The contribution summary of this thesis is as follows:

Contribution Summary

Introducing the concept of neural architecture search

Initially, the thesis underscored the pivotal role of NAS in automating and optimizing the design of neural network architectures, highlighting its potential to democratize and enhance the development of deep learning models across various domains. We explored how NAS, despite being an effective solution for automating model design, has its share of challenges and complexity. In our analysis, we highlighted how new design methods could benefit from straightforward approaches, fighting the overly complex strategies prevalent in the literature.

Exploring simplicity while maintaining efficiency with LS-Weight and LS-PON

The thesis then focused on the search for simplicity while maintaining efficiency in the NAS process. We explored various approaches like Local Search (LS), LS-Weights, and LS-PON, each to optimize the design process by reducing the necessity for extensive human expertise and intervention, while also maintaining or enhancing efficiency and quality.

In the first method, called LS-Weights, we incorporated human knowledge in the search process to guide it to better solutions. We initiated this by analyzing state-of-the-art manually-designed architectures to extract insights and knowledge. After that, we added a mechanism to incorporate

these insights and see their effect on the speed and quality of the obtained architectures. Based on our tests on benchmarks, we achieved significant speedup (from 7% to 49%) with a simple modification of the neighborhood function while maintaining state-of-the-art results on benchmarks.

In our second method, LS-PON, our approach added an online learning module using machine learning to automate the knowledge extraction. This module automatically learns which architecture has more potential and guides the neighbor exploration. This alleviates the human intervention for knowledge extraction and gives more autonomy to the method. Upon testing this on multiple benchmarks, we confirmed a significant speedup compared to vanilla local search in most cases, achieving from 15% to 109% speedup on these datasets.

With these two approaches, we emphasized the significance of maintaining simplicity in search methods to ensure that they do not negate the benefits of NAS automation. We confirmed that even with approaches with no parameters tuning, we were still able to achieve state-of-the-art results in reduced time.

Tackling Multi-objective NAS

Furthermore, we presented the multi-objective approach to neural network architecture design. We highlighted the importance of balancing several competing factors, such as predictive performance, computational efficiency, and resource utilization, which are crucial for application in real-world scenarios. We presented different approaches to multi-objective optimization in NAS, classifying them and highlighting the benefits and challenges each of them offers. This enabled us to identify new paths for improvements that our contribution could focus on.

A new framework with dominance-based multi-objective local search

In light of the insights gained from our study of multi-objective NAS, we proposed a new approach using dominance-based multi-objective local search (DMLS). Our method addresses the challenges previously discovered in existing strategies. In this contribution, we presented a formulation of the NAS problem, as well as the design of each DMLS component adapted to solve it. Our method required minimal tuning to allow flexibility. We ran experiments on specialized benchmarks to optimize both accuracy and complexity. We proved our method's efficiency and superiority compared to other classical multi-objective NAS methods. Our approach, while requiring minimal tuning, leverages the strengths of LS and introduces the flexibility to accommodate multiple objectives, thereby, providing a promising solution to navigate the complexities of multi-objective NAS.

Transitioning to real-world applications

We transitioned from benchmark scenarios to real-world applications with a use case in healthcare. We highlighted the challenges and considerations essential for implementing NAS effectively in practical contexts. Our approach followed a meticulous protocol to select the best components at

each step of our implementation when using real data. We showcased using practical examples, how each of these components interferes with the results of the NAS framework. Starting from selecting the appropriate search space to fine-tuning the training parameters, we backed every step with empirical studies.

After setting these components, we also validated the effectiveness of LS-PON in real-world datasets, illustrating its potential to fit different modalities and tasks. Using this framework, we achieved results comparable with the state-of-the-art on two distinct healthcare tasks.

NAS in Healthcare: opportunities and challenges

Lastly, we explored the use of NAS in the healthcare domain. In this part, we presented a comprehensive review of over 40 contributions, highlighting the potential of NAS in this domain. Our work provided a structured starting point for those new to the field or considering leveraging NAS for healthcare applications. It highlights the potential of NAS in healthcare and underscores the current gaps and research opportunities.

Future Research

During this thesis, the exploration of neural architecture search has revealed a range of opportunities, challenges, and methodologies that combine the fields of deep learning, optimization, and practical applications. Based on the insights gained and the gaps identified during this research, several paths can be highlighted for future research.

- **Expanding to Other Data Types:** A noticeable gap exists in applying NAS to non-image data such as sequential, tabular, or text data. Future research could explore NAS frameworks specifically designed for these data types, especially for healthcare tasks like medical report analysis or patient prognosis based on health records.
- **Interpretability and Explainability:** There is a need to develop NAS methodologies that not only target performance metrics but also ensure the models are interpretable by domain experts. This can be achieved by designing search spaces with interpretable architectures or by developing explainability metrics to be optimized during the search process.
- **Expanding Multi-Objective Approaches:** There is an opportunity to explore and create more strategies for multi-objective NAS. These strategies can weigh and balance modern objectives, such as fairness and robustness, which are becoming increasingly important in practical applications.
- **NAS for Non-Traditional Architectures:** Modern architectures, like quantum neural networks and spiking neural networks, among others, can also benefit from NAS. Investigating NAS methods specifically designed for these innovative architectures can significantly expand their practicality.

- **Dynamic and Adaptive NAS:** Instead of a one-size-fits-all approach, future NAS methodologies could be dynamic and adaptive, adjusting the search based on the differences of the specific task and data at hand.

Published and Submitted Works

The following works have been either published and/or presented at conferences during this thesis in chronological order:

- Meyssa Zouambi, Clarisse Dhaenens, Julie Jacques. LS-Weights: Local Search with Weighted Neighborhood Sampling for Neural Architecture Search. NLDL2022, Jan 2022, Tromso, Norway.
- Meyssa Zouambi, Clarisse Dhaenens, Julie Jacques. Improving Local Search for Neural Architecture Search. 23ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, INSA Lyon, Feb 2022, Villeurbanne - Lyon, France.
- Meyssa Zouambi, Julie Jacques, Clarisse Dhaenens. Dominance-Based Local Search for Neural Architecture Search, DSO Workshop, IJCAI 2022, Vienna, Austria.
- Meyssa Zouambi, Julie Jacques, Clarisse Dhaenens. LS-PON: a Prediction-based Local Search for Neural Architecture Search. The 8th Annual Conference on machine Learning, Optimization and Data science LOD2022, Sep 2022, Siena, Italy. Page 108-122.
- Meyssa Zouambi, Clarisse Dhaenens, Julie Jacques. An Alternative Pareto-based Approach to Multi-objective Neural Architecture Search. IEEE 2023 Congress on Evolutionary Computation, Jul 2023, Chicago, United States.
- Meyssa Zouambi, Clarisse Dhaenens, Louis Rousselet, Julie Jacques. Neural Architecture Search for Healthcare: a Comprehensive Survey. Journal submitted to Artificial Intelligence Review.

Bibliography

Kaggle Ultrasound Nerve Segmentation, $http://fhtagn.net/prog/2016/08/19/kaggle-uns.html$, 2016.

Exascale Deep Learning and Simulation Enabled Precision Medicine for Cancer, $https://candle.cels.anl.gov/$, 2021.

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L.: Deep learning with differential privacy, in: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pp. 308–318, 2016.

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Hasan, M., Van Essen, B. C., Awwal, A. A., and Asari, V. K.: A state-of-the-art survey on deep learning theory and architectures, electronics, 8, 292, 2019.

Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al.: Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI, Information Fusion, 58, 82–115, 2020.

Bae, W., Lee, S., Lee, Y., Park, B., Chung, M., and Jung, K. H.: Resource Optimized Neural Architecture Search for 3D Medical Image Segmentation, Lecture Notes in Computer Science, 11765 LNCS, 228–236, 2019.

Baker, B., Gupta, O., Naik, N., and Raskar, R.: Designing neural network architectures using reinforcement learning, arXiv preprint arXiv:1611.02167, 2016.

Baker, B., Gupta, O., Raskar, R., and Naik, N.: Accelerating neural architecture search using performance prediction, arXiv:1705.10823, 2017.

Balaprakash, P., Egele, R., Salim, M., Wild, S., Vishwanath, V., Xia, F., Brettin, T., and Stevens, R.: Scalable reinforcement-learning-based neural architecture search for cancer deep learning research, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–33, 2019.

Baldeon Calisto, M. and Lai-Yuen, S. K.: AdaEn-Net: An ensemble of adaptive 2D–3D Fully Convolutional Networks for medical image segmentation, Neural Networks, 126, 76–94, 2020.

Baldeon Calisto, M. G. and Lai-Yuen, S. K.: EMONAS: efficient multiobjective neural architecture search framework for 3D medical image segmentation, 1159607, 3, 2021.

Bernal, J., Tajkbaksh, N., Sanchez, F. J., Matuszewski, B. J., Chen, H., Yu, L., Angermann, Q., Romain, O., Rustad, B., Balasingham, I., et al.: Comparative validation of polyp detection methods in video colonoscopy: results from the MICCAI 2015 endoscopic vision challenge, IEEE transactions on medical imaging, 36, 1231–1249, 2017.

Bottou, L. et al.: Stochastic gradient learning in neural networks, Proceedings of Neuro-Nımes, 91, 12, 1991.

Cai, H., Zhu, L., and Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware, arXiv preprint arXiv:1812.00332, 2018.

Chen, C., Li, O., Tao, C., Barnett, A. J., Su, J., and Rudin, C.: This looks like that: deep learning for interpretable image recognition, arXiv preprint arXiv:1806.10574, 2018.

Chitty-Venkata, K. T., Emani, M., Vishwanath, V., and Somani, A. K.: Neural Architecture Search Benchmarks: Insights and Survey, IEEE Access, 11, 25 217–25 236, 2023.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation, arXiv preprint arXiv:1406.1078, 2014.

Cortes, C., Gonzalvo, X., Kuznetsov, V., Mohri, M., and Yang, S.: Adanet: Adaptive structural learning of artificial neural networks, in: International conference on machine learning, pp. 874–883, PMLR, 2017.

Dai, H., Ge, F., Li, Q., Zhang, W., and Liu, T.: Optimize CNN Model for FMRI Signal Classification Via Adanet-Based Neural Architecture Search, Proceedings - International Symposium on Biomedical Imaging, 2020-April, 1399–1403, 2020.

Deb, K.: Multi-objective optimisation using evolutionary algorithms: an introduction, in: Multi-objective evolutionary optimisation for product design and manufacturing, pp. 3–34, Springer, 2011.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE transactions on evolutionary computation, 6, 182–197, 2002.

Den Ottelander, T., Dushatskiy, A., Virgolin, M., and Bosman, P. A.: Local search is a remarkably strong baseline for neural architecture search, in: Evolutionary Multi-Criterion Optimization: 11th International Conference, EMO 2021, Shenzhen, China, March 28–31, 2021, Proceedings 11, pp. 465–479, Springer, 2021.

Deng, J., Berg, A., Satheesh, S., Su, H., Khosla, A., and Fei-Fei, L.: Imagenet large scale visual recognition competition 2012 (ILSVRC2012), See net. org/challenges/LSVRC, 41, 2012.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805, 2018.

Dhaenens, C., Lemesre, J., and Talbi, E.-G.: K-PPM: A new exact method to solve multi-objective combinatorial optimization problems, European Journal of Operational Research, 200, 45–53, 2010.

Di Martino, A., Yan, C.-G., Li, Q., Denio, E., Castellanos, F. X., Alaerts, K., Anderson, J. S., Assaf, M., Bookheimer, S. Y., Dapretto, M., et al.: The autism brain imaging data exchange: towards a large-scale evaluation of the intrinsic brain architecture in autism, Molecular psychiatry, 19, 659–667, 2014.

Dong, H.-W., Hsiao, W.-Y., Yang, L.-C., and Yang, Y.-H.: Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.

Dong, N., Xu, M., Liang, X., Jiang, Y., Dai, W., and Xing, E.: Neural Architecture Search for Adversarial Medical Image Segmentation, vol. 2, Springer International Publishing, 2019.

Dong, X. and Yang, Y.: Nas-bench-201: Extending the scope of reproducible neural architecture search, arXiv preprint arXiv:2001.00326, 2020.

Dong, X., Liu, L., Musial, K., and Gabrys, B.: Nats-bench: Benchmarking nas algorithms for architecture topology and size, IEEE transactions on pattern analysis and machine intelligence, 44, 3634–3646, 2021.

Elgammal, A., Liu, B., Elhoseiny, M., and Mazzone, M.: Can: Creative adversarial networks, generating" art" by learning about styles and deviating from style norms, arXiv preprint arXiv:1706.07068, 2017.

Elsken, T., Metzen, J.-H., and Hutter, F.: Simple and efficient architecture search for convolutional neural networks, arXiv preprint arXiv:1711.04528, 2017.

Elsken, T., Metzen, J. H., and Hutter, F.: Efficient multi-objective neural architecture search via lamarckian evolution, arXiv preprint arXiv:1804.09081, 2018.

Elsken, T., Metzen, J. H., Hutter, F., et al.: Neural architecture search: A survey., J. Mach. Learn. Res., 20, 1–21, 2019.

Fan, Z., Wei, J., Zhu, G., Mo, J., and Li, W.: Evolutionary neural architecture search for retinal vessel segmentation, arXiv, 2020.

Farsiu, S., Chiu, S. J., O'Connell, R. V., Folgar, F. A., Yuan, E., Izatt, J. A., Toth, C. A., Group, A.-R. E. D. S. . A. S. D. O. C. T. S., et al.: Quantitative classification of eyes with and without intermediate age-related macular degeneration using optical coherence tomography, Ophthalmology, 121, 162–172, 2014.

Fedorov, I., Adams, R. P., Mattina, M., and Whatmough, P.: Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers, Advances in Neural Information Processing Systems, 32, 2019.

Freiburg-Hannover: literature on neural architecture search, $https://www.automl.org/automl/literature-on-neural-architecture-search/$, 2021a.

Freiburg-Hannover: Medical Segmentation Decathlon (MSD) challenge tasks, $http://medicaldecathlon.com/$, 2021b.

Gessert, N. and Schlaefer, A.: Efficient neural architecture search on low-dimensional data for OCT Image Segmentation, arXiv, pp. 1–5, 2019a.

Gessert, N. and Schlaefer, A.: Efficient neural architecture search on low-dimensional data for OCT image segmentation, arXiv preprint arXiv:1905.02590, 2019b.

Gheshlaghi, S. H., Dehzangi, O., Dabouei, A., Amireskandari, A., Rezai, A., and Nasrabadi, N. M.: Efficient Oct Image Segmentation Using Neural Architecture Search, Proceedings - International Conference on Image Processing, ICIP, 2020-Octob, 428–432, 2020.

Gondara, L.: Medical image denoising using convolutional denoising autoencoders, in: 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), pp. 241–246, IEEE, 2016.

Goodfellow, I., Bengio, Y., and Courville, A.: Deep learning, MIT press, 2016.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y.: Generative adversarial networks, arXiv preprint arXiv:1406.2661, 2014.

Guo, D., Jin, D., Zhu, Z., Ho, T. Y., Harrison, A. P., Chao, C. H., Xiao, J., and Lu, L.: Organ at risk segmentation for head and neck cancer using stratified learning and neural architecture search, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 4222–4231, 2020.

Haimes, Y.: On a bicriterion formulation of the problems of integrated system identification and system optimization, IEEE transactions on systems, man, and cybernetics, pp. 296–297, 1971.

He, K., Zhang, X., Ren, S., and Sun, J.: Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.

He, X., Wang, S., Shi, S., Chu, X., Tang, J., Liu, X., Yan, C., Zhang, J., and Ding, G.: Benchmarking deep learning models and automated model design for COVID-19 detection with chest CT scans, medRxiv, 2020.

He, X., Wang, S., Ying, G., Zhang, J., and Chu, X.: Efficient Multi-objective Evolutionary 3D Neural Architecture Search for COVID-19 Detection with Chest CT Scans, arXiv preprint arXiv:2101.10667, 2021a.

He, X., Wang, Y., Wang, X., Huang, W., Zhao, S., and Chen, X.: Simple-Encoded evolving convolutional neural network and its application to skin disease image classification, Swarm and Evolutionary Computation, 67, 100 955, 2021b.

He, X., Zhao, K., and Chu, X.: AutoML: A survey of the state-of-the-art, Knowledge-Based Systems, 212, 106 622, 2021c.

He, Y., Yang, D., Roth, H., Zhao, C., and Xu, D.: DiNTS: Differentiable Neural Network Topology Search for 3D Medical Image Segmentation, arXiv preprint arXiv:2103.15954, 2021d.

Hinton, G., Vinyals, O., and Dean, J.: Distilling the knowledge in a neural network, arXiv preprint arXiv:1503.02531, 2015.

Hinton, G. E., Osindero, S., and Teh, Y.-W.: A fast learning algorithm for deep belief nets, Neural computation, 18, 1527–1554, 2006.

Hochreiter, S. and Schmidhuber, J.: Long short-term memory, Neural computation, 9, 1735–1780, 1997.

Hsu, C.-H., Chang, S.-H., Liang, J.-H., Chou, H.-P., Liu, C.-H., Chang, S.-C., Pan, J.-Y., Chen, Y.-T., Wei, W., and Juan, D.-C.: Monas: Multi-objective neural architecture search using reinforcement learning, arXiv preprint arXiv:1806.10332, 2018.

Hu, L., Liu, Q., Zhang, J., Jiang, F., Liu, Y., and Zhang, S.: A-DARTS: attention-guided differentiable architecture search for lung nodule classification, Journal of Electronic Imaging, 30, 013 012, 2021.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q.: Densely connected convolutional networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700–4708, 2017.

Huang, Q., Xian, Y., Wu, P., Yi, J., Qu, H., Metaxas, D., et al.: Enhanced MRI Reconstruction Network Using Neural Architecture Search, in: International Workshop on Machine Learning in Medical Imaging, pp. 634–643, Springer, 2020.

Hutter, F., Hoos, H. H., and Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration, in: International conference on learning and intelligent optimization, pp. 507–523, Springer, 2011.

Irvin, J., Rajpurkar, P., Ko, M., Yu, Y., Ciurea-Ilcus, S., Chute, C., Marklund, H., Haghgoo, B., Ball, R., Shpanskaya, K., et al.: Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 590–597, 2019.

Jacques, J., Taillard, J., Delerue, D., Dhaenens, C., and Jourdan, L.: Conception of a dominance-based multi-objective local search in the context of classification rule mining in large and imbalanced data sets, Applied Soft Computing, 34, 705–720, 2015.

Ji, Y., Zhang, R., Li, Z., Ren, J., Zhang, S., and Luo, P.: Uxnet: Searching multi-level feature aggregation for 3d medical image segmentation, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 12261 LNCS, 346–356, 2020.

Jiang, C., Wang, S., Xu, H., Liang, X., and Xiao, N.: ElixirNet: Relation-aware network architecture adaptation for medical lesion detection, arXiv, 2020.

Jiang, H., Shen, F., Gao, F., and Han, W.: Learning efficient, explainable and discriminative representations for pulmonary nodules classification, Pattern Recognition, 113, 2021.

Jiang, Y., Krishnan, D., Mobahi, H., and Bengio, S.: Predicting the generalization gap in deep networks with margin distributions, arXiv preprint arXiv:1810.00113, 2018.

Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., and Xing, E. P.: Neural architecture search with bayesian optimisation and optimal transport, Advances in neural information processing systems, 31, 2018.

Kavur, A. E., Gezer, N. S., Barış, M., Aslan, S., Conze, P.-H., Groza, V., Pham, D. D., Chatterjee, S., Ernst, P., Özkan, S., et al.: CHAOS challenge-combined (CT-MR) healthy abdominal organ segmentation, Medical Image Analysis, 69, 101 950, 2021.

Kim, S., Kim, I., Lim, S., Baek, W., Kim, C., Cho, H., Yoon, B., and Kim, T.: Scalable neural architecture search for 3d medical image segmentation, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 220–228, Springer, 2019.

Kingma, D. P. and Ba, J.: Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014.

Kingma, D. P. and Welling, M.: Stochastic gradient VB and the variational auto-encoder, in: Second International Conference on Learning Representations, ICLR, vol. 19, 2014.

Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F.: Learning curve prediction with Bayesian neural networks, in: International Conference on Learning Representations, 2017.

Klyuchnikov, N., Trofimov, I., Artemova, E., Salnikov, M., Fedorov, M., Filippov, A., and Burnaev, E.: Nas-bench-nlp: neural architecture search benchmark for natural language processing, IEEE Access, 10, 45 736–45 747, 2022.

Knowles, J. and Corne, D.: The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation, in: Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), vol. 1, pp. 98–105, IEEE, 1999.

Kramer, M. A.: Nonlinear principal component analysis using autoassociative neural networks, AIChE journal, 37, 233–243, 1991.

Kügler, D., Stefanov, A., and Mukhopadhyay, A.: i3PosNet: Instrument pose estimation from X-ray, arXiv preprint arXiv:1802.09575, 2018.

Kügler, D., Uecker, M., Kuijper, A., and Mukhopadhyay, A.: AutoSNAP: Automatically Learning Neural Architectures for Instrument Pose Estimation, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 12263 LNCS, 375–384, 2020.

Kwasigroch, A., Grochowski, M., and Mikolajczyk, M.: Deep neural network architecture search using network morphism, in: International Conference on Methods and Models in Automation and Robotics, pp. 30–35, IEEE, 2019.

Kwasigroch, A., Grochowski, M., and Mikolajczyk, A.: Neural architecture search for skin lesion classification, IEEE Access, 8, 9061–9071, 2020.

LeCun, Y., Bengio, Y., and Hinton, G.: Deep learning, nature, 521, 436–444, 2015.

Lee, S., Kim, J., Kang, H., Kang, D.-Y., and Park, J.: Genetic Algorithm Based Deep Learning Neural Network Structure and Hyperparameter Optimization, Applied Sciences, 11, 744, 2021.

Lemesre, J., Dhaenens, C., and Talbi, E.-G.: Parallel partitioning method (PPM): A new exact method to solve bi-objective problems, Computers & operations research, 34, 2450–2462, 2007.

Li, L. and Talwalkar, A.: Random search and reproducibility for neural architecture search, in: Uncertainty in artificial intelligence, pp. 367–377, PMLR, 2020.

Li, Q., Zhang, W., Lv, J., Wu, X., and Liu, T.: Neural Architecture Search for Optimization of Spatial-Temporal Brain Network Decomposition, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 377–386, Springer, 2020.

Li, Q., Wu, X., and Liu, T.: Differentiable neural architecture search for optimal spatial/temporal brain function network decomposition, Medical Image Analysis, 69, 101 974, 2021.

Liefooghe, A., Humeau, J., Mesmoudi, S., Jourdan, L., and Talbi, E.-G.: On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems, Journal of Heuristics, 18, 317–352, 2012.

Lim, B., Son, S., Kim, H., Nah, S., and Mu Lee, K.: Enhanced deep residual networks for single image super-resolution, in: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp. 136–144, 2017.

Lin, J.-H. and Haug, P. J.: Data preparation framework for preprocessing clinical data in data mining, in: AMIA annual symposium proceedings, vol. 2006, p. 489, American Medical Informatics Association, 2006.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P.: Focal loss for dense object detection, in: Proceedings of the IEEE international conference on computer vision, pp. 2980–2988, 2017.

Lin, Y., Liu, Y., Liu, J., Liu, G., Ma, K., and Zheng, Y.: LE-NAS: Learning-based Ensenble with NAS for Dose Prediction, arXiv preprint arXiv:2106.06733, 2021.

Litjens, G., Toth, R., van de Ven, W., Hoeks, C., Kerkstra, S., van Ginneken, B., Vincent, G., Guillard, G., Birbeck, N., Zhang, J., et al.: Evaluation of prostate segmentation algorithms for MRI: the PROMISE12 challenge, Medical image analysis, 18, 359–373, 2014.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K.: Progressive neural architecture search, in: Proceedings of the European conference on computer vision (ECCV), pp. 19–34, 2018a.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K.: Hierarchical representations for efficient architecture search, arXiv preprint arXiv:1711.00436, 2017a.

Liu, H., Simonyan, K., and Yang, Y.: Darts: Differentiable architecture search, arXiv preprint arXiv:1806.09055, 2018b.

Liu, L., Wen, Z., Liu, S., Zhou, H.-Y., Zhu, H., Xie, W., Shen, L., Ma, K., and Zheng, Y.: MixSearch: Searching for Domain Generalized Medical Image Segmentation Architectures, XX, 1–10, 2021.

Liu, P., El Basha, M. D., Li, Y., Xiao, Y., Sanelli, P. C., and Fang, R.: Deep Evolutionary Networks with Expedited Genetic Algorithms for Medical Image Denoising, Medical Image Analysis, 54, 306–315, 2019.

Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., and Song, L.: Sphereface: Deep hypersphere embedding for face recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 212–220, 2017b.

Long, J., Shelhamer, E., and Darrell, T.: Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3431–3440, 2015.

Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., and Banzhaf, W.: Nsga-net: neural architecture search using multi-objective genetic algorithm, in: Proceedings of the genetic and evolutionary computation conference, pp. 419–427, 2019.

Lu, Z., Whalen, I., Dhebar, Y., Deb, K., Goodman, E. D., Banzhaf, W., and Boddeti, V. N.: Multiobjective evolutionary design of deep convolutional neural networks for image classification, IEEE Transactions on Evolutionary Computation, 25, 277–291, 2020.

Luo, R., Tan, X., Wang, R., Qin, T., Chen, E., and Liu, T.-Y.: Accuracy Prediction with Non-neural Model for Neural Architecture Search, arXiv:2007.04785, 2020.

Lv, J., Ye, Q., Sun, Y., Zhao, J., and Lv, J.: Heart-Darts: Classification of Heartbeats Using Differentiable Architecture Search, arXiv preprint arXiv:2105.00693, 2021.

Marler, R. T. and Arora, J. S.: Survey of multi-objective optimization methods for engineering, Structural and multidisciplinary optimization, 26, 369–395, 2004.

Mehrotra, A., Ramos, A. G. C., Bhattacharya, S., Dudziak, Ł., Vipperla, R., Chau, T., Abdelfattah, M. S., Ishtiaq, S., and Lane, N. D.: Nas-bench-asr: Reproducible neural architecture search for speech recognition, in: International Conference on Learning Representations, 2021.

Miahi, E., Mirroshandel, S. A., and Nasr, A.: Genetic Neural Architecture Search for automatic assessment of human sperm images, arXiv preprint arXiv:1909.09432, 2019.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al.: Mixed precision training, arXiv preprint arXiv:1710.03740, 2017.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al.: Evolving deep neural networks, in: Artificial intelligence in the age of neural networks and brain computing, pp. 293–312, Elsevier, 2019.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J.: Pruning convolutional neural networks for resource efficient inference, arXiv preprint arXiv:1611.06440, 2016.

Mortazi, A. and Bagci, U.: Automatically designing CNN architectures for medical image segmentation, in: International Workshop on Machine Learning in Medical Imaging, pp. 98–106, Springer, 2018.

Odema, M., Rashid, N., and Al Faruque, M. A.: Energy-aware design methodology for myocardial infarction detection on low-power wearable devices, in: 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 621–626, IEEE, 2021.

Paquete, L., Chiarandini, M., and Stützle, T.: Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study, in: Metaheuristics for multiobjective optimisation, pp. 177–199, Springer, 2004.

Peng, Y., Bi, L., Fulham, M., Feng, D., and Kim, J.: Multi-modality Information Fusion for Radiomics-Based Neural Architecture Search, Lecture Notes in Computer Science (including

subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 12267 LNCS, 763–771, 2020.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J.: Efficient neural architecture search via parameters sharing, in: International Conference on Machine Learning, pp. 4095–4104, PMLR, 2018.

Qiang, N., Dong, Q., Zhang, W., Ge, B., Ge, F., Liang, H., Sun, Y., Gao, J., and Liu, T.: Modeling task-based fMRI data via deep belief network with neural architecture search, Computerized Medical Imaging and Graphics, 83, 101 747, 2020.

Qin, Y., Zhang, Z., Wang, X., Zhang, Z., and Zhu, W.: NAS-Bench-Graph: Benchmarking Graph Neural Architecture Search, arXiv preprint arXiv:2206.09166, 2022.

Qiu, Z., Yao, T., and Mei, T.: Learning spatio-temporal representation with pseudo-3d residual networks, in: proceedings of the IEEE International Conference on Computer Vision, pp. 5533–5541, 2017.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving language understanding by generative pre-training, mikecaptain, 2018.

Radiuk, P. and Kutucu, H.: Heuristic architecture search using network morphism for Chest X-Ray classification, CEUR Workshop Proceedings, 2623, 107–121, 2020.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J.: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, 2020.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A.: Large-scale evolution of image classifiers, in: International Conference on Machine Learning, pp. 2902–2911, PMLR, 2017.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V.: Regularized evolution for image classifier architecture search, in: Proceedings of the aaai conference on artificial intelligence, vol. 33, pp. 4780–4789, 2019.

Ronneberger, O., Fischer, P., and Brox, T.: U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical image computing and computer-assisted intervention, pp. 234–241, Springer, 2015.

Ruder, S.: An overview of gradient descent optimization algorithms, arXiv preprint arXiv:1609.04747, 2016.

Salakhutdinov, R., Mnih, A., and Hinton, G.: Restricted Boltzmann machines for collaborative filtering, in: Proceedings of the 24th international conference on Machine learning, pp. 791–798, 2007.

Saxe, A. M., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. Y.: On random weights and unsupervised feature learning., in: Icml, vol. 2, p. 6, 2011.

Schwartz, O.: You thought fake news was bad? Deep fakes are where truth goes to die, The Guardian, 12, 2018.

Shin, H.-C., Tenenholtz, N. A., Rogers, J. K., Schwarz, C. G., Senjem, M. L., Gunter, J. L., Andriole, K. P., and Michalski, M.: Medical image synthesis for data augmentation and anonymization using generative adversarial networks, in: International workshop on simulation and synthesis in medical imaging, pp. 1–11, Springer, 2018.

Siems, J., Zimmer, L., Zela, A., Lukasik, J., Keuper, M., and Hutter, F.: Nas-bench-301 and the case for surrogate benchmarks for neural architecture search, arXiv preprint arXiv:2008.09777, 2020.

Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., and Marculescu, D.: Single-path nas: Designing hardware-efficient convnets in less than 4 hours, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 481–497, Springer, 2019.

Stanley, K. O. and Miikkulainen, R.: Evolving neural networks through augmenting topologies, Evolutionary computation, 10, 99–127, 2002.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V.: Mnasnet: Platform-aware neural architecture search for mobile, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 2820–2828, 2019.

Tenfelde-Podehl, D.: A Recursive Algorithm for Multi-Objective Combinatorial Optimization Problems with q-Criteria, unpublished, 2003.

Tenorio, M. and Lee, W.-T.: Self organizing neural networks for the identification problem, Advances in Neural Information Processing Systems, 1, 1988.

Tieleman, T., Hinton, G., et al.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural networks for machine learning, 4, 26–31, 2012.

Ulungu, E. L. and Teghem, J.: The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems, Foundations of computing and decision sciences, 20, 149–165, 1995.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I.: Attention is all you need, Advances in neural information processing systems, 30, 2017.

Veniat, T. and Denoyer, L.: Learning time/memory-efficient deep architectures with budgeted super networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3492–3500, 2018.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A.: Extracting and composing robust features with denoising autoencoders, in: Proceedings of the 25th international conference on Machine learning, pp. 1096–1103, 2008.

Wang, B., Sun, Y., Xue, B., and Zhang, M.: Evolving deep neural networks by multi-objective particle swarm optimization for image classification, in: Proceedings of the genetic and evolutionary computation conference, pp. 490–498, 2019.

Wang, F.: Neural Architecture Search for Gliomas Segmentation on Multimodal Magnetic Resonance Imaging, arXiv, 2020.

Wang, H., Li, R., Chen, X., Duan, B., Xiong, L., Yang, X., Fan, H., and Ni, D.: Remote Intelligent Assisted Diagnosis System for Hepatic Echinococcosis, in: Medical Ultrasound, and Preterm, Perinatal and Paediatric Image Analysis, pp. 3–12, Springer, 2020.

Wei, C., Niu, C., Tang, Y., Wang, Y., Hu, H., and Liang, J.: Npenas: Neural predictor guided evolution for neural architecture search, arXiv:2003.12857, 2020.

Wei, T., Wang, C., Rui, Y., and Chen, C. W.: Network morphism, in: International Conference on Machine Learning, pp. 564–572, PMLR, 2016.

Weng, Y., Zhou, T., Li, Y., and Qiu, X.: NAS-Unet: Neural architecture search for medical image segmentation, IEEE Access, 7, 44 247–44 257, 2019.

White, C., Nolen, S., and Savani, Y.: Exploring the Loss Landscape in Neural Architecture Search, arXiv:2005.02960, 2020.

White, C., Neiswanger, W., and Savani, Y.: Bananas: Bayesian optimization with neural architectures for neural architecture search, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 10 293–10 301, 2021.

Williams, R. J. and Zipser, D.: A learning algorithm for continually running fully recurrent neural networks, Neural computation, 1, 270–280, 1989.

Wistuba, M., Rawat, A., and Pedapati, T.: A survey on neural architecture search, arXiv preprint arXiv:1905.01392, 2019.

Woo, S., Park, J., Lee, J.-Y., and Kweon, I. S.: Cbam: Convolutional block attention module, in: Proceedings of the European conference on computer vision (ECCV), pp. 3–19, 2018.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 10 734–10 742, 2019.

Wu, J., Dai, X., Chen, D., Chen, Y., Liu, M., Yu, Y., Wang, Z., Liu, Z., Chen, M., and Yuan, L.: Weak NAS Predictors Are All You Need, arXiv:2102.10490, 2021.

Xie, L. and Yuille, A.: Genetic cnn, in: Proceedings of the IEEE international conference on computer vision, pp. 1379–1388, 2017.

Xie, S., Zheng, H., Liu, C., and Lin, L.: SNAS: stochastic neural architecture search, arXiv preprint arXiv:1812.09926, 2018.

Xu, Y., Xie, L., Zhang, X., Chen, X., Shi, B., Tian, Q., and Xiong, H.: Latency-aware differentiable neural architecture search, arXiv preprint arXiv:2001.06392, 2020a.

Xu, Z., Zuo, S., Lam, E. Y., Lee, B., and Chen, N.: AutoSegNet: An Automated Neural Network for Image Segmentation, IEEE Access, 8, 92 452–92 461, 2020b.

Yan, S., White, C., Savani, Y., and Hutter, F.: Nas-bench-x11 and the power of learning curves, Advances in Neural Information Processing Systems, 34, 22 534–22 549, 2021.

Yan, X., Jiang, W., Shi, Y., and Zhuo, C.: Ms-nas: Multi-scale neural architecture search for medical image segmentation., Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 12261 LNCS, 388–397, 2020.

Yang, J., Shi, R., Wei, D., Liu, Z., Zhao, L., Ke, B., Pfister, H., and Ni, B.: MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification, Scientific Data, 10, 41, 2023.

Yang, Q., Liu, Y., Chen, T., and Tong, Y.: Federated machine learning: Concept and applications, ACM Transactions on Intelligent Systems and Technology (TIST), 10, 1–19, 2019.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F.: Nas-bench-101: Towards reproducible neural architecture search, in: International Conference on Machine Learning, pp. 7105–7114, PMLR, 2019.

Yu, K., Sciuto, C., Jaggi, M., Musat, C., and Salzmann, M.: Evaluating the search phase of neural architecture search, arXiv preprint arXiv:1902.08142, 2019.

Yu, Q., Yang, D., Roth, H., Bai, Y., Zhang, Y., Yuille, A. L., and Xu, D.: C2FNAs: Coarse-to-fine neural architecture search for 3D medical image segmentation, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 4125–4134, 2020.

Zela, A., Klein, A., Falkner, S., and Hutter, F.: Towards automated deep learning: Efficient joint neural architecture and hyperparameter search, arXiv preprint arXiv:1807.06906, 2018.

Zeng, D., Jiang, W., Wang, T., Xu, X., Yuan, H., Huang, M., Zhuang, J., Hu, J., and Shi, Y.: Towards Cardiac Intervention Assistance: Hardware-aware Neural Architecture Exploration for Real-Time 3D Cardiac Cine MRI Segmentation, IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD, 2020-Novem, 2020.

Zhang, N., Wang, J., Yang, J., Qu, X., and Xiao, J.: Multi-objective cuckoo algorithm for mobile devices network architecture search, in: Artificial Neural Networks and Machine Learning–ICANN 2020: 29th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 15–18, 2020, Proceedings, Part I 29, pp. 312–324, Springer, 2020.

Zhang, Q. and Li, H.: MOEA/D: A multiobjective evolutionary algorithm based on decomposition, IEEE Transactions on evolutionary computation, 11, 712–731, 2007.

Zhu, H., Zhang, H., and Jin, Y.: From federated learning to federated neural architecture search: a survey, Complex & Intelligent Systems, 7, 639–657, 2021.

Zhu, Z., Liu, C., Yang, D., Yuille, A., and Xu, D.: V-NAS: Neural Architecture Search for Volumetric Medical Image Segmentation, Proceedings - 2019 International Conference on 3D Vision, 3DV 2019, pp. 240–248, 2019.

Zitzler, E., Laumanns, M., and Thiele, L.: SPEA2: Improving the strength Pareto evolutionary algorithm, TIK report, 103, 2001.

Zoph, B. and Le, Q. V.: Neural architecture search with reinforcement learning, arXiv preprint arXiv:1611.01578, 2016.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V.: Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8697–8710, 2018.