Exploring deep neural network differentiable architecture design

Alexandre Heuillet

université PARIS-SACLAY

Exploring Deep Neural Network Differentiable Architecture Design

Optimisation Automatique des Architectures de Réseaux de Neurones Profonds via un Objectif Différentiable

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°580 Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Informatique
Graduate School : Informatique et Sciences du Numérique. Référent : Université d'Évry Val d'Essonne

Thèse préparée dans l'unité de recherche **IBISC (Université Paris-Saclay, Univ Evry)**, sous la direction de **Hedi TABIA**, professeur et la co-direction de **Hichem ARIOUI**, Maître de conférences HDR.

Thèse soutenue à Paris-Saclay, le 4 décembre 2023, par

Alexandre HEUILLET

Composition du jury

Membres du jury avec voix délibérative

Blaise HANCZAR Professeur des universités, Université Paris-Saclay	Président
Florence D'ALECHE-BUC Professeure des universités, Télécom Paris	Rapporteure & Examinatrice
Raul SANTOS-RODRIGUEZ Professeur, University of Bristol	Rapporteur & Examinateur
David PICARD Directeur de recherche, Ecole des Ponts ParisTech	Examinateur

THESE DE DOCTORAT
NNT : 2023UPASG069

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Optimisation Automatique des Architectures de Réseaux de Neurones Profonds via un Objectif Différentiable

Mots clés: Apprentissage automatique, Apprentissage profond, Recherche d'architecture neuronale, Réseaux convolutifs, Intelligence Artificielle

Résumé: L'intelligence artificielle (IA) a gagné en popularité ces dernières années, principalement en raison de ses applications réussies dans divers domaines tels que l'analyse de données textuelles, la vision par ordinateur et le traitement audio. La résurgence des techniques d'apprentissage profond a joué un rôle central dans ce succès. L'article révolutionnaire de Krizhevsky et al., AlexNet, a réduit l'écart entre les performances humaines et celles des machines dans les tâches de classification d'images. Des articles ultérieurs tels que Xception et ResNet ont encore renforcé l'apprentissage profond en tant que technique de pointe, ouvrant de nouveaux horizons pour la communauté de l'IA. Le succès de l'apprentissage profond réside dans son architecture, conçue manuellement avec des connaissances d'experts et une validation empirique. Cependant, ces architectures n'ont pas la certitude d'être la solution optimale. Pour résoudre ce problème, des articles récents ont introduit le concept de Recherche d'Architecture Neuronale (*NAS*), permettant l'automatisation de la conception des architectures profondes. Cependant, la majorités des approches initiales se sont concentrées sur de grandes architectures avec des objectifs spécifiques (par exemple, l'apprentissage supervisé) et ont utilisé des techniques d'optimisation coûteuses en calcul telles que l'apprentissage par renforcement et les algorithmes génétiques. Dans cette thèse, nous approfondissons cette idée en explorant la conception automatique d'architectures profondes, avec une emphase particulière sur les méthodes *NAS* différentiables (*DNAS*), qui représentent la tendance actuelle en raison de leur efficacité computationnelle. Bien que notre principal objectif soit les réseaux convolutifs (*CNNs*), nous explorons également les Vision Transformers (ViTs) dans le but de concevoir des architectures rentables adaptées aux applications en temps réel.

Cette thèse est basée sur les publications suivantes:

- Alexandre Heuillet, Hedi Tabia, Hichem Arioui, and Kamal Youcef-Toumi. "D-DARTS: Distributed Differentiable Architecture Search". *Pattern Recognition Letters*, 2023.
- Alexandre Heuillet, Ahmad Nasser, Hichem Arioui, and Hedi Tabia. "Efficient Automation of Neural Network Design: A Survey on Differentiable Neural Architecture Search". *arXiv preprint arXiv:2304.05405*, 2023. Soumis à ACM Computing Surveys.
- Alexandre Heuillet, Hedi Tabia, and Hichem Arioui. "NASiam: Efficient Representation Learning using Neural Architecture Search for Siamese Networks". *INNS DLIA 2023, IJCNN 2023, Procedia Computer Science, Elsevier*.
- Alexandre Heuillet, Hedi Tabia, and Hichem Arioui. "Automated Siamese Network Design for Image Similarity Computation". *International Conference on Content-Based Multimedia Indexing*, 2023.
- Alexandre Heuillet, Haozhe Sun, Isabelle Guyon, Felix Mohr, and Hedi Tabia. "DARIO: Differentiable vision transformer pruning with low-cost proxies". Soumis à *IEEE Journal of Selected Topics in Signal Processing*, 2023.

Title: Exploring Deep Neural Network Differentiable Architecture Design
Keywords: Machine Learning, Deep Learning, Neural Architecture Search, Convolutional Neural Networks, Artificial Intelligence

Abstract: Artificial Intelligence (AI) has gained significant popularity in recent years, primarily due to its successful applications in various domains, including textual data analysis, computer vision, and audio processing. The resurgence of deep learning techniques has played a central role in this success. The groundbreaking paper by Krizhevsky et al., AlexNet, narrowed the gap between human and machine performance in image classification tasks. Subsequent papers such as Xception and ResNet have further solidified deep learning as a leading technique, opening new horizons for the AI community. The success of deep learning lies in its architecture, which is manually designed with expert knowledge and empirical validation. However, these architectures lack the certainty of an optimal solution. To address this issue, recent papers introduced the concept of Neural Architecture Search (NAS), enabling the learning of deep architectures. However, most initial approaches focused on large architectures with specific targets (e.g., supervised learning) and relied on computationally expensive optimization techniques such as reinforcement learning and evolutionary algorithms. In this thesis, we further investigate this idea by exploring automatic deep architecture design, with a particular emphasis on differentiable NAS (DNAS), which represents the current trend in NAS due to its computational efficiency. While our primary focus is on Convolutional Neural Networks (CNNs), we also explore Vision Transformers (ViTs) with the goal of designing cost-effective architectures suitable for real-time applications.

This thesis is based on the following publications:

- Alexandre Heuillet, Hedi Tabia, Hichem Arioui, and Kamal Youcef-Toumi. "D-DARTS: Distributed Differentiable Architecture Search". *Pattern Recognition Letters*, 2023.
- Alexandre Heuillet, Ahmad Nasser, Hichem Arioui, and Hedi Tabia. "Efficient Automation of Neural Network Design: A Survey on Differentiable Neural Architecture Search". *arXiv preprint arXiv:2304.05405*, 2023. Submitted to ACM Computing Surveys.
- Alexandre Heuillet, Hedi Tabia, and Hichem Arioui. NASiam: "Efficient Representation Learning using Neural Architecture Search for Siamese Networks". *INNS DLIA 2023, IJCNN 2023, Procedia Computer Science, Elsevier*.
- Alexandre Heuillet, Hedi Tabia, and Hichem Arioui. "Automated Siamese Network Design for Image Similarity Computation". *International Conference on Content-Based Multimedia Indexing*, 2023.
- Alexandre Heuillet, Haozhe Sun, Isabelle Guyon, Felix Mohr, and Hedi Tabia. "DARIO: Differentiable vision transformer pruning with low-cost proxies". Submitted to *IEEE Journal of Selected Topics in Signal Processing*, 2023.

Remerciements

En premier lieu, je tiens à remercier mes encadrants Hedi Tabia et Hichem Arioui pour leurs conseils avisés, nos discussions passionnantes ainsi que leur bienveillance et leur soutien lors des épreuves difficiles que j'ai pu traverser pendant cette thèse. Ils ont toujours eu confiance en moi et su me remonter le moral.

Il m'est également nécessaire de remercier Blaise Hanczar, Florence d'Alché-Buc, David Picard et Raùl Santos-Rodriguez pour avoir accepté de faire partie de mon jury et leur bienveillance lors de la soutenance.

Je tiens égalemement à remercier Isabelle Guyon, Haozhe Sun et Felix Mohr pour la collaboration fructueuse que nous avons pu avoir cette année et nos disucussions toujours enrichissantes.

Merci à Pierre-Yves Oudeyer, directeur de l'équipe Flowers de l'INRIA, pour m'avoir donné le goût de la recherche en intelligence artificielle lors d'une présentation de son ouvrage "Aux sources de la parole" à la librairie Mollat.

Merci à Natalia Diaz-Rodriguez et à mon ami Fabien Couthouis pour m'avoir accompagné dans mes premiers pas (difficiles) dans le monde de la recherche.

Milles merci aussi à Kamal Youcef-Toumi et tous mes amis du MIT MRL et de MIT VISTA (Jun, Marc, Malek, Abhishek, Aadi, Xiaotong, Jiajie, Fangzhou, Steven, Tony, Mayar, Ishfaaq, Yanis, Jonas, Paul, Daniel, Katarina, Henrique, et Amaury) pour leur accueil chaleureux et les excellents moments que nous avons partagé ensemble.

Merci également à mes amis Hugo, Hugues et Rémi pour nos parties de jeu du soir qui m'ont été essentielles pour décompresser (EaW forever!).

Merci aussi à tous mes amis doctorants et post-docs d'IBISC (Quentin, Martin, Obaïda, Hicham, Victoria, Tina, Alice, Rodolfo, Sana, Lara, Ahmed, Mohammed) pour tous nos repas, échanges et discussions.

Enfin, je tiens à remercier mes parents pour leur soutien constant, surtout lors des épreuves difficiles que j'ai pu traverser pendant ces trois années, et tout particulièrement mon père pour m'avoir transmis sa passion pour l'informatique.

Synthèse en français

Durant la dernière décennie , l'Intelligence Artificielle (IA) a réalisé d'importants progrès. En effet, l'IA joue désormais un rôle de plus en plus important dans divers aspects de la société, notamment les transports publics, la sécurité, l'éducation et les soins de santé. Ces progrès remarquables peuvent être largement attribués à l'adoption généralisée de l'apprentissage profond [105, 88], une famille de réseaux de neurones artificiels capables d'apprendre efficacement des modèles en tirant parti de vastes quantités de données. Toutefois, nombre de ces modèles sont conçus de manière empirique, avec des améliorations basées sur l'intuition, telles que l'approfondissement de l'architecture [101] ou l'incorporation de connexions résiduelles [69]. Par conséquent, leurs architectures restent quelque peu "génériques", sans l'assurance d'une solution optimale.

Pour répondre à cette préoccupation, la recherche d'architecture neuronale (NAS) [231, 232, 117] a connu un développement rapide au cours des dernières années. NAS vise à surmonter l'approche par essais et erreurs et à fournir un moyen plus formel pour concevoir les architectures d'apprentissage profond. Ainsi, la dépendance à l'égard de l'ingénierie manuelle des caractéristiques et du développement de modèles a progressivement diminué, ce qui a donné lieu à de nouveaux défis, notamment l'efficacité de la mémoire, la transférabilité entre les ensembles de données et l'efficacité de calcul. Par conséquent, les chercheurs s'efforcent d'intégrer diverses approches issues de la littérature afin d'améliorer les méthodes NAS. En particulier, la différentiabilité de l'espace des paramètres, qui utilise des optimiseurs performants pour la formation de modèles d'apprentissage profond, est considérée comme l'une des voies d'exploration les plus prometteuses.

Ces dernières années, une catégorie spécifique de méthodes NAS a fait l'objet d'une attention particulière : Les approches NAS différentiables (DNAS). Ces approches représentent la tendance actuelle en matière de NAS en raison de leur efficacité, de leurs performances élevées et de leur faible coût de calcul. Cependant, malgré leur utilisation répandue, elles ne sont pas exemptes de défauts et de limitations (voir le chapitre 3).

Par conséquent, cette thèse vise à améliorer les DNAS afin de surmonter certaines de leurs limites et d'explorer l'application des DNAS à de multiples domaines et paradigmes d'apprentissage. Dans le chapitre 3, nous avons dressé l'état-de-l'art du DNAS et avons analysé les forces et faiblesses des différentes méthodes proposées. Par conséquent, le chapitre 4 se concentre sur l'amélioration d'une méthode DNAS souffrant d'une des limitation identifiée au chapitre précédent. Puis, dans le chapitre 5 nous avons choisi d'aborder un nouveau paradigme d'apprentissage (apprentissage auto-supervisé). Enfin, dans le chapitre 6, nous avon exploré diverses applications des méthodes DNAS à des cas d'utilisation concrets (contrôle d'un moteur électrique et compression d'un modèle ViT). Lors de nos expériences, nous avons ciblé différentes tâches de vision par ordinateur (par exemple, la classification d'images, la recherche d'images et la reconnaissance d'objets).

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence. 19, 30

ANN Artificial Neural Network. 21

AutoML Automated Machine Learning. 20, 136

CNN Convolutional Neural Network. 20–22, 30, 35, 93, 135

CS Computer Science. 19

CV Computer Vision. 25, 26, 109, 128

DARTS Differentiable ARchiTecture Search. 21, 33, 37, 71

DL Deep Learning. 19, 21, 30

DNAS Differentiable Neural Architecture Search. 7, 20, 31, 32, 37

EA Evolutionary Algorithms. 31, 32

MLP Multi-Layer Perceptron. 20, 21, 35, 93, 128, 129

NAS Neural Architecture Search. 7, 20, 21, 30, 71

NLP Natural Language Processing. 23

PMSM Permanent Magnet Synchronous Motor. 20, 109, 128

RL Reinforcement Learning. 31

SGD Stochastic Gradient Descent. 21

SSL Self-Supervised Learning. 20, 89, 93

ViT Vision Transformers. 20, 21, 23, 35, 109, 135

XAI Explainable Artificial Intelligence. 33, 136

1 - Introduction

> This is only a foretaste of what is to come, and only the shadow of what is going to be. We have to have some experience with the machine before we really know its capabilities. It may take years before we settle down to the new possibilities, but I do not see why it should not enter any one of the fields normally covered by the human intellect, and eventually compete on equal terms.
>
> *Dr. Alan M. Turing (1912-1954)*

Since the early days of Computer Science (CS), visionaries like Alan Turing and John von Neumann have pondered the prospect of creating an "artificial brain" capable of exhibiting creativity and initiative beyond mere number crunching. In the face of rapid technological advancements, some of these pioneers, like Herbert Simon, were optimistic, envisioning that this dream would materialize within a few decades [167]. However, von Neumann expressed skepticism as early as 1956, highlighting the fundamental distinction between biological systems and electronic computers that would pose significant challenges [186]. History has proven him right, as achieving human-level intelligence has proven to be more complex than initially believed. In particular, properly defining what intelligence is challenging as experts typically agree that there are multiple forms of intelligence (e.g., emotional, logical, or social) [28, 103]. Hence, it is difficult to measure the degree of intelligence of autonomous systems as passing a task-specific evaluation such as the Turing test [181] does not guarantee a form of “general” intelligence. Nevertheless, Artificial Intelligence (AI) has made considerable strides.

As a matter of fact, AI is increasingly playing a pivotal role in various aspects of society, encompassing public transportation, security, education, and healthcare. This remarkable progress can largely be attributed to the widespread adoption of Deep Learning (DL) [105, 88], a family of artificial neural networks capable of effectively learning patterns by leveraging vast amounts of data. However, many of these models are designed empirically, with enhancements based on intuition, such as deepening the architecture [101] or incorporating residual connections [69]. Consequently, their architectures remain some-

what "generic", lacking the assurance of an optimal solution.

To address this concern, Neural Architecture Search (NAS) [231, 232, 117] has witnessed rapid development in recent years. NAS aims to overcome the trial-and-error approach and provide a systematic, more formal means to advance the design of deep learning architectures. Furthermore, the automatic discovery of more efficient architectures holds particular relevance in the context of the ecological transition (i.e., Green Deep Learning [204]). The reliance on manual feature engineering and model development has gradually diminished, giving rise to new challenges, including memory efficiency, transferability between datasets, and computational efficiency. Consequently, researchers are striving to integrate various approaches from the literature to enhance NAS methodologies. Notably, parameter space differentiability, which employs state-of-the-art optimizers for training deep learning models, is regarded as one of the most promising avenues of exploration. Furthermore, NAS is part of the global Automated Machine Learning (AutoML) effort to remove the human factor from the ML pipeline.

In recent years, a specific category of NAS methods has gained significant attention: Differentiable NAS (DNAS). These approaches represent the current trend in NAS due to their efficiency, high performance, and low computational cost. However, despite their widespread use, they are not exempt from defects and limitations (see Chapter 3).

Therefore, this thesis aims to improve DNAS to overcome some of its limitations and explore DNAS application to multiple fields and learning paradigms. We devised several original DNAS methods for different types of deep neural networks such as Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), and Vision Transformers (ViTs). We also targeted different computer vision tasks (e.g., image classification, image retrieval, and object recognition).

This thesis is organized as follows: In Chapter 2, we first provide some preliminary reminders about DL and NAS. Then, in Chapter 3, we conducted a comprehensive survey of DNAS works, accompanied by a novel taxonomy where we identified challenges that DNAS must address. In Chapter 4, we proposed a new DNAS method to overcome one of those challenges. In Chapter 5, we applied DNAS to a new learning paradigm: Self-Supervised Learning (SSL). In Chapter 6, we explored two DNAS applications: pruning ViT models with low-cost proxies in a few minutes and using DNAS-designed networks to perform torque control of Permanent Magnet Synchronous Motors (PMSM). Finally, Chapter 7 brings a conclusion to this thesis.

2 - Preliminaries

In this chapter, we provide some useful reminders about Artificial Neural Networks (ANNs), Deep Learning (DL), Convolutional Neural Networks (CNNs), Vision Transformers (ViTs), and Neural Architecture Search (NAS). We also present Differentiable ARchiTecture Search (DARTS), the prevailing DNAS approach, in detail in Section 2.7.

2.1 . Artificial Neural Networks and Deep Learning

McCulloch and Pitts first formulated the first mathematical model of a biological neuron in 1943 [131]. Subsequently, Rosenblatt proposed the Perceptron [160] (see Fig. 2.1), a neural network algorithm leveraging McCulloch-Pitts neurons to learn patterns. Following this success, Deep Learning was created in 1965 when Ivaknenko and Lapa introduced the Multi-Layer Perceptron [88] (MLP), a network consisting of 3 layers of Perceptrons (the middle layer being dubbed "hidden layer"). With Rosenblatt's extreme machine learning model [159], this was the first exploration of how the depth of an ANN could benefit the learning performance. The question of how to design these DL architectures quickly arose.

In the following decades, many improvements were proposed, such as training with stochastic gradient descent (SGD) [4] and the backpropagation algorithm (also known as automatic differentiation) [114, 162]. These two techniques combined allow for the efficient training of ANNs with SGD computing corrective values for each learnable parameter according to a differentiable objective function and backpropagation applying these corrections by leveraging the chain rule of Leibniz.

In the past decade, Deep Learning (DL) has gained tremendous popularity since the breakthrough of Krizhevsky et al. with AlexNet [101] in the early 2010s that proved that deepening architectures could lead to significant performance gain for image classification on the challenging ImageNet large-scale dataset. This regain in interest can also be explained by the availability of high-quality training datasets (e.g., ImageNet [163]) and a drastic increase in computational resources, notably with the introduction of modern Graphics Processing Units (GPUs) by the US company Nvidia with combined 2D/3D acceleration, programmable shaders, floating point support and large quantities of onboard memory. GPUs are able to perform matrix multiplication, paramount in DL, way faster than CPUs.

AlexNet is a Convolutional Neural Network (CNN), a category of deep neural networks that was proposed several decades prior.

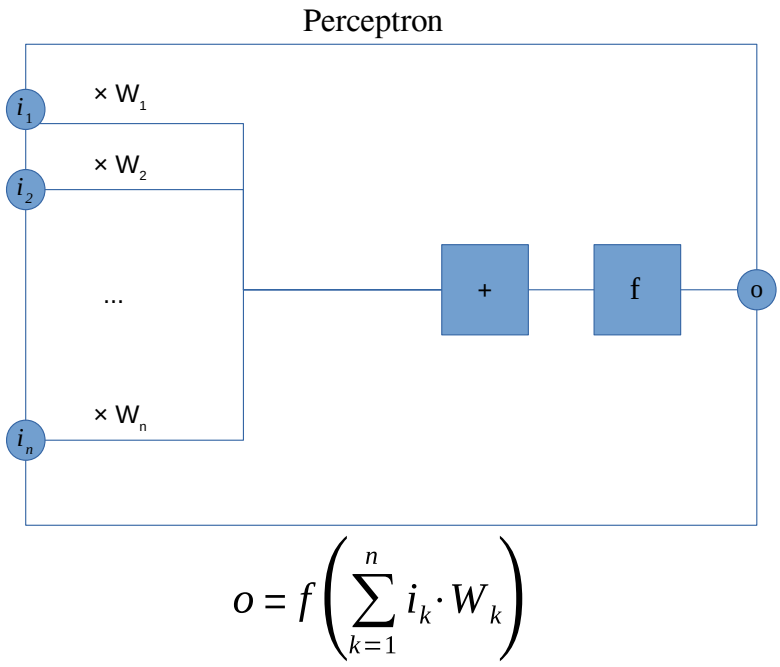


Figure 2.1: **Layout of a single layer perception.** (Source: wikimedia.org)

2.2 . Convolutional Neural Networks

In the early 1990s, LeCun et al. [106, 104] introduced Convolutional Neural Networks (CNNs) with backpropagation to perform pattern recognition. CNNs became widely used due to their innate ability to efficiently extract meaningful features from images. The basic concept behind these models is to perform a sequence of convolutions over multidimensional inputs. More specifically, a $k \times k$ filter F is applied over a $n \times n$ input x in order to produce a feature map M that is fed to the next layer. k is denoted as the kernel (or filter) size and defines the output size $|M|$ of the feature map as follows:

$$|M| = \frac{n - k + 2P}{S} + 1 \tag{2.1}$$

where P is the amount of padding added to the border of M, and S is the stride which is the number of pixels over which the filter is translated at a time when sliding over the input. Hence, the objective of CNN optimization is to learn the weights W that define the multiple filters composing the convolution operations. This kind of network also relies on pooling layers (e.g., average pooling) to reduce feature dimensionality and normalization layers (e.g., batch normalization [87]) to perform regularization. Each layer is followed by an activation function (most commonly ReLU [56]). A classifier head (fully-connected layer) is placed at the end of the CNN to transform the ag-

gregated feature maps into logits. The layout of a typical CNN is illustrated in Fig. 2.2.

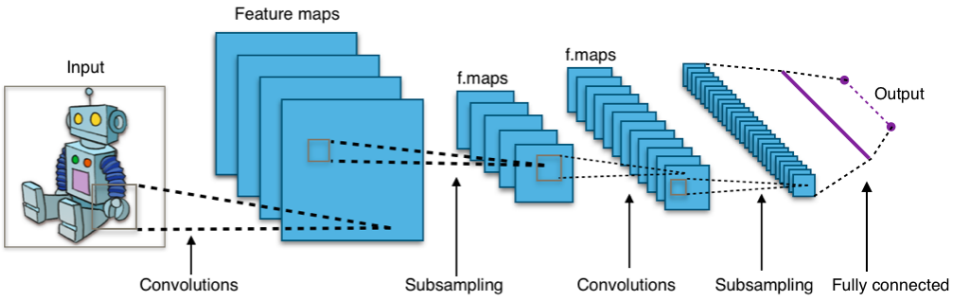


Figure 2.2: **Typical Convolutional Neural Network architecture.** (Source: wikimedia.org).

AlexNet [101] in 2012 demonstrated the relevance of using deep CNNs to perform image classification. Since then, numerous improvements and new CNN architectures were introduced, such as VGG-16 [168] (large depth), ResNet [69] (residual connections), or Xception [29] (depthwise separable convolution). However, most of these changes were driven by empiricism. To bring more formality to the field of DL architecture design and accelerate research, it is essential to look towards Neural Architecture Search (see Section 2.6).

However, a novel category of DL models is now gradually replacing CNNs: Vision Transformers.

2.3 . Vision Transformers

Vision Transformers (ViTs) were introduced by Dosovitskiy et al. [46] in 2020 and have had a tremendous impact on the computer vision field since. ViTs are based on transformers [165, 185], a DL family of models that are now ubiquitous in Natural Language Processing (NLP) [40, 19, 98, 129].

Transformers follow an encoder/decoder scheme. The encoder comprises a series of encoding layers that sequentially process the input (i.e., a sequence of words). Similarly, the decoder consists of decoding layers that operate on the output of the encoder. Each encoder layer generates encodings that capture the interrelationships between different parts of the input. These encodings are then passed on as inputs to the subsequent encoder layer. In contrast, each decoder layer performs the reverse process. It utilizes the collective contextual information from all the encodings to generate an output sequence. In summary, the encoding layers in the encoder progressively analyze the input, while the decoding layers in the decoder leverage the contextual information from the encodings to produce the desired output sequence. To accomplish this, both the encoder and decoder rely on a self-

attention mechanism. This attention mechanism allows transformers to get information about each point ("token") in the sequence by assigning a relevance weight to each previous state. It is denoted "self-attention" as this mechanism does not use RNNs to process data recurrently but simply computes attention on all the tokens simultaneously in a decomposable, highly-parallelizable way [149].

ViTs work in a similar way by considering an image as a sequence of patches (e.g., 16×16 pixels in the original ViT article [46]) and computing relationships between pairs of patches using the self-attention mechanism. Working on patches aims to reduce computational cost compared to directly comparing pairs of pixels (elementary units of images). This way, ViTs benefit from the powerful ability of transformers to learn meaningful representations from enormous amounts of data. The ViT architecture is described in Fig. 2.3.

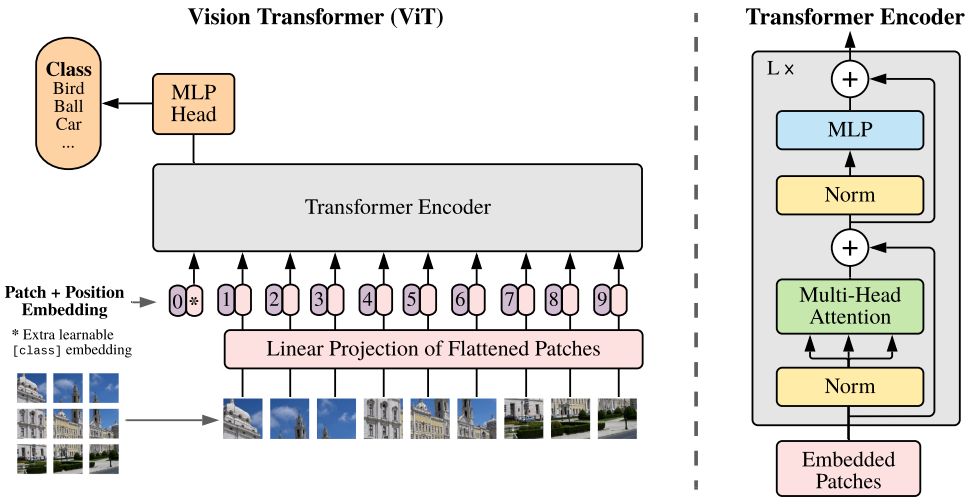


Figure 2.3: **Overview of the original Vision Transformer architecture from Dosovitskiy et al. [46].**

Since the publication of the original ViT article in 2021 [46], numerous variants have been proposed. For instance, Swim Transformers [118] designed a hierarchical transformer architecture with shifted windows that reached state-of-the-art scores in object recognition tasks. Furthermore, DeiT [179, 178] proposed a ViT architecture that can be trained using a teacher-student strategy, specifically through distillation from a CNN teacher. Finally, some works, such as MobileViT [133, 134] combined ViTs with convolutional layers in order to increase their computational efficiency and make them deployable on low-end devices such as mobile phones.

2.4 . Datasets Employed

As this thesis focuses on Computer Vision (CV) applications, we employed popular image datasets to perform tasks such as image recognition (CIFAR-10, CIFAR-100 [100], ImageNet-1K [163], Meta-Album [183]), object detection/instance segmentation (Cityscapes [35], MS-COCO [113]), and content-based image retrieval (INRIA Holidays [92]). These datasets reflect the wide diversity of tasks in CV, ranging from small, toy-like, datasets used to quickly validate a prototype up to very large and challenging collections.

2.4.1 . CIFAR-10 and CIFAR-100

CIFAR-10 [100] comprises 60000 tiny (i.e. 32x32) labeled images in 10 classes with 6000 images per class. CIFAR-100 has a similar composition but comprises 100 classes with 600 images per class. These toy-like image classification datasets are useful to quickly get preliminary results when prototyping, but they do not represent real-world scenarios. Hence, it is often necessary to perform additional experiments on large-scale datasets.

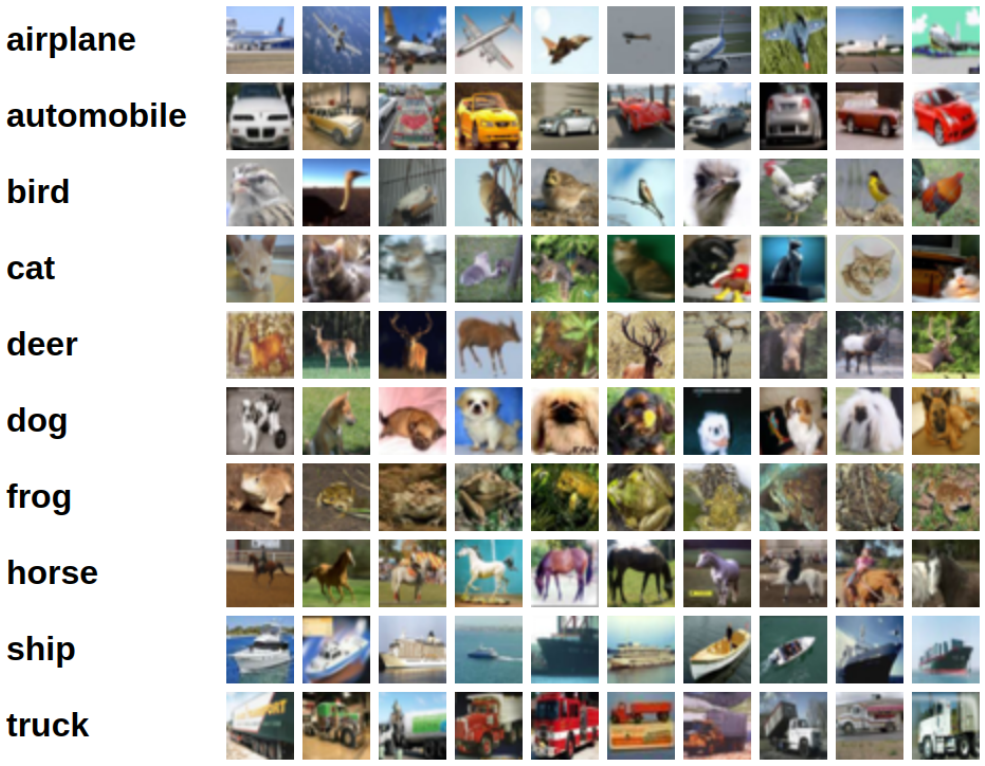


Figure 2.4: **Example images from CIFAR-10 [46].** Images from all 10 classes are featured. Figure reproduced from [33]

2.4.2 . ImageNet-1k

ImageNet-1k, also called ILSVRC2012 (or simply ImageNet), [163] is a large-scale image classification dataset comprising 1,431,167 images of various sizes

(usually resized to 224×224 when training) sorted into 1000 classes. ImageNet proved to be one of the most challenging CV ever introduced and is still nowadays considered as the primary benchmark for image classification more than 10 years after its inception. Newer iterations of ImageNet have cataloged up to 14 million images. ImageNet is a challenging dataset and has been the reference benchmark for image classification for more than a decade [176]. However, it is often relevant to also evaluate CV models on other downstream tasks such as object detection or semantic segmentation.

Figure 2.5: **Example images from ImageNet [163].** This illustration features some of ImageNet's fine-grained classes (e.g., dog breeds). Figure reproduced from [163].

2.4.3 . Cityscapes

Cityscapes [35] is a dataset composed of 5000 human-labeled high-resolution images of urban scenes taken from 50 different cities. It features annotations for semantic segmentation, instance segmentation, and panoptic segmentation. This dataset has been widely used for conducting experiments on neural network models aiming to visually understand urban street scenes (e.g., autonomous vehicles, delivery drones). Figure 2.6 features a an example annotated image from Cityscapes.

2.4.4 . MS-COCO

Microsoft Common Objects in Context (MS-COCO) [113] is a highly popular object detection and instance segmentation dataset comprising over 200,000

Figure 2.6: **Example annotated image from Cityscapes [35].** Footage taken in Zurich (Switzerland).

labeled images with 91 object categories and 1.5 million object instances. Similar to ImageNet (see Section 2.4.2), it is a challenging dataset that has been one of the standard benchmarks for conducting object detection experiments for nearly a decade.

Figure 2.7: **Example annotated images from MS-COCO [35].** Figure reproduced from [141].

2.4.5 . INRIA Holidays

INRIA Holidays [92] is an image retrieval dataset containing 1491 high-resolution images categorized into 500 image groups. Each group is composed of images that bear similar traits and should be returned by an image search model when submitting one image from the group as a query. The dataset includes a wide variety of images featuring diverse transformations (e.g., weather effects, illumination, rotations, or blurring). Holidays also includes a set of precomputed image descriptors that can be leveraged by image retrieval algorithms. Figure 2.8 features example images from Holidays.

2.4.6 . Meta-Album

Figure 2.8: **Example images from INRIA Holidays [92].** Figure reproduced from [86].

Meta-Album [183] is a meta-dataset of 40 image classification datasets. Each dataset is composed of 128×128 images, as featured in Figure 2.9. Datasets in Meta-Album come from various domains such as ecology, manufacturing, textures, object classification, and character recognition, and cover a variety of scales: microscopic, macroscopic (human scale), and distant (remote sensing). Compared to other meta-datasets, it has, by far, the largest number of domains and datasets, collected in different conditions. Hence it provides a challenging benchmark because of its diversity. Meta-Album provides 3 different versions to be used for different amounts of computational resources, which makes it accessible for benchmarking with moderate computing resources. Its micro version allows imitating the low-resource scenarios, where there are only a few training examples per class, across different domains. We chose the micro version for our experiments because the low-resource scenarios are common in real-world applications, it allows benchmarking models' performance with limited data. More importantly, Meta-Album contains datasets that are not typically used in transfer learning or meta-learning benchmarks (e.g., ImageNet) which is typically used to pre-train backbone networks, which avoids giving an unfair advantage to methods that were developed using such commonly used datasets.

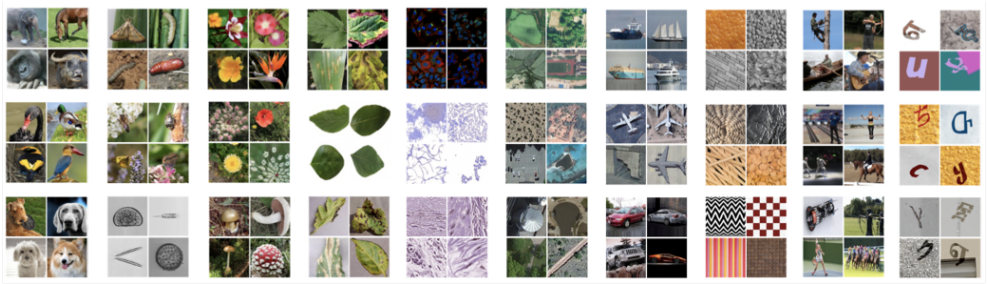


Figure 2.9: **Sample images from Meta-Album.** Reproduced from Ullah et al. [183].

2.5 . Evaluation Criteria

Most of our experiments (with the exception of Chapters 5 and 6) were conducted using a supervised learning paradigm, and thus rely on the Cross-Entropy loss $\mathcal{L}_{CE}$ when training:

$$\mathcal{L}_{CE}(x, y) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} log\frac{exp(x_{n,c})}{\sum_{i=1}^{C} exp(x_{n,i})}y_{n,c}, \tag{2.2}$$

where x and y are batched probability distributions of size N and C is the number of classes.

When evaluating performance once the training process is completed, we relied on different metrics depending on the task at hand.

2.5.1 . Image Classification

For image classification, we used the top-1 accuracy expressing the percentage of correctly classified images (according to the class associated with the highest value in the output probability distribution) in the validation set:

$$\text{top-1} = \frac{\text{number of samples correctly classified}}{\text{total number of samples}}. \tag{2.3}$$

In addition, it is usual (for large-scale datasets) to employ the top-5 accuracy. This metric is similar to the top-1 accuracy except that the 5 best values of the output probabilities are taken into account. If the label is among the classes associated with those 5 values, the sample is considered correctly labeled.

2.5.2 . Object Detection and Instance Segmentation

When performing object detection and instance segmentation, we considered the Average Precision (AP), the mean Average Precision (AP_m), and its variants the small object mean Average Precision (AP_s) and the large object mean Average Precision (AP_l). The precision is simply defined as follows:

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{2.4}$$

where TP represents the number of true positives among the predictions while FP is the number of false positives. AP also considers the recall similarly defined as:

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{2.5}$$

where FN stands for the number of false negatives among the predictions. A prediction is considered a true positive if the Intersection over Union (IoU) (in $[0, 1]$) between the segmentation mask or bounding box is superior or equal to a cut-off value (1 by default). Hence, AP corresponds to the area under the precision-recall curve for a specific class :

$$AP = \int p(r)dr, \tag{2.6}$$

where p stands for the precision and r stands for the recall. AP_m is defined as the mean AP over all classes. AP_s is the mean AP over small object classes. Similarly, AP_l is the mean AP over larger object classes. Furthermore, it is possible to derive additional metrics from AP_m by adjusting the IoU cut-off value: AP_75 (0.75 cut-off value) and AP_50 (0.5 cut-off value).

2.5.3 . Content-Based Image Retrieval

In a similar manner to object detection and instance segmentation tasks (see Section 2.5.2), we used mean Average Precision (AP_m) to evaluate models on content-based image retrieval tasks. In addition, we also considered the mean query time (in seconds) to get an estimate of the inference speed.

2.6 . Neural Architecture Search

Neural Architecture Search (NAS) aims to automatize the design of novel neural network architectures (e.g., CNNs and Transformers), a field traditionally lacking mathematical formalization and heavily relying on empiricism and intuition. The development of NAS is paramount to accelerate the discovery of ever more efficient architectures that would be able to conform to low-carbon, sustainable computing objectives (e.g., green DL, frugal AI) and be deployable on low-power embedded systems.

A typical NAS method comprises three essential components, regardless of the employed methodology, as illustrated in Figure 2.10. Firstly, there is a search space that encompasses all possible sets of hyperparameters for the architecture. Typically, this space is discrete, focusing on categorical choices of operations that form the architecture. However, it can be made continuous by applying a projection function (e.g., *softmax*) [117] or by incorporating continuous hyperparameters such as the learning rate. While theoretically infinite, in practice, the search space is constrained to finite bounds to reduce computational costs and prevent excessively deep, dense, or large ar-

chitectures. Secondly, a search strategy, also known as a search algorithm or optimizer, is responsible for exploring the search space and sampling candidate architectures. This strategy considers the performance feedback from previously sampled candidates. Lastly, an evaluation strategy assesses each selected candidate architecture identified by the search strategy. The objective of this strategy is to estimate the model's performance ideally without the need for full training, thereby minimizing computational costs.

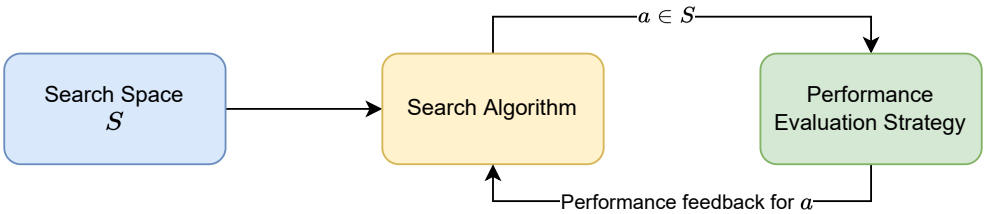


Figure 2.10: **Typical layout of a Neural Architecture Search framework.**

In the following subsections, the different trends of Neural Architecture Search will be defined: Reinforcement Learning (RL), Evolutionary Algorithms (EA), and Differentiable NAS (DNAS).

2.6.1 . Reinforcement Learning

Zoph et al. [231] utilized Reinforcement Learning (RL) to drive the architecture search process, marking one of the pioneering ML-based NAS methods. The RL controller, also known as the agent, iteratively selects new sets of hyperparameters based on the previously evaluated ones. Each set of hyperparameters undergoes full training on a given dataset, and the resulting evaluation score serves as a reward for the RL controller. To accelerate the search process, the authors incorporated parallelism and asynchronous parameter updates.

Subsequently, Zoph et al. [232] introduced the notion of cell-based hyperparameter tuning, wherein only the internal architecture of a building block (referred to as a "cell") is optimized, rather than the complete neural network. Consequently, the final model consists of a series of stacked cells. This approach, called NASNet, aimed to enhance model transferability and reduce the search space. As discussed later, this structural concept has since been adopted by other NAS approaches. However, NASNet introduced new challenges regarding the disparity between the evolving cell structure and the actual multi-cell model used for evaluation. Furthermore, the computational cost of RL-based NAS methods is often ludicrous (e.g., around 2000 GPU days for Zoph et al. [231, 232]) due to the slow convergence speed of Reinforcement Learning algorithms.

2.6.2 . Evolutionary Algorithms

Evolutionary Algorithms (EA) have emerged as a viable approach for NAS. The application of EA to NAS is straightforward, as hyperparameter sets can be considered as "genotypes" that define an architecture. In EA, a Darwinian process gradually enhances an initial population of randomly initialized architectures across multiple generations. Each new generation is derived by recombining (crossover) the genes of the best individuals in the current population to produce "children". Recent studies have incorporated evolutionary strategies such as guided evolution [119], reinforced evolution [26], and regularized EA [158]. These NAS methods were among the early implementations; however, they suffer from high computational costs, particularly when dealing with large search spaces, as they employ blind exploration during the initial iterations. Consequently, the overall computational time, often exceeding 3000 GPU days for architectures like AmoebaNet [158], becomes impractical. Additionally, most works in the literature are specific to particular cases, limiting the practicality of transferability compared to later gradient-based approaches.

2.6.3 . Gradient Descent and Differentiable NAS

Gradient descent is a powerful technique that has been known since the early 19th century, pioneered by the French mathematician Augustin-Louis Cauchy [107]. Its practical implementation dates back to the early days of computer science when Haskell Curry explored it in the 1940s [37]. Over time, gradient descent gained significant attention and eventually revolutionized machine learning with the introduction of the gradient backpropagation algorithm, as described by Linnainmaa [114] and Rumelhart et al. [162]. Hyperparameter optimization methods based on gradients have also been studied extensively since the early 1990s [6, 18]. These early works addressed control challenges in machine learning and employed gradient descent to automatically select and adjust the activation functions in artificial neural networks. This process can be likened to assigning weights to neurons, which directly impacts the shape of the model's output.

In recent years, a novel approach called Differentiable NAS (DNAS) has emerged, combining gradient descent with NAS techniques, as initially proposed by Bender et al. [9]. DNAS formulates the hyperparameter tuning problem as a continuous optimization task by treating the search space as a smooth manifold. Similar to model training, DNAS leverages gradient information to find the optimal set of hyperparameters. This approach enables the use of a supernetwork that instantiates all candidate architectures in memory, eliminating the need for independent evaluation of each candidate to obtain performance feedback. As a result, DNAS requires fewer computational resources compared to other approaches like evolutionary strategies or reinforcement learning to reach the optimal solution. The introduction of

Differentiable ARchitecTure Search (DARTS) by Liu et al. [117] marked a significant milestone in enabling the relaxation of the discrete search space through a cell-based paradigm. DARTS quickly gained popularity due to its computational efficiency, contributing to the democratization of NAS. It is worth noting that DNAS and other ML-based NAS methods discussed earlier are generally considered black-box approaches. However, the field of ML explainability, known as eXplainable Artificial Intelligence (XAI), is starting to emerge [125, 74].

2.7 . DARTS: Differentiable ARchiTecture Search

DARTS, introduced by Liu et al. [117], presents a novel approach to implementing Differentiable NAS (DNAS) using a cell-based framework. Unlike the majority of evolutionary-based NAS approaches [201, 26, 119, 158] and other DNAS methods such as FBNet [197, 187, 198], DARTS defines a modular search space consisting of building blocks referred to as "cells." This modular approach shares similarities with certain RL-based works [231, 232]. The cells in DARTS can be classified into two types: normal cells, which form the majority of the architecture, and reduction cells, which perform dimension reduction. Each cell represents a directed acyclic graph with N nodes, where each node corresponds to an intermediary data representation, such as a feature map. The nodes are interconnected by edges, and each edge $e_{i,j}$ connecting node i to node j is the sum of the outputs from $|O_{i,j}| = K$ operations. Here, $O_{i,j} = \{o^1_{i,j}, ..., o^K_{i,j}\}$ represents the set of all possible operations for edge $e_{i,j}$. Consequently, each node receives a combination of operation outputs from all preceding nodes.

For the search of CNN cells, the DARTS search space comprises $K = 7$ operations: *skip_connect*, *max_pool_3x3*, *avg_pool_3x3*, *sep_conv_3x3*, *sep_conv_5x5*, *dil_conv_3x3*, and *dil_conv_5x5*.

Therefore, the search strategy employed by DARTS involves gradually pruning incoming edges from each cell node until only a maximum of 2 edges remain (refer to Fig 2.11). For this purpose, each operation $o \in O_{i,j}$ in the mixed output of an edge $e_{i,j}$ is associated with a specific weight $\alpha^k_{i,j} \in \alpha_{i,j} = \{\alpha^1_{i,j}, ..., \alpha^K_{i,j}\}$, where $\alpha_{i,j} \in \alpha$. To transform the categorical choice of operations for edge $e_{i,j}$ into a continuous form (i.e., a probability distribution), DARTS applies the *softmax* operation σ_{SM} to $\alpha_{i,j}$. This relaxation allows for a continuous representation of the selection probabilities for the operations. Thus, the mixed output $\overline{o}_{i,j}$ of $e_{i,j}$ is defined as

$$\overline{o}_{i,j}(x) = \sum_{k=1}^{K} \frac{exp(\alpha^k_{i,j})}{\sum_{k'=1}^{K} exp(\alpha^{k'}_{i,j})} o^k_{i,j}(x) = \sum_{k=1}^{K} \sigma_{SM}(\alpha^k_{i,j}) o^k_{i,j}(x), \tag{2.7}$$

where x is the input feature and $\alpha^k_{i,j} \in \alpha_{i,j}$ is the weight associated with op-

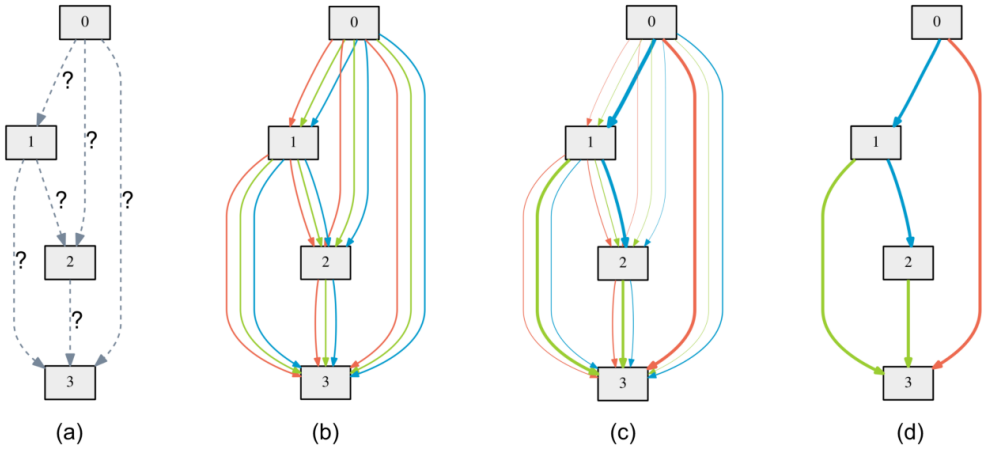


Figure 2.11: **Overview of the search strategy of DARTS.** Figure from Liu et al. [117].

eration $o_{i,j}^k \in O_{i,j}$.

The architectural parameters α are trained using gradient descent to minimize the validation loss $\mathcal{L}_{val}$, while a supernet (also referred to as a proxy network) is trained to minimize the training loss $\mathcal{L}_{train}$. The supernet consists of a compact set (e.g., 8) of stacked search cells and serves as a representation of all potential architectures. This way, DARTS solves a bi-level optimization problem formulated as

$$\begin{aligned} &\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha), \\ \text{s.t.} &w^*(\alpha) = \operatorname*{argmin}_{w} \mathcal{L}_{train}(w, \alpha), \end{aligned} \tag{2.8}$$

where w denotes the supernet weights. The gradient of the architectural parameters α is thus computed as follows:

$$\Delta_\alpha \mathcal{L}_{val} = \frac{\partial \mathcal{L}_{val}}{\partial \alpha} + \frac{\partial \mathcal{L}_{val}}{\partial w} \frac{\partial w^*(\alpha)}{\partial \alpha}, \tag{2.9}$$

where $w^*(\alpha) = \text{argmin}_w \mathcal{L}_{train}(w, \alpha)$ (i.e., the optimal value of w obtained by minimizing the training loss $\mathcal{L}_{train}$). Finally, after the completion of the search phase, the internal structure of each cell type is discretized to form the final model by selecting edges using a *softmax* operation. Stacking repetitive sequences of the two cell types makes it straightforward to derive a final architecture of any desired size.

This way, the search process of DARTS does not require fully training each candidate architecture. Instead, it utilizes the supernet as an approximator to obtain performance feedback by considering the current weights w as equivalent to the optimal weights w^*. As a result, DARTS exhibits significantly faster execution compared to RL-based or EA-based approaches.

Hence, Differentiable NAS possesses significant advantages compared to other NAS paradigms. In this thesis, we aim to improve DNAS (e.g., by addressing the limitations of DARTS) and explore new domains of application. Specifically, we devised several original DNAS methods for different types of deep neural networks such as Multi-Layer Perceptrons (MLPs, see Chapter 5), Convolutional Neural Networks (CNNs, see Chapter 4), and Vision Transformers (ViTs, see Chapter 6). We also targeted different computer vision tasks (e.g., image classification, image retrieval, and object recognition).

DNAS and DARTS are discussed in depth in Chapter 3, where we conducted a literature survey on recent DNAS methods, highlighted the current challenges posed by DNAS, and identified the emerging trends to overcome those challenges.

3 - Literature survey of state-of-the-art Differentiable Neural Architecture Search

The objective of this chapter is to provide a comprehensive discussion and analysis of recent advancements in Differentiable NAS (DNAS) using a novel taxonomy. DNAS methods can be broadly categorized into two classes: **(a)** derivatives of DARTS [117], which constitute the majority (63%) of the reviewed works, and **(b)** other DNAS studies. The works in category **(a)** assume that DARTS, being one of the most popular DNAS methods, has a solid foundation but can be further enhanced to overcome its limitations (as discussed in Section 2.7). The substantial number of works falling into category **(a)** can be attributed to the enduring popularity of DARTS over the past few years. Despite being an older method (published in 2019), DARTS continues to inspire new publications up to 2023 [190, 95], thus passing the test of time. On the other hand, category **(b)** encompasses studies that propose novel DNAS algorithms and search spaces. In many instances [197, 15, 187], these studies argue that the limitations of DARTS are inherent to its design and advocate for a fresh start.

However, this classification approach is simplistic and does not provide a comprehensive understanding of the DNAS field. Therefore, in this chapter, we introduce a novel taxonomy that categorizes methods based on the specific challenges they address rather than merely falling into categories **(a)** or **(b)**. We have identified four distinct challenges: **(I)** Bridging the optimization gap between the proxy used during the search process and the final models while addressing gradient approximation issues, **(II)** Addressing the over-representation of non-parametric operations (e.g., skip connections), **(III)** Enhancing computational efficiency and reducing latency during inference, and **(IV)** Overcoming the inherent limitations imposed by the search space in DNAS. These challenges are elaborated in detail in Section 3.1.

The rest of the chapter is organized as follows: Section 3.2 presents a literature review of 30 recent DNAS works. Section 3.3 features an analysis of the type of tasks DNAS can be applied to. Finally, Section 3.4 gives insights on the future trends developing in the DNAS field and concludes this chapter.

3.1 . DARTS and the challenges of Differentiable NAS

In Section 2.7, we presented the Differentiable ARchiTecture Search (DARTS) family, which is currently the prevailing DNAS approach.

However, DARTS suffers from major limitations and introduces new challenges. Firstly, as discussed in several articles (Heuillet et al. [76], Chu et al.

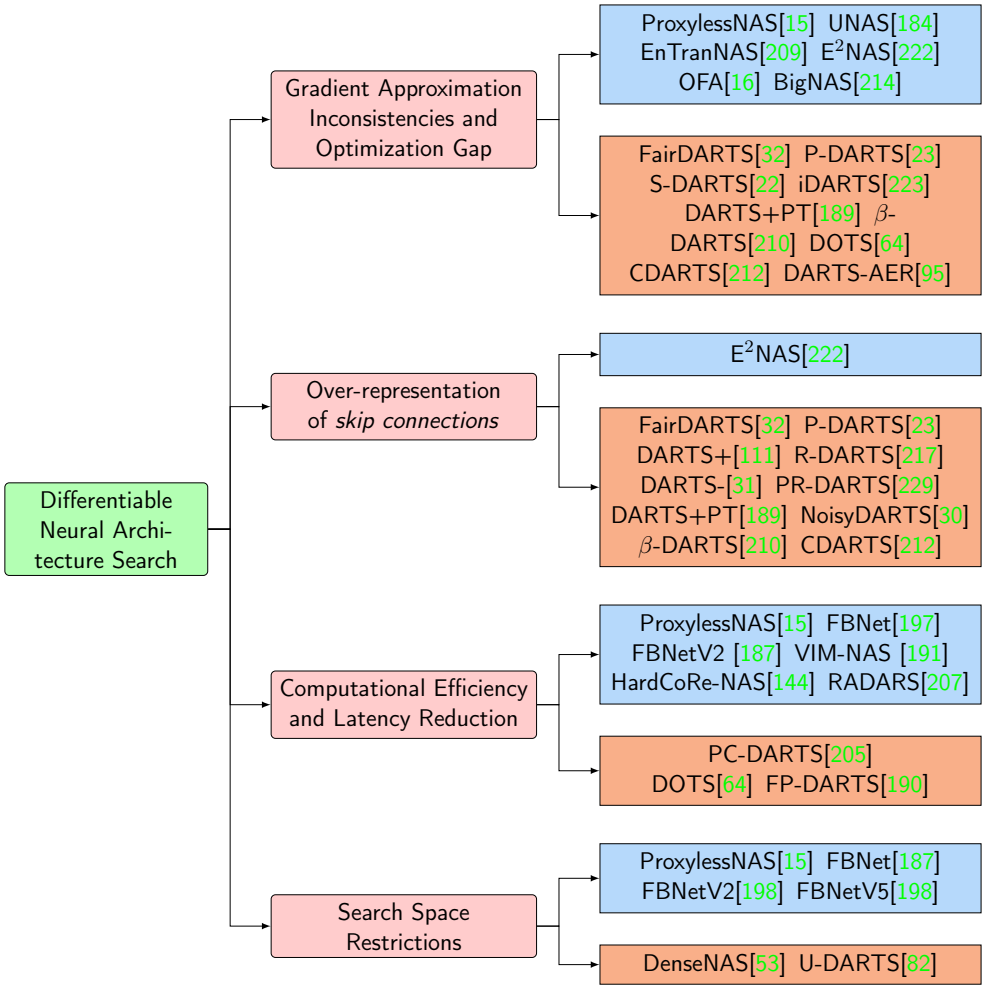


Figure 3.1: **Taxonomy of the reviewed Differentiable Neural Architecture Search literature.** References in orange, and light blue correspond to DARTS-based, and non-DARTS-based approaches respectively.

[31, 32], and Ye et al. [210]), gradient approximation methods inherently result in **inconsistencies during the optimization process (I)**, impacting the architectural parameters. Additionally, due to the limited convergence ability of the *softmax* function (which is itself a smooth approximation of the *argmax* function), the final probability distribution is dominated by values that are very close to each other. Consequently, the standard deviation of the distribution is low compared to the mean, making the discretization process more challenging. For example, distinguishing between operation o_1 with a probability of $p_{o_1} = 0.92$ and operation o_2 with a probability of $p_{o_2} = 0.91$ is non-trivial. Therefore, it is important to note the significant disparity between the proxy network (smaller, with mixed outputs on edges) used during the search process and the final discretized model (larger, with a maximum of 2 operations

per edge). Chen et al. [23] have also emphasized that this gap arises when transferring the model to a different dataset than the one used during the search phase.

Secondly, DARTS tends to **overly represent *skip connections* (II)** within the discovered architectures. This issue is closely connected to the first limitation since, as argued by Chu et al. [32], the *softmax* operation amplifies the exclusive competition among different operations. In other words, if one operation is favored, it comes at the expense of others. Edges that incorporate at least one *skip connection* resemble residual blocks [69] and provide a rapid performance boost by accelerating forward and backward operations, which causes other operations to be suppressed by the *softmax* function. Furthermore, *skip connections* are weight-free operations and lack the ability to effectively learn data representations. Consequently, there is a global degradation in performance as *skip connections* are selected even in edges where they are not the most suitable operation. Figure 3.2 provides an illustration of the phenomenon of excessive representation of *skip connections*.

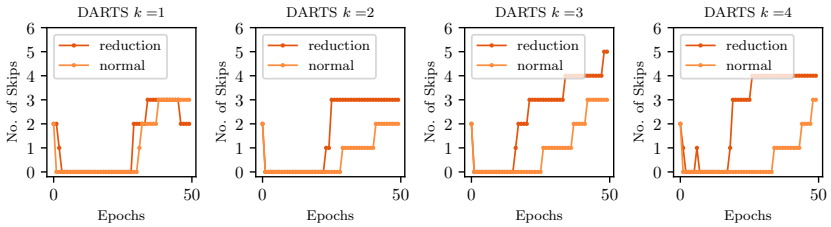


Figure 3.2: **Evolution of the number of *skip connections* w.r.t. the number of search epochs.** Figure from Chu et al. [32]

Thirdly, DARTS presents another challenge in terms of **efficiently exploring the search space (III)**, which refers to limiting the computational resources required by the search algorithm. Equation 2.7 demonstrates that a supernet must instantiate every potential path connecting each pair of nodes, storing them in memory. While this limitation is manageable when using toy or proxy datasets like CIFAR-10 and CIFAR-100 [100], it becomes especially problematic when dealing with large-scale datasets such as ImageNet [39] or MS-COCO [113]. Fortunately, modern Nvidia GPUs (starting from the Volta architecture) allow leveraging Tensor Cores, which are specialized hardware for matrix multiplication, through Automatic Mixed Precision (AMP) [139]. By utilizing the IEEE half-precision format (FP16) [85] when single-precision (FP32) is unnecessary, the computation speed is accelerated, and the memory footprint is reduced. However, it is essential to note that using lower precision can introduce numerical instability (e.g., gradient overflow). Therefore, it is crucial for researchers to prioritize the design of efficient Differentiable Neural Architecture Search (DNAS) approaches rather than relying solely on AMP techniques.

Lastly, DARTS faces a significant limitation in the form of **a highly restricted search space (IV)**. By only considering two building blocks, the optimization problem becomes considerably simpler. However, empirical evidence has demonstrated that modern high-performing CNN architectures, such as ResNets [69], ResNext [202], or EfficientNetV2 [174], consist of more than just two different blocks. Therefore, increasing the diversity of discoverable architectures is a key challenge that DARTS-based methods must overcome. Figure 3.1 presents a visual representation of our proposed taxonomic tree, summarizing the different works reviewed and their corresponding classification in DNAS.

In Section 3.2, we present a thorough review of some of the most influential Differentiable Neural Architecture Search (DNAS) methods, including DARTS derivatives and other approaches. We analyzed how these methods addressed the four challenges we identified **(I, II, III, and IV)**.

3.2 . Literature Review of Differentiable NAS

In this section, we conducted a comprehensive review of recent literature on Differentiable Neural Architecture Search (DNAS), with a specific focus on one of its most extensively studied subcategories: DARTS [117] and its derivatives. We examined a total of **30 approaches**, each of which is discussed in the following subsections based on the specific challenge (as outlined in Section 2.7) it aims to address. Our proposed taxonomy, presented in Fig. 3.1, serves as the basis for categorizing these approaches. It is worth noting that some approaches are mentioned in multiple subsections as they tackle multiple challenges within the field of DNAS. To provide an overview of our review, we present a summary table in Table 3.1.

3.2.1 . Gradient Approximation Inconsistencies and Optimization Gap

FairDARTS [32] tackled DARTS' gradient-related issues by replacing *softmax* for the categorical choice of operations by the *sigmoid* operation σ. This change is motivated by the fact that, contrary to *softmax*, σ does not create exclusive competition between the different operations (i.e., the weights associated with operations can independently increase or decrease). This improves fairness in the operation selection and thus results in better gradient approximations (see Section 3.2.1).In addition, FairDARTS introduced a novel loss function dubbed zero-one loss and denoted $\mathcal{L}_{01}$. This loss function aims to push the sigmoid values of the architectural weights (i.e., $\sigma(\alpha)$) towards 0 or 1 to minimize the discretization gap. Its gradient magnitude is adequately designed to let the α weights fluctuate but still pull them towards 0 or 1 if they

Table 3.1: **Summary of the literature we reviewed in this survey.** We included DARTS [117] for comparison purposes. The search cost is expressed with the GPU used by the authors of the original article. † denotes models searched on CIFAR-10 or CIFAR-100 [100].

Title	Type	Challenges tackled	Top 1 accuracy on ImageNet (%)	Search cost (GPU days)	Search space
DARTS [117]	N.A.	N.A.	73.1	4	*darts*
ProxylessNAS [15]	other	**(I, III, IV)**	75.1	8.3	*MobileNet-like*
P-DARTS† [23]	DARTS-based	**(I, II)**	75.3	0.3	*darts*
FBNet [197]	other	**(III, IV)**	74.9	9	*MobileNet-like*
PC-DARTS [205]	DARTS-based	**(III)**	75.8	3.8	*darts*
DARTS+ [111]	DARTS-based	**(II)**	76.1	6.8	*MobileNet-like*
PR-DARTS† [229]	DARTS-based	**(II)**	75.9	0.17	*darts*
OFA [16]	other	**(I, III)**	80.0	1.7	*MobileNet-like*
BigNAS [214]	other	**(I, III)**	80.9	1.6	*MobileNet-like*
FBNetV2 [187]	other	**(III, IV)**	77.2	25	*custom*
R-DARTS† [217]	DARTS-based	**(II)**	-	1.6	*custom*
S-DARTS† [22]	DARTS-based	**(I)**	74.8	1.3	*darts*
FairDARTS [32]	DARTS-based	**(I, II)**	75.6	3	*MobileNet-like*
DenseNAS [53]	DARTS-based	**(IV)**	75.3	2.7	*MobileNet-like*
UNAS [184]	other	**(I)**	75.5	4.3	*MobileNet-like*
iDARTS† [220]	DARTS-based	**(I)**	75.7	-	*darts*
FBNetV5 [198]	other	**(IV)**	81.7	>100	*custom*
NoisyDARTS [30]	DARTS-based	**(II)**	76.1	12	*MobileNet-like*
DARTS- [31]	DARTS-based	**(II)**	76.2	4.5	*MobileNet-like*
VIM-NAS [191]	other	**(III)**	76.2	0.26	*darts*
DOTS [64]	DARTS-based	**(I, III)**	76.0	1.3	*darts*
DARTS+PT† [189]	DARTS-based	**(I, II)**	74.5	0.8	*darts*
HardCoRe-NAS [144]	other	**(III)**	77.9	16.7	*custom*
EnTranNAS [209]	other	**(I)**	75.7	1.9	*darts*
RADARS [207]	other	**III**	73.8	3.1	*MobileNet-like*
$beta$-DARTS [210]	DARTS-based	**(I, II)**	76.1	0.4	*darts*
CDARTS [212]	DARTS-based	**(I, II)**	76.3	1.7	*darts*
DARTS-AER [95]	DARTS-based	**(I)**	76.0	N.D.	*darts*
FP-DARTS [190]	DARTS-based	**(III)**	76.3	2.4	*darts*
U-DARTS [82]	DARTS-based	**(IV)**	73.8	N.D.	*darts*

stray away from 0.5. $\mathcal{L}_{01}$ is designed as follows:

$$\mathcal{L}_{01}(\alpha) = -\frac{1}{N}\sum_{i}^{N}(\sigma(\alpha_i) - 0.5)^2, \tag{3.1}$$

where N is the number of nodes in a cell. $\mathcal{L}_{01}$ is differentiable and thus can be backpropagated to help optimize the architectural weights α. This loss is combined with $\mathcal{L}_{val}$ to form the total loss

$$\mathcal{L}_F(w^*, \alpha) = \mathcal{L}_{val}(w^*(\alpha), \alpha) + w_{01}\mathcal{L}_{01} \tag{3.2}$$

where w_{01} is an hyperparameter weighting $\mathcal{L}_{01}$.

ProxylessNAS [15] also sought to close the optimization gap but strayed away from what had been set up by DARTS. The authors proposed to search directly on the target large-scale dataset (e.g., ImageNet [39]) instead of transferring from a small-scale proxy dataset (e.g., CIFAR-10) as done by DARTS and most of its derivatives. To achieve this, they designed a search space that is no longer composed of repetitive building blocks but instead comprises an entire architecture and includes additional candidate operations. However, this comes at the cost of a greatly increased memory consumption as, if we recall Eq. 2.7, every output feature vector associated with every path in the mixed output of every cell edge must be instantiated and stored in GPU memory. To alleviate this issue, ProxylessNAS replaced DARTS' real-value architectural weights α with binary gates g that output one-hot vectors according to a probability distribution $\{p_1, ..., p_K\}$:

$$g = \text{binarize}(p_1, ..., p_K) = \begin{cases} [1, 0, ..., 0] & \text{with probability } p_1 \\ ... & \\ [0, 0, ..., 1] & \text{with probability } p_K \end{cases} \tag{3.3}$$

Thus, Eq. 2.7 is modified as follows:

$$\overline{o}_{i,j}^{Binary}(x) = \sum_{k=1}^{K} g_{i,j}^{k} o_{i,j}^{k}(x) = \begin{cases} o_{i,j}^{1}(x) & \text{with probability } p_1 \\ ... & \\ o_{i,j}^{K}(x) & \text{with probability } p_K \end{cases} \tag{3.4}$$

This mechanism allows entire paths to be binarized and instantiates only one path at a time in memory during the search phase. Thus, it reduces memory consumption to the same level as a regular model. ProxylessNAS successfully overperforms DARTS by 2 % on ImageNet. Nevertheless, these positive results come with a drastically increased search cost from 1.5 GPU days (DARTS) to 8.3 GPU days.

Another work, entitled Progressive DARTS (**P-DARTS**) [23], focused on reducing the optimization gap between the search and final architectures by improving the proxy model used during search. More precisely, the authors gradually increased the proxy network depth during search (e.g., from 5 cells to 20 cells) in contrast with the original DARTS that uses a fixed 8-cell proxy network that is later derived into a 20-cell final model. Furthermore, the number of candidate operations is progressively reduced according to their performance score. This search space approximation method alleviates the computational efficiency issues encountered when increasing the depth of the proxy network. This process, resumed in Fig. 3.3, improved performance by around 0.5 % on CIFAR-10/CIFAR-100 [100] and reduced the search cost from 1.5 GPU days to 0.3 GPU day compared to DARTS.

SmoothDARTS (**SDARTS**) was designed by Chen et al. [22] as a way to stabilize the bi-level optimization in DARTS (see Eq. 2.8). Similarly to the authors

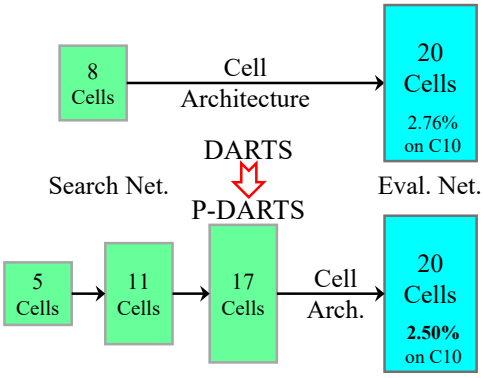


Figure 3.3: **Layout of the P-DARTS search process.** Figure from Chen et al. [23].

of R-DARTS [217] (discussed in Section 3.2.2), they argued that the optimization gap between the proxy model and the final discretized architecture is highly correlated (inversely proportional) to the spectral norm of the Hessian matrix of the validation loss $\Delta^2_\alpha \mathcal{L}_{val}$. Hence, they proposed to smooth the validation landscape of DARTS by computing $\min_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha)$ using $w^*(\alpha)$ obtained either through random smoothing (SDARTS-RS) or through adversarial training (SDARTS-ADV). SDARTS-RS reformulates Eq. 2.8 as

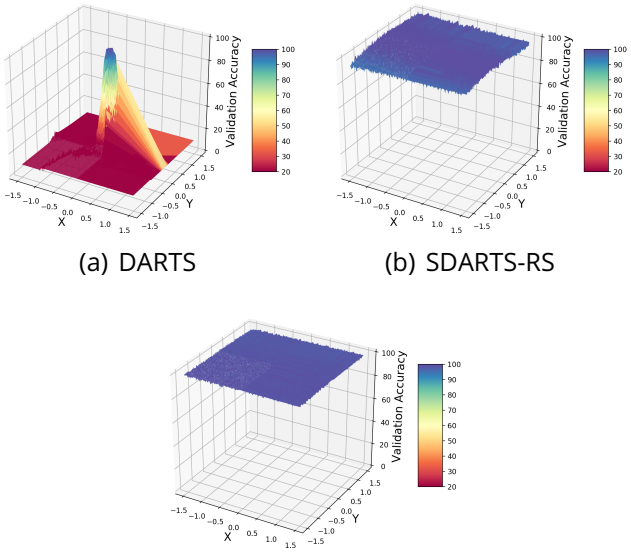
$$w^*(\alpha) = \underset{w}{\text{argmin}}\ \mathbb{E}_{\delta\ U_{[-\epsilon,\epsilon]}} \mathcal{L}_{train}(w, \alpha + \delta), \tag{3.5}$$

where δ is a perturbation sampled from the uniform distribution $U_{[-\epsilon,\epsilon]}$ between $-\epsilon$ and ϵ. The idea behind this is to minimize $\mathcal{L}_{val}(\alpha)$ under a small randomized perturbation ϵ. Similarly, SDARTS-ADV is formulated as follows:

$$w^*(\alpha) = \underset{w}{\text{argmin}}\ \underset{||\delta||<\epsilon}{\text{max}} \mathcal{L}_{train}(w, \alpha + \delta). \tag{3.6}$$

Here, Chen et al. strove to increase adversarial robustness by minimizing the worst-case loss under a certain perturbation (computed using a multistep Projected Gradient Descent). They theoretically and empirically demonstrated that both SDARTS-RS and SDARTS-ADV improve the stability and generability of DARTS (e.g., SDARTS-ADV overperforms DARTS by +1.1 % top 1 accuracy on ImageNet). However, both methods have downsides: SDARTS-ADV increases computational cost sharply, but SDARTS-RS is less accurate. Fig. 3.4 provides an illustration of the smoothing at work in SDARTS.

BigNAS [214] is an original approach where the authors proposed to palliate the optimization gap by directly reusing the weights of the supernet/one-shot model to evaluate the performance of the final architecture. This contrasts with previous works [117, 15, 16] that either retrained the network weights or preprocessed them somehow. To achieve this, BigNAS performs single-stage training using adapted versions of existing training methods such as



(a) DARTS (b) SDARTS-RS

(c) SDARTS-ADV

Figure 3.4: **The landscape of validation accuracy w.r.t. the architectural parameters α of DARTS [117], SDARTS-RS and SDARTS-ADV.** Figure from Chen et al. [22].

inplace distillation [213], the sandwich rule [213], exponential learning rate decay scheduling, or dropout-based regularization [174]. These modifications aim to stabilize training and enable BigNAS to efficiently train both large and small candidate architecture within its supernet. Furthermore, the authors proposed a coarse-to-fine architectural selection scheme where a skeleton architecture is first selected according to specific sets of requirements (e.g., input resolution, network depth, or kernel size). Then, these sets are fine-tuned with random mutations to obtain an optimal architecture. BigNAS reaches up to 80.9 % top-1 accuracy on ImageNet [39] for its largest model (9.5 M parameters, 1 GFLOPS), thus overperforming previous approaches.

Vahdat et al. [184] proposed to combine DNAS and Reinforcement Learning-based NAS in a unified framework, denoted **UNAS**, that would bring out the strengths of both approaches. This way, UNAS can search for both differentiable and non-differentiable objectives. In particular, they combine a corrected variant of the classical REINFORCE RL algorithm [194] with a Gumbel-Softmax [91] sampled DNAS algorithm to jointly search for either a differentiable or a non-differentiable objective. Hence, the gradient of a differentiable

loss $\mathcal{L}_d$ can be computed as

$$\begin{aligned}\Delta_\alpha \mathcal{L}_d &= \text{REINFORCE}(\mathcal{L}_d, c_d(\alpha)) + C(c_d(\alpha)) + \text{gumbel\_softmax}(\alpha, c_d(\alpha)) \\ &= \mathbb{E}_{\mid_\phi(\alpha)}\left[(\mathcal{L}_d(\alpha) - c_d(\alpha))\frac{\partial \log p_\phi(\alpha)}{\partial \phi}\right] - \mathbb{E}_{\mid_\phi(\alpha)}\left[\frac{\partial c_d(\alpha)}{\partial \phi}\right] + \frac{\partial \mathbb{E}_{\mid_\phi(\alpha)}\left[c_d(\alpha)\right]}{\partial \phi},\end{aligned} \tag{3.7}$$

where c_d is a control variate defined as $c_d(\alpha) = \mathbb{E}_{\diagdown_\phi(\zeta|\alpha)}\left[\mathcal{L}_d(\zeta)\right]$ used to lower the high variance of REINFORCE where ζ is a smooth architecture sampled from a conditional Gumbel-Softmax distribution $r_\phi(\zeta|\alpha)$. In addition, UNAS can help bridge the optimization gap by introducing a novel objective function $\mathcal{L}_{gen}$ to avoid architectural overfitting by taking into account the gap between $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ in the optimization process. $\mathcal{L}_{gen}$ is defined as follows:

$$\mathcal{L}_{gen}(\alpha, w) = \mathcal{L}_{train}(\alpha, w) + \lambda|\mathcal{L}_{val}(\alpha, w) - \mathcal{L}_{train}| \tag{3.8}$$

where α denotes the architectural parameters, w represents the network weights, and λ is a coefficient weighting the generalization gap. UNAS overperforms previous DNAS works on CIFAR-10/100 and ImageNet while maintaining a search cost comparable to DARTS (4 GPU days).

In addition to the optimization gap issue, Zhang et al. [222] observed that a catastrophic forgetting problem (multi-model forgetting [10]) occurs in the supernet's weights training, leading to a deterioration of the optimization process for all the candidate architectures derived from the supernet. To palliate these issues, they introduced **E^{2}NAS** (Exploration Enhancing Neural Architecture Search with Architecture Complementation), a novel DNAS approach that leverages a VGAE (Variational Graph AutoEncoder) to create an injection between the final discrete architectures and the continuous search space. More precisely, an asynchronous message-passing scheme encodes the architecture into an injective space by encoding the final output C of the network into a continuous representation z (i.e., a latent space). Hence, the hidden state h_v of node v is defined as

$$h_v = \mathcal{U}(w_v, h_v^{in}) \text{ with } h_v^{in} = \mathcal{G}(h_u : u \to v), \tag{3.9}$$

where $\mathcal{U}$ is a function updating the hidden state h_v^{in} obtained by aggregating all its predecessors with function $\mathcal{G}$. Since both $\mathcal{G}$ and $\mathcal{U}$ are injective, the VGAE maps C to z injectively. From then, E^{2}NAS performs differentiable architecture search on the latent continuous space. In addition, a new complementation loss $\mathcal{L}_c$ is introduced to tackle the catastrophic forgetting problem. This loss works in conjunction with a replay buffer that contains the last architecture α_{i-1} along with another complementary architecture α_i^c. $\mathcal{L}_{train}$ in Eq. 2.8 is replaced with L_c defined for weights $w_i = w^*(\alpha_i)$ and $w_i^c = w^*(\alpha_i^c)$ at step i as

$$\mathcal{L}_c(w_i) = (1-\epsilon)\mathcal{L}_{CE} + \epsilon(\mathcal{L}_{CE}(w_i^c) + \mathcal{L}_{CE}(w_{i-1})) + \eta\mathcal{R}(w_i), \tag{3.10}$$

where $\mathcal{L}_{CE}$ is the Cross-Entropy loss, $\mathcal{R}$ is a l_2 regularization term, ϵ is a value that balances between optimizing the current architecture (exploitation) or preventing other alternatives from vanishing (exploration). E^{2}NAS successfully overperformed previous works on the three datasets available in NAS-Bench-201 [45] (e.g., a +29.45 % top 1 accuracy improvement on ImageNet-16-120).

Zhang et al. [223] proposed **iDARTS**, a solution that reformulates the optimization process of DARTS with a Neumann-approximation of the Implicit Theorem Function (IFT) [121]. Concretely, the architectural parameter gradients $\Delta_\alpha \mathcal{L}_{val}$ (see Eq. 2.9) are calculated as follows:

$$\Delta_\alpha \mathcal{L}_{val} = \frac{\partial \mathcal{L}_{val}}{\partial \alpha} - \frac{\partial \mathcal{L}_{val}}{\partial w} \left[\frac{\partial^2 \mathcal{L}_{train}}{\partial w^2} \right]^{-1} \frac{\partial^2 \mathcal{L}_{train}}{\partial \alpha \partial w}, \tag{3.11}$$

where $\mathcal{L}_{val}$ is the validation loss, $\mathcal{L}_{train}$ is the training loss, and w are the network weights. However, it is computationally intensive to compute the inverse of the Hessian matrix $\frac{\partial^2 \mathcal{L}_{train}}{\partial w \partial w}$ in Eq. 3.11. Hence, to alleviate this burden, the authors approximated this inverse matrix using a Neumann series [121]. This Neumann approximation is computed in a stochastic setting where minibatches are used instead of the whole dataset. Thus, the stochastic approximation of the gradients described in Eq. 3.11 is formulated as follows:

$$\Delta_\alpha \hat{\mathcal{L}}^i_{val}(w^j(\alpha), \alpha) = \frac{\partial \mathcal{L}^i_{val}}{\partial \alpha} - \gamma \frac{\partial \mathcal{L}^i_{val}}{\partial w} \sum_{k=0}^{K} \left[I - \frac{\partial^2 \mathcal{L}^j_{train}}{\partial w^2} \right]^k \frac{\partial^2 \mathcal{L}^j_{train}}{\partial \alpha \partial w}, \tag{3.12}$$

where i and j are minibatches randomly sampled from the training and validation datasets respectively, γ is the learning rate, K is the number of terms of the Neumann series used for approximation, and I is the identity matrix. This reformulation of the architectural gradient computation performs multiple optimization steps before updating α, hence making $w(\alpha)$ closer to its optimal value $w^*(\alpha)$. The authors empirically showed that iDARTS improves performance over standard DARTS by 2.6 % on ImageNet [39].

Wang et al. [189] argued that the optimization gap in DARTS is linked to the architecture selection process, as the α weight values associated with an operation might not always reflect this operation's strength. They defined the discretization accuracy at convergence of an operation as the supernet accuracy after discretizing to this operation and fine-tuning the remaining network until it converges again. Hence, Fig. 3.5 showcases empirical evidence that the discretization accuracy at convergence of an operation does not necessarily match its α weight value. In fact, some operations with a small α value can reach a high discretization accuracy, further reinforcing the critical aspect of the underlying architecture selection problem.

To alleviate this issue, the authors proposed a perturbation-based architecture selection (PT) where each operation of each edge of the architecture

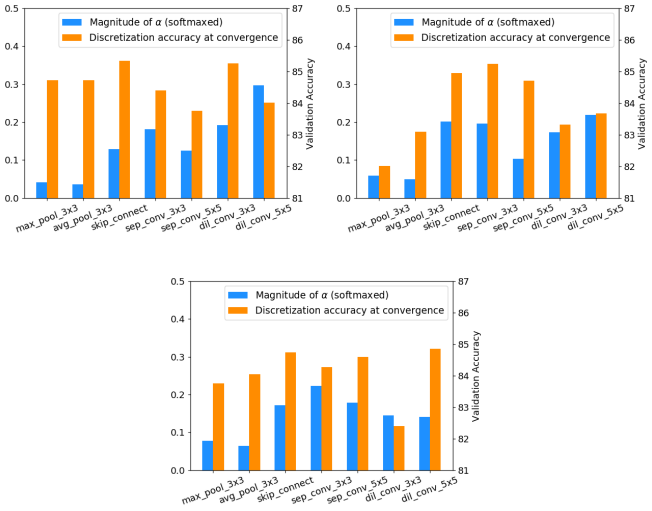


Figure 3.5: **Bar chart comparing the value of α w.r.t. the discretization accuracy at convergence for each operation of 3 randomly selected edges from a pretrained DARTS model.** Figure from Wang et al. [189].

is masked in turn. Then, the operation that leads to the highest drop in performance when masked is considered to be the most important on that edge. This process is not too invasive as it only masks one operation at a time, thus making the supernet accuracy close to the one of the unmodified supernet. Finally, the authors showed that training a DARTS supernet normally and then using PT to discretize the architecture (a process denoted **DARTS+PT**) significantly improves performance (e.g., +0.4 % top 1 accuracy on CIFAR-10 compared to DARTS).

In the continuation of DARTS- (discussed in Section 3.2.2), **β-DARTS** [210] introduced a novel and very simple regularization method called Beta-Decay inspired from $\mathcal{L}_2$ regularization that involves imposition restrictions on the architectural parameters to reduce optimization discrepancies. This regularization occurs on the α parameters after the *softmax* activation and consists of a straightforward loss function $\mathcal{L}_{Beta}$:

$$\mathcal{L}_{Beta}(\alpha) = log(\sum_{k=1}^{K} \exp(\alpha^k)), \tag{3.13}$$

where K is the total number of candidate operations. $\mathcal{L}_{Beta}$ is differentiable and added to the validation loss $\mathcal{L}_{val}$ pondered by a parameter denoted λ. Thus, Eq. 2.8 is modified as follows:

$$\min_{\alpha}(\mathcal{L}_{val}(w^*(\alpha), \alpha) + \lambda \mathcal{L}_{Beta}(\alpha)). \tag{3.14}$$

According to the theoretical analysis provided by the authors, $\mathcal{L}_{Beta}$ improves

generalization and increases robustness. Ultimately, β-DARTS reached competitive scores on both small-scale (CIFAR-10/100) and large-scale (ImageNet) datasets while searching only on CIFAR-10 and CIFAR-100. The search process of β-DARTS is resumed in Fig. 3.6.

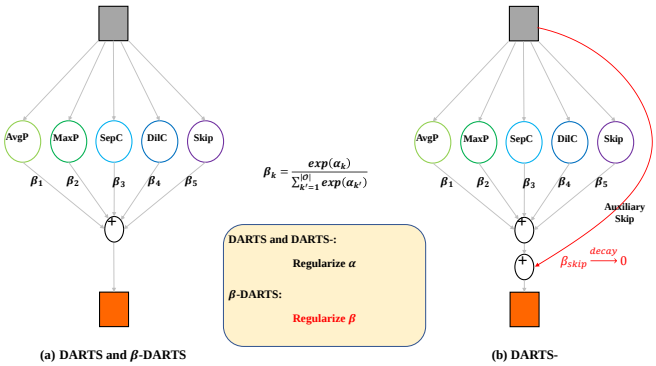


Figure 3.6: **Layout of the β-DARTS search process in comparison with DARTS [117] and DARTS- [31].** Figure from Ye et al. [210]

Gu et al. [64] argued that the ranking of operations in DARTS edges is not representative of the final model performance as it does not take correctly into account operations that are related to topology (e.g., *skip connections*), hence there is an optimization gap between the proxy model and the final model. To solve this issue, they proposed the novel concept of decoupling the operation and topology search that are performed simultaneously in the original DARTS. This solution, named **DOTS**, divides the search process into two stages. First, during the topology search stage, the topology search space $\mathcal{E}$ is continuously relaxed into topology weights that are associated with pairwise combinations of edges. For instance, considering node x_j, $\mathcal{E}_{x_j}$ is defined as follows:

$$\mathcal{E}_{x_j} = \{\langle e_{i_1,j}, e_{i_2,j}\rangle | 0 < i_1 < i_2 < j\}. \tag{3.15}$$

Moreover, for each edge $e_{i,j}$, weights of combinations containing this edge are aggregated into $\gamma_{i,j}$ weights to reduce the search cost. $\gamma_{i,j}$ is defined by the following equation:

$$\gamma_{i,j} = \sum_{c \in \mathcal{E}_{i,j}, e_{i,j} \in c} \frac{exp(\beta^c_{x_j}/T_\beta)}{N(c)\sum_{c' \in \mathcal{E}_{x_j}} exp(\beta^{c'}_{x_j}/T_\beta)}, \tag{3.16}$$

where $N(c)$ is the number of edges in edge combination c, and $\beta^c_{x_j}$ represents the weight associated with c. Eq. 3.16 uses an architectural annealing scheme with temperature T_β as previous works [147, 203] found that this mechanism helps to bridge the optimization gap when searching. In the second phase, DOTS performs an operation search to select the single optimal operation for

each edge according to architectural weights α (similarly to DARTS). However, this strategy could drop some topology-oriented operations before the topology search, thus altering the optimization process. To prevent this, DOTS introduced a group strategy where the operation search space O is divided into p subspaces on which the search process is performed independently. Once the search process is over, the best operation from each subspace is selected and merged into a new operation search space. The authors showed that this group strategy effectively preserves topology-related and topology-agnostic operations. DOTS successfully overperformed DARTS top 1 accuracy scores by +0.63 % on CIFAR-10 and by +2.7 % on ImageNet. The global process of DOTS is summarized in Fig. 3.7.

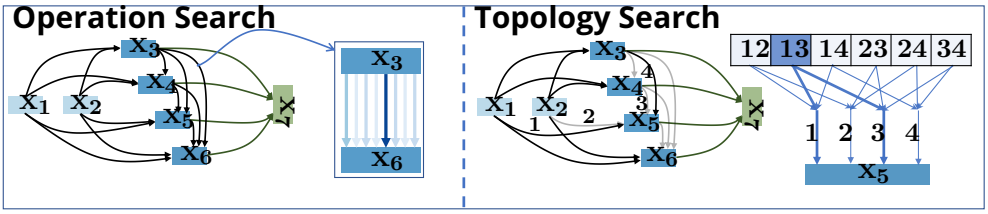


Figure 3.7: **Layout of the DOTS search process featuring both the operational and topological subprocesses.** Figure from Gu et al. [64]

Yang et al. [209] introduced **EnTranNAS** as a different solution to the optimization gap problem. EnTranNAS comprises Engine-cells (standard DARTS-like differentiable cells) and Transit-cells (transits the derived/discretized architecture). It only searches for a single cell, as the author argues it is sufficient to perform DNAS. Contrary to DARTS, the architecture discretization process in EnTranNAS is no longer part of post-processing but rather done at the end of each search iteration. Hence, the Transit-cells serve to host the currently derived architecture and transmit it to later cells. EnTranNAS includes the target (derived) architecture in the search process, resulting in higher confidence when selecting operations. In addition, the authors introduced a feature-sharing strategy to improve search efficiency, assuming that the same operation from node i to node $j > i$ always shares the same features in a single cell. Thus, Eq. 2.7 is modified as follows

$$\overline{o}_{i,j}(x) = \begin{cases} \sum_{i<j} \sum_{k=1}^{K} \frac{exp(\alpha_{i,j}^{k}/\tau)}{\sum_{k'=1}^{K} exp(\alpha_{i,j}^{k'}/\tau)} o_{i,j}^{k}(x) = p_{i,j}^{k} o_{i,j}^{k}(x), \text{ in Engine-cells,} \\ \sum_{(i,k)\in S_j} o_{i,j}^{k}(x), \text{ in Transit-cells,} \end{cases} \tag{3.17}$$

where τ is a temperature parameter that acts as a regularization factor for the differentiable process in the Engine-cells. This strategy helps to balance optimization between parametric and non-parametric operations. It reduces the computational cost by only computing feature maps of each operation only once per cell (from node i to ulterior nodes $j > i$). However, EnTranNAS

does not completely eliminate the optimization gap. Hence the authors also proposed a novel topology-search-oriented architecture derivation method dubbed EnTranNAS-DST. Concretely, they introduced an additional set of trainable parameters $\{\beta_j\}_{j=2}^{n}$ for each intermediary node j and implemented thresholds $t_j = \text{sigmoid}(\beta_j)$ to perform operation pruning on those nodes as

$$q_{i,j}^{k} = \text{ReLU}(\frac{p_{i,j}^{k}}{max_{i<j,1\leq k\leq K}\{p_{i,j}^{k}\}} - t_j). \tag{3.18}$$

If there is k s.t. $q_{i,j}^{k} \neq 0$, $q_{i,j}^{k}$ is further normalized by

$$\hat{q}_{i,j}^{k} = \frac{q_{i,j}^{k}}{\sum_k q_{i,j}^{k}}. \tag{3.19}$$

EnTranNAS-DST node output is thus obtained simply by replacing $p_{i,j}^{k}$ with $\hat{q}_{i,j}^{k}$ in Eq. 3.17. The authors experimentally showed that EnTranNAS overperforms most prior works on both CIFAR-10 (+0.28 % top 1 accuracy vs. DARTS) and ImageNet (+2.9 % top 1 accuracy compared to DARTS).

CDARTS [212] proposed to address the optimization gap issue by implementing a cyclic feedback mechanism between the search and evaluation networks analogous to a teacher-student model. The search network (composed of 8 cells) provides an intermediate architecture to the evaluation network (composed of 20 cells) and, in return, gets performance feedback. Hence, the search strategy takes into account the performance of the final discretized (and larger) architecture. Furthermore, the two networks are jointly trained and unified into a single architecture. The joint optimization problem is defined as:

$$\min_{\alpha} \mathcal{L}_{val}(w_E^*, w_S^*, \alpha) \quad \text{s.t.} \begin{cases} w_E^* = \underset{w_E}{\text{argmin}}\ \mathcal{L}_{val}(w_E, \alpha), \\ w_S^* = \underset{w_S}{\text{argmin}}\ \mathcal{L}_{train}(w_S, \alpha), \end{cases} \tag{3.20}$$

where w_S and w_E are the weights of the search and evaluation networks respectively. CDARTS' search process comprises two stages. Firstly, the separate learning stage during which both networks are trained individually on the input dataset. α weights are initialized with random values, whereas the cell architectures of the evaluated are initialized from the top-k discretization $\overline{\alpha}$ of the learned α. Secondly, the joint optimization stage where the search algorithm leverages performance feedback from the evaluation network to update α defined as follows:

$$\alpha^*, w_E^* = \underset{\alpha, w_E}{\text{argmin}}\ \mathcal{L}_{val}^{S}(w_S^*, \alpha) + \mathcal{L}_{val}^{E}(w_E, \overline{\alpha}) + \lambda \mathcal{L}_{val}^{S,E}(w_S^*, \alpha, w_E, \overline{\alpha}), \tag{3.21}$$

where $\mathcal{L}_{val}^{S,E}$ denotes the knowledge transfer procedure between the search and evaluation networks, dubbed *introspective distillation* and formulated as:

$$\mathcal{L}_{val}^{S,E}(w_S^*, \alpha, w_E, \overline{\alpha}) = \frac{T^2}{N} \sum_{i=1}^{N} p(w_E, \overline{\alpha}) \log\left(\frac{p(w_E, \overline{\alpha})}{q(w_S^*, \alpha)}\right) \tag{3.22}$$

where N is the number of training samples, T is a temperature coefficient, and p and q are the output feature logits of the evaluation and search networks respectively (computed using a *softmax*). CDARTS overperforms previous methods on DARTS search space (e.g., + 3% top 1 accuracy improvement compared to DARTS) while keeping the computational cost reasonable (1.7 GPU days). The main concept behind CDARTS is showcased in Fig. 3.8.

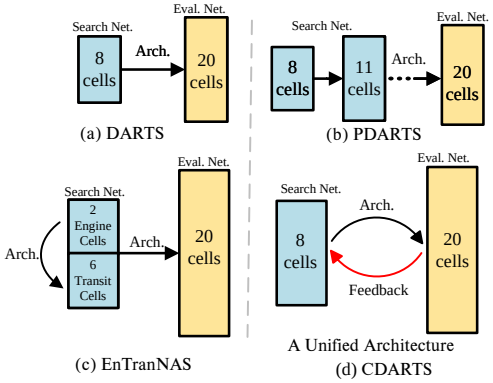


Figure 3.8: **Comparison between the search processes of DARTS [117], P-DARTS [23], EnTranNAS [209], and CDARTS [212].** Figure from Yu et al. [212]

3.2.2 . Over-representation of *skip connections* in DARTS

As already discussed in Section 3.2.1, **FairDARTS** [32] replaced *softmax* with the *sigmoid* operation σ to ensure fair competition between the different operations (i.e., the weights associated with operations can independently increase or decrease). This means that a high prominence of *skip connections* will not suppress the other operations that can thus overperform and replace them. Empirically, this results in a lessened presence of *skip connections* in the final architectures as shown in Fig. 3.9.

In a different manner, the authors of **P-DARTS** [23] managed to restrict the number of *skip connections* by introducing an operation-level *dropout* [171] to regularize the search space. More accurately, the *dropout* mechanism is placed after every *skip connection* to block the path and entice the search algorithm to explore other operations. In addition, the *dropout* rate is gradually decayed to prevent the *skip connections* from being completely suppressed (i.e., *skip connections* are heavily penalized at the start of the search process

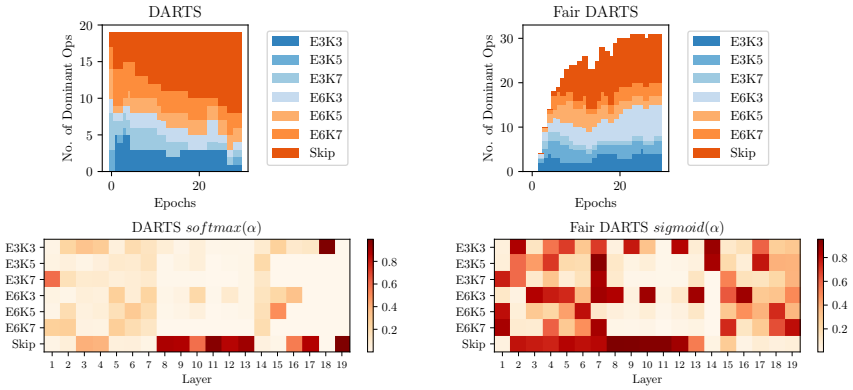


Figure 3.9: **Stacked area plot of the number of dominant operations of DARTS and FairDARTS when searching on ImageNet.** Figure from Chu et al. [32]

and then treated equally with the other operations at the end). Additional details on P-DARTS can be found in Section 3.2.1.

Liang et al. [111] argued that the over-representation of *skip connections* results from an overfitting phenomenon in the optimization process of DARTS. To alleviate this issue, they proposed **DARTS+**, an early-stopping procedure that ends the search phase if the following criteria are met:

1. *Two or more skip connections are present in the normal cell architecture.*
2. *The ranking of architecture parameters α for learnable operations becomes stable for a determined number of epochs (e.g., 10 epochs).*

The authors showed that using either of these criteria led to performance improvements over previous baselines (e.g., + 0.7 % top 1 accuracy compared to DARTS when using Criterion 1). Furthermore, they provided empirical evidence that Criterion 1 is easier to use and implement but yields less accurate results than Criterion 2. This simple early stopping procedure was dubbed DARTS+ and is illustrated by Fig. 3.10.

Zela et al. [217] also focused on robustifying DARTS as they found out that performance collapses in many cases with high dominance of unparameterized (i.e., *skip/pooling*) operations. Hence, they proposed DARTS-ES, a novel method that performs early stopping according to the eigenvalues of the Hessian matrix of the validation loss $\Delta^2_\alpha \mathcal{L}_{val}$ w.r.t. the α weights. More precisely, they showed that large eigenvalues often lead to degenerate architectures and tracked these values to stop the search process before the performance collapses. Furthermore, they implemented two different regularization methods. The first one uses a combination of well-known data augmentations techniques (Cutout [42], and ScheduledDropPath [232]). The second one increases $\mathcal{L}_2$ regularization by choosing among several factors

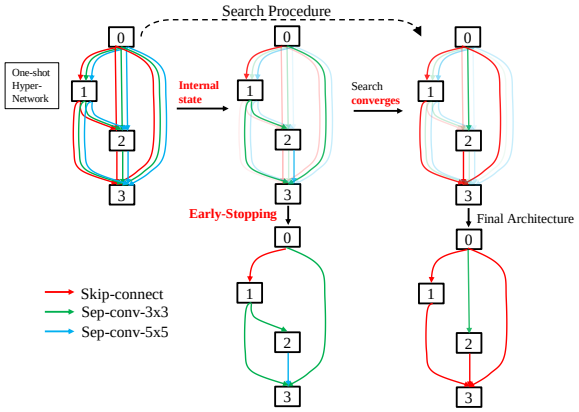


Figure 3.10: **Illustration of the early stopping process in DARTS+.** Figure from Liang et al. [111]

(e.g., $1, 3, 9, 27, 81$). Both techniques successfully increased robustness, especially when combined with DARTS-ES, an approach dubbed **R-DARTS**. The authors tested their approach on several computer vision datasets (CIFAR10/100 [100], SVHN [145]) and under different search spaces. R-DARTS improved top 1 accuracy on CIFAR-10 up to +3.64 % compared to DARTS.

As a non-DARTS approach, **E^{2}NAS** [222] (first presented in Section 3.2.1) also addressed the over-representation of non-parametric operations in their own way. They tackled *the rich-get-richer problem*, in which the optimizer is biased towards architectures with high performance in their early stage. They added a measure of the novelty into the gradient to avoid being stuck in local minima, hence computing architectural weights α_θ update as

$$\alpha_\theta^{i+1} \leftarrow \alpha_\theta^i - (1-\gamma)\nabla_{\alpha_\theta^i}\mathcal{L}_{val}(\alpha_\theta^i, w*) - \gamma\nabla_{\alpha_\theta^i}N(\alpha_\theta^i, A), \tag{3.23}$$

where $N(\alpha_\theta^i, A)$ is a measure of architecture α_θ^i from the history of architectures A. This enhancement led to a higher probability of sampling novel architectures rather than well-trained architectures from previous iterations.

DARTS- [31] tackled the global performance collapse induced by *skip connections* by adding an auxiliary *skip connection* to the classic mixed output of operations (see Eq. 2.7). The authors asserted that previous works based on analyzing the Hessian matrix eigenvalues (e.g., R-DARTS [217]) were imperfect as those methods tend to reject good architectures if they do not meet some arbitrary threshold. This auxiliary operation is pondered by β, a coefficient independent from architectural weights that is progressively decayed to 0 during the search phase. Moreover, the authors introduced β^{skip}, a parameter that denotes the importance of *skip connections* inside of the mixed output of

operations. Thus Eq. 2.7 is modified as follows:

$$\overline{o}_{i,j}(x) = (\beta + \beta_{i,j}^{skip})x + \sum_{o \in O \setminus \{skip\}} \frac{exp(\alpha_{i,j}^{o})}{\sum_{o' \in O \setminus \{skip\}} exp(\alpha_{i,j}^{o'})} o_{i,j}(x). \tag{3.24}$$

Fig. 3.6 (b) illustrates this mechanism. The authors showed that DARTS- can significantly improve robustness and stabilization during the search process, with +0.5 % improvement on CIFAR-10 [100], and +4.5 % on ImageNet [39] compared to standard DARTS. DARTS- also uses fewer computational resources than previous approaches such as R-DARTS [217]. In addition to standard DARTS, this approach is able to improve the performance of other derivatives such as P-DARTS [23] or PC-DARTS [205].

Path-Regularized Differential Network Architecture Search (**PR-DARTS**) [229] was the first method to propose a theoretical in-depth analysis of why the over-representation of *skip connections* phenomena happens and how it is connected to performance collapse. This differs from prior works [32, 23, 111] that mainly observed this issue and empirically tested their own solutions. In particular, the authors introduced a convergence theorem demonstrating that the number of *skip connections* heavily influences the supernet's convergence rate (i.e., the more *skip connections*, the faster the supernet converges). This is linked to *skip connections* faster decaying the validation loss $\mathcal{L}_{val}$, and thus leading DARTS search algorithm to increase α weights associated with *skip connections* at the cost of decreasing all other weights. To palliate this issue, they replaced architectural weights with stochastic binary gates, denoted $g_{i,j}^{k}$ for the kth operation between nodes i and j. At each iteration, $g_{i,j}^{k}$ is sampled from a Bernoulli distribution to compute the output of each node. Thus, Eq. 2.7 is modified as follows:

$$\overline{o}_{i,j} = \sum_{k=1}^{K} g_{i,j}^{k} o_{i,j}^{k}(x). \tag{3.25}$$

However, leaving the gates unregularized could bias the operation selection in cells since DARTS will increase the weights of all operations to achieve faster convergence. Furthermore, increasing the value of any operation weight could reduce or maintain the loss $\mathcal{L}_{val}$. The authors resolve these issues by using a group-structured sparsity regularization on the gates via rescaling and imposing thresholds: $g_{i,j}^{k} = min(1, max(0, a + (b-a))\overline{g}_{i,j}^{k})$ with $a < 0$ and $b > 1$, and $\overline{g}_{i,j}^{k}$ is an approximation of $g_{i,j}^{k}$ using the Gumbel reparametrization trick. This regularization is expressed by two loss functions targeting *skip* and *non-skip*

connections respectively:

$$\begin{aligned}\mathcal{L}_{skip}(\alpha) =& \zeta \sum_{l=1}^{h-1} \sum_{s=0}^{l-1} \sigma(\alpha_{s,l}^{skip} - \tau \log\left(\frac{-a}{b}\right)), \\ \mathcal{L}_{non-skip}(\alpha) =& \frac{\zeta}{r-1} \sum_{l=1}^{h-1} \sum_{s=0}^{l-1} \sum_{k=1}^{K} \sigma(\alpha_{s,l}^{k} - \tau \log\left(\frac{-a}{b}\right)),\end{aligned} \tag{3.26}$$

where σ denotes the sigmoid function, h is the path depth, $\zeta = \frac{2}{h(h-1)}$ and τ is a temperature hyperparameter. In addition, they introduced path regularization to reduce the unfair competition between deep and shallow cells (i.e., a cell containing a large amount of intermediary *skip connections*) as follows:

$$\mathcal{L}_{path}(\alpha) = \prod_{l=1}^{h-1} \sum_{k \in O_p} \sigma(\alpha_{l,l+1}^{k} - \tau \log\left(\frac{-a}{b}\right)), \tag{3.27}$$

where O_p denotes the parameterized operations. Hence, Eq. 2.8 is modified as follows:

$$\begin{aligned}\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha) + \lambda_1 \mathcal{L}_{skip}(\alpha) + \lambda_2 \mathcal{L}_{non-skip}(\alpha) - \lambda_3 \mathcal{L}_{path}(\alpha), \\ \text{s.t. } w^*(\alpha) = \operatorname*{argmin}_{w} \mathcal{L}_{train}(w, \alpha),\end{aligned} \tag{3.28}$$

where λ_1, λ_2 and λ_3 are constants. All of these improvements make PR-DARTS search for performance-oriented networks rather than fast-convergence-oriented ones. Empirical results show that PR-DARTS overperformed DARTS and earlier variants on image classification datasets (ImageNet, CIFAR-10) by a large margin.

The authors **DARTS+PT** [189] (presented in Section 3.2.1) showed that their proposed perturbation-based architecture selection method prevents *skip connections* from becoming dominant. From a theoretical aspect, they refer to Greff et al. [61], who proved that ResNet [69] layers are robust to reordering as their outputs correspond to the same estimated optimal feature map values. As the presence of *skip connections* makes DARTS' supernet resembles ResNet, this may explain why DARTS layers are also robust to reordering. Thus, this fact indicates that edges in a cell all try to estimate the same optimal feature maps m^*. Wang et al. [189] used this finding to define the estimated optimal feature maps $\bar{m}_e$ for input x_e of edge e:

$$\begin{aligned}\bar{m}_e(x_e) = \frac{exp(\alpha_{conv})}{exp(\alpha_{conv}) + exp(\alpha_{skip})} o_e(x_e) + \frac{exp(\alpha_{skip})}{exp(\alpha_{conv}) + exp(\alpha_{skip})} x_e, \\ \text{with } \alpha_{conv} \propto (x_e - m^*), \alpha_{skip} \propto (o_e(x_e) - m^*)\end{aligned} \tag{3.29}$$

where α_{conv} and α_{skip} are architectural parameters, and o_e is the mixed output of operations associated with edge e. It can be deduced from Eq. 3.29 that

the better the supernet is optimized, the closer x_e will get to m^* (since the goal of the training phase is to make edges estimate m^*). Consequently, this will widen the $(\alpha_{skip} - \alpha_{conv})$ gap and ultimately this will lead to $\alpha_{skip} > \alpha_{conv}$. However, Wang et al. showed that this only becomes problematic if the architecture selection process relies on α. On the contrary, DARTS+PT does not suffer from this issue, although it retains the same search algorithm as DARTS.

Another work dubbed **NoisyDARTS** [30] addressed this problem in an original way. The authors proposed to inject unbiased random noise during training to prevent the optimizer from increasing architectural weights associated with *skip connections* (α_{skip}) too much. They argued that adding noise is an efficient way to improve generalization by smoothing the loss landscape, as pointed out by a prior study [199]. In practice, NoisyDARTS adds Gaussian noise to the input of *skip connections*. Thus, Eq. 2.7 can be rewritten as

$$\overline{o}_{i,j}(x) = \sum_{k=1}^{K-1} \text{softmax}(\alpha_{i,j}^{k}) o_{i,j}^{k}(x) + \text{softmax}(\alpha_{i,j}^{\text{skip}}) o_{i,j}^{\text{skip}}(x + \tilde{x}), \tag{3.30}$$

where $\tilde{x} \sim \mathcal{N}(\mu, \sigma^2)$ is a random noise sampled from a Gaussian distribution parameterized by mean μ and variance σ^2 (μ = 0 and $\sigma = 0.2$ when searching on ImageNet). Despite its simplicity, NoisyDARTS managed to suppress *skip connections* and consistently overperformed prior DARTS derivatives on CIFAR-10 [100], ImageNet [39], and NAS-Bench-201 [45] (e.g., +9.7 % top 1 accuracy on ImageNet compared to DARTS).

Ye et al. (β-**DARTS**) [210] continued the work initiated by DARTS- [31] with the introduction of the Beta-Decay regularization method (presented in detail in Section 3.2.1). In addition to reducing the optimization gap problem, they argued that the Beta-Decay mechanism also alleviates the over-representation of *skip connections* issue and ensures fair competition between the operations. More accurately, the authors provide a theoretical explanation with the following equation:

$$\phi \propto \sum_{i=0}^{h-2} [(\theta_{i,h-1}^{conv} \beta_{i,h-1}^{conv})^2 \prod_{t=0}^{i-1} (\theta_{i,h-1}^{skip} \beta_{i,h-1}^{skip})], \tag{3.31}$$

where h is the number of layers in the supernet, ϕ represents the architectural weight gradients, θ represents the influence of the Beta Decay regularization, and $\beta_k = \frac{exp(\alpha_k)}{\sum_{k' \in O} exp(\alpha_{k'})} = softmax(\alpha_k)$. By taking into account that θ varies antagonistically to β (as it is a regularization function), Eq. 3.31 shows us that the convergence of networks relies more on β_{conv} than on β_{skip}. Consequently, this means that Beta-Decay helps to reduce the prominence of *skip connections*.

CDARTS [212] straightforwardly addressed the over-representation issue. They simply added a L1 regularization factor to the architectural weights of

non-parametric operations $O_{np} = skip\_connect, max\_pool\_3x3, avg\_pool\_3x3$ as:

$$\mathcal{L}_{reg} = \lambda \sum_{o \in O_{np}} |\alpha_o|, \tag{3.32}$$

where λ is a hyperparameter that balances the value of $\mathcal{L}_{reg}$. The authors showed that this method successfully prevented the operations in O_{np} from becoming dominant.

3.2.3 . Computational Efficiency and Latency Reduction

Chen et al. [23] defined the problem of *NAS in the wild* as being able to search for an architecture on a proxy dataset (e.g., CIFAR-10 [100]) to limit computational cost and successfully transfer to another, more challenging dataset (e.g., ImageNet [39]). Most DARTS derivatives followed this paradigm, contrary to most non-DARTS approaches such as ProxylessNAS [15].

As discussed above (see Section 3.2.1 for additional details), **ProxylessNAS** searches directly on the target dataset (e.g., ImageNet [39]) rather than a proxy dataset (e.g., CIFAR-10). It also enforces latency constraints on specific hardware (e.g., mobile phones, GPU, or CPU). Hence, it is a multi-objective NAS method, but one of the objectives (latency) is not differentiable. Instead, latency is measured in real-time on GPU/CPU and is predicted from a lookup table on mobile settings. ProxylessNAS successfully constrained mobile latency to a similar level to MobileNetV2 [164] and improved top 1 accuracy by 2.6 % on ImageNet.

With **PC-DARTS** (Partially-Connected DARTS), Xu et al. [205] sought to improve computational efficiency without compromising performance. To this end, they perform architecture search in only a subset of randomly sampled channels while bypassing the rest. This concept is based on the assumption that computation on this subset is an adequate approximation of the effective computation on all the channels. Considering edge $e_{i,j}$, partial channel connection involves defining a channel sampling mask $S_{i,j}$ which nullifies (i.e., assigns a weight value of 0) all channels except selected ones in the mixed output $\overline{o}_{i,j}$, thus modifying Eq. 2.7 as follows:

$$\overline{o}_{i,j}^{PC}(x) = \sum_{k=1}^{K} \frac{\exp(\alpha_{i,j}^{k})}{\sum_{k'=1}^{K} \exp(\alpha_{i,j}^{k'})} o_{i,j}^{k}(S_{i,j}x) + (1 - S_{i,j})x. \tag{3.33}$$

This process has the advantage of reducing the memory overhead by K times, with $\frac{1}{K}$ being the channel selection ratio. Subsequently, it helps reduce the search cost on CIFAR-10 from 1 GPU day (DARTS) to only 0.1 GPU day, and PC-DARTS achieves a better top 1 accuracy on ImageNet than ProxylessNAS (75.8 % vs. 75.1 %) with half the search cost. However, partial channel connection induces an inconsistency in the selection of operations across the different sampled channels. To palliate this issue, the authors introduced an additional

set of learning parameters $\beta_{i,j}$ that are shared throughout the search process to act as an *edge normalization* mechanism. The PC-DARTS approach is summarized in Fig 3.11.

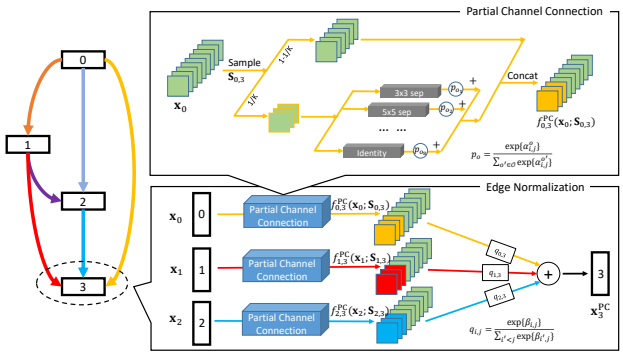


Figure 3.11: **Layout of the PC-DARTS search process.** Figure from Xu et al. [205]

Cai et al. proposed Once-for-All (**OFA**) [16] as a solution for decoupling the training and search phases with the objective of drastically reducing the computational cost of DNAS. In particular, they trained a single large supernet whose configuration (e.g., kernel size, depth, or width) can be altered and directly deployed without further training. OFA follows two stages: (1) a training phase where the different subnetworks that compose the supernet are optimized to improve their accuracy (2) a hardware-aware NAS phase (model specialization stage) where sub-networks are sampled to train accuracy and latency predictors. This enables OFA to target specific hardware and latencies. However, since simultaneously optimizing the parameters of the very large (10^{19}) number of subnetworks is prohibitively expensive, the authors introduced a novel training process during which the OFA network is progressively fine-tuned to train subnetworks of increasingly smaller size. This is akin to a pruning process performed over different modalities (i.e., input resolution, width, depth, and kernel size). Overall, OFA only requires $4.2k$ GPU hours for end-to-end training (3 times lower than DARTS [117]) to overperform all previous approaches on ImageNet in the mobile setting (i.e., less than 600M FLOPS).

Although it was not their primary objective, the authors of **DOTS** [64] were able to reduce computational cost to only 0.26 GPU day when searching on CIFAR-10 and 1.3 GPU days when searching on ImageNet. This is due to the decoupling between the topology search and operation search that greatly reduces the number of candidate operations on each edge (and thus the search space size), making both processes converge fast.

Wu et al. [197] proposed **FBNet** as a DNAS framework aimed at improving latency and computational efficiency, especially targeted at low-power

hardware such as mobile phones. Firstly, FBNet browses a search space $\mathcal{A}$ different than DARTS' that is not organized around cell building blocks but rather around layers. The macro-architecture (i.e., the pre-processing/post-processing layers, the number of intermediate layers, and their input shapes) is fixed, whereas independent architectures (from a selection of "blocks") are searched for each layer. This leads to a greater diversity of candidate architectures than in DARTS' search space. In addition, FBNet combines the standard Cross-Entropy loss $\mathcal{L}_{CE}$ with a hardware-aware latency loss $\mathcal{L}_{LAT}$ defined as follows:

$$\mathcal{L}_{LAT} = \alpha \log \left(\sum_i LAT(b_l^a) \right)^\beta, \tag{3.34}$$

where $LAT(b_l^a)$ denotes the latency of block a of the l-th layer, and α and β are coefficients weighting $\mathcal{L}_{LAT}$. The latency values are retrieved from a latency lookup table (similarly to ProxylessNAS [15]), as measuring latency from mobile processors in real time is prohibitively expensive. In addition, using a lookup table makes $\mathcal{L}_{LAT}$ differentiable. Finally, the authors devised a differentiable NAS algorithm where the search space is modeled by a stochastic supernet. Thus, only one candidate block is sampled at a time independently for each layer from a probability distribution obtained through a softmax instead of a weighted mixed output of operation as featured in DARTS (see Eq. 2.7). Consequently, the output x_l of layer l is a masked output defined as:

$$x_l = \sum_i m_{l,i} b_{l,i}(x_{l-1}), \tag{3.35}$$

where $m_{l,i}$ is a mask that equals to 1 if block $b_{l,i}$ is sampled or 0 otherwise. Therefore, the probability of sampling an architecture $a \in \mathcal{A}$ is described by the following equation:

$$P_\Theta(a) = \prod_l P_{\Theta_l}(b_l = b_{l,i}^a), \tag{3.36}$$

where Θ is composed of all the parameters that determine the sampling probabilities of blocks for each layer. Furthermore, the authors resorted to the Gumbel-Softmax [91] function to relax the discrete masks m_l into a continuous distribution and thus make the whole search process differentiable w.r.t. the sampling parameters Θ. The authors empirically showed that FBNet reached a higher top 1 accuracy (e.g., +1.8 % for FBNet-C) on ImageNet than DARTS for a 33 % lower search cost. FBNet-A also reached a latency as low as 19.8 ms when targeting a Samsung Galaxy S8.

However, FBNet is not free from limitations, and hence Wan et al. [187] designed an updated method dubbed **FBNetV2**. Their primary concern was palliating the small search space size issue in FBNet and DARTS. Consequently, they introduced a greatly enlarged search space (see Section 3.2.4). To keep

their method computationally efficient, the authors proposed DMaskingNAS. This NAS algorithm uses weight-sharing approximations to efficiently search over additional hyperparameters, such as the number of filters and the input dimensions. They kept the layer-wise DNAS paradigm described in FBNet but used a channel-masking mechanism parameterized through a Gumbel-Softmax function. Thus the output y of a block b can be computed as follows:

$$y = b(x) \circ \sum_{i=1}^{k} g_i \mathbb{1}_i, \tag{3.37}$$

where g_i denotes Gumbel weights and $\mathbb{1}_i$ is a mask vector whose first i values are 1s with the rest being 0s. This way, each block's channel number can be searched without significant additional computational cost. Furthermore, FBNetV2 searches for different input resolutions by performing resolution subsampling from the original input (i.e., extracting smaller input feature maps using the nearest neighbors method). Once the output feature map has been computed, it is upsampled into a larger fixed-size one to preserve dimensional consistency. FBNetV2 maintains a computational cost similar to FBNet despite searching on a search space up to 10^{14} times larger.

VIM-NAS (Variational Information Maximization Neural Architecture Search) [191] observed that each cell edge is considered independent in the global architecture of previous DNAS methods. In contrast, the authors introduced a novel way of formulating the NAS problem by assuming that the architectural distribution $\mathcal{A}$ is a latent representation of specific data points from dataset $\mathcal{D}$ such as there is a distribution $p_\phi(\mathcal{D}, \mathcal{A}) = p(\mathcal{D})p_\phi(\mathcal{A}|\mathcal{D})$ parameterized by ϕ. More specifically, VIM-NAS strives to maximize the mutual information $I_\phi(\mathcal{D}, \mathcal{A})$ between $\mathcal{A}$ and $\mathcal{D}$ as

$$\max_{\phi} I_\phi(\mathcal{D}, \mathcal{A}) = \mathbb{E}_{p_\phi(\mathcal{D},\mathcal{A}}[\log p_\phi(\mathcal{D}|\mathcal{A})]. \tag{3.38}$$

Thus, the objective $\mathcal{L}(\phi, \theta, \mathcal{D})$ of the DNAS process can be formulated as

$$\max_{\theta,\phi} \mathcal{L}(\phi, \theta, \mathcal{D}) = \sum_{d \in \mathcal{D}} \mathbb{E}_{p_\phi(\mathcal{D},\mathcal{A}}[\log q_\theta(\mathcal{D}, \mathcal{A}], \tag{3.39}$$

where $\log q_\theta(\mathcal{D}, \mathcal{A})$ is a supernet approximation of $\log p_\phi(\mathcal{D}|\mathcal{A})$. In practice, $p_\phi(\mathcal{D}|\mathcal{A})$ is reformulated to the Gaussian noise $\mathcal{N}(\mu_\theta, 1)$ where μ_ϕ is parameterized by the convolutional network ϕ. This makes VIM-NAS very fast as it can converge in only a single epoch in DARTS' search space (i.e., a 0.007 GPU day search cost) while providing a top-1 accuracy improvement of +0.55 % on CIFAR-10 [100] and +2.04 % on ImageNet [39] compared to DARTS.

Methods such as FBNet [197] or ProxylessNAS [15] sought to impose hardware and latency constraints softly by formulating an objective function which is a trade-off between accuracy and computational resources. In contrast,

HardCoRe-NAS (Hard Constrained diffeRentiable NAS) [144] searches for high-accuracy architectures that strictly respect a hard latency constraint. The authors reformulated the classic bilevel optimization problem of DNAS (see Eq. 2.8) as

$$\begin{aligned} \min_{\zeta \in S} \mathbb{E}_{x,y\ \mathcal{D}_{val};\hat{\zeta}\ \mathcal{P}_{\zeta}(\mathcal{S})}[\mathcal{L}_{CE}(x,y|w^*,\hat{\zeta})] \text{ s.t. } \mathrm{LAT}(\zeta) \leq T \\ w^* = \operatorname*{argmin}_{w} \mathbb{E}_{x,y\ \mathcal{D}_{train};\hat{\zeta}\ \mathcal{P}_{\zeta}(\mathcal{S})}[\mathcal{L}_{CE}(x,y|w,\hat{\zeta})], \end{aligned} \tag{3.40}$$

where $\mathcal{S}$ is a fully differentiable block-based search space parameterized by $\zeta = (\alpha, \beta)$, $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ are the train and validation datasets' distributions, $\mathcal{P}_{\zeta}(\mathcal{S})$ is a probability measure over S, and $\mathrm{LAT}(\alpha, \beta)$ is the estimated latency of the model. $\mathcal{S}$ is composed of a micro $\mathcal{A}$ (i.e., block internal architecture $c \in \mathcal{C}$) and macro (i.e., connections between blocks at every stage $s \in S$) $\mathcal{B}$ search spaces parameterized by $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{B}$ respectively. Thus, the overall expected latency $\mathrm{LAT}(\alpha, \beta)$ is computed by summing over the latency $t^s_{b,c}$ for every possible configuration $c \in \mathcal{C}$ of every block b, over all possible depths d, and over all the stages:

$$\mathrm{LAT}(\alpha, \beta) = \sum_{s=1}^{S} \sum_{b'=1}^{d} \sum_{b=1}^{b'} \sum_{c \in \mathcal{C}} \alpha^s_{b,c} \cdot t^s_{b,c} \cdot \beta^s_{b'}. \tag{3.41}$$

HardCoRe-NAS uses LAT to build a constrained search space $\mathcal{S}_{LAT} = \{\zeta | \zeta \in \mathcal{P}_{\zeta}(\mathcal{S}), \mathrm{LAT}(\zeta) \leq T\}$. Similarly to DARTS, $\mathcal{S}$ is relaxed to be continuous by searching for $\zeta \in \mathcal{S}_{LAT}$. The search process of HardCoRe-NAS is visually summarized in Fig 3.12. The authors experimentally showed that HardCoRe-NAS managed to constrain latency to the same level or lower than previous methods while overperforming them on ImageNet [39] (e.g., -9 ms latency reduction and +1.6 % top 1 accuracy improvement compared to FBNet on an Nvidia P100 GPU).

Similarly to UNAS [184], **RADARS** [207] (Reinforcement Learning Aided Differentiable Architecture Search) leverages Reinforcement Learning to help the differentiable search process. However, RADARS focuses on reducing computational and memory costs, whilst UNAS is performance-oriented. The RL algorithm prunes the search space through iterative exploration/exploitation phases. It identifies promising subsets of operations for each laver and prunes the search space of the other operations (exploration phase). Differentiable NAS is then performed on this reduced search space instead of the entire search space (exploitation phase). The authors bounded the GPU memory usage to a maximum of 12 Go (Nvidia RTX 2080ti). They showed that RADARS could reach competitive scores for restricted memory (11 Go) and time (3.08 GPU days) on ImageNet despite using a large MobileNet-like search space.

Finally, **FP-DARTS (Fast-Parallel-DARTS)** [190] proposed to decompose DARTS' search space into two sub-spaces comprising 4 operations each. These

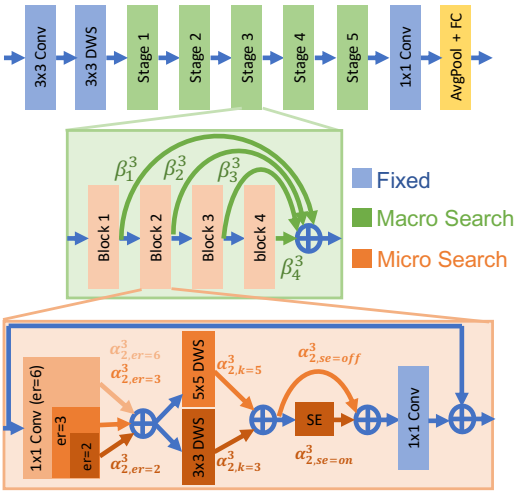


Figure 3.12: **Layout of the HardCoRe-NAS search process.** Figure from Nayman et al. [144]

two spaces are browsed in parallel to discover two subnetworks that jointly form a global supernet by addition. Furthermore, the authors introduced a binary gate to control whether a path in one of the two sub-spaces participates to the supernet training. This parallelization process aims to reduce the search cost drastically. FP-DARTS reaches competitive results on ImageNet (76.3 %).

3.2.4 . Search Space Restrictions

Bypassing the search space restrictions of DNAS (especially concerning DARTS) is one of the main goals pursued by researchers in the field. For instance, **ProxylessNAS** [15] step out of the cell-based paradigm to instead search directly for an entire architecture. However, as discussed in Section 3.2.1, this led to an exponential increase in computational resources that the authors alleviated by using a binarization mechanism to only instantiate a single path in memory at a given time.

U-DARTS (Uniform-DARTS) [82] proposed to unify the search and evaluation spaces into a single one. This space is also expanded to include searchable connections between the different cells (by opposition to the sequential connections featured in DARTS). In addition, U-DARTS uses a random sampling scheme to reduce computational overhead. Finally, a regularization method is also employed to mitigate the *skip-connection* overrepresentation issue (see Section 3.2.2). U-DARTS is able to reach competitive results on CIFAR-10 (i.e., 97.41 % top-1 accuracy with 3.3 M parameters).

The authors of DenseNAS [53] introduced a densely-connected block-based search space that allows them to search for block widths and the number of blocks per layer. They designed routing blocks with gradually increasing

widths, and each block output linked to multiple other blocks, covering a large spectrum of block widths and block numbers per layer. Each routing block comprises several shape-alignment layers and several basic layers, which are mixed outputs of the different candidate operations. The basic layers are relaxed into continuous operations using *softmax*, similarly to DARTS (see Eq. 2.7). In the same way, the routing block output is a mixed output of all the possible paths leading to the next routing block and relaxed using a *softmax*. Furthermore, DenseNAS features a chained cost estimation algorithm that aims to restrict the computational cost and latency of the final architecture. The cost of each basic layer operation is retrieved from a lookup table and is used to estimate the cost of the whole architecture as follows:

$$cost^l = \sum_{o \in O} w_o^l cost_o^l \tag{3.42}$$

$$\tilde{cost}^i = cost_b^i + \sum_{j=i+1}^{i+m} p_{i,j}(cost_{align}^{i,j} + cost_b^j), \tag{3.43}$$

where w_o^l is the *softmax* weight of operation o in basic layer l, $p_{i,j}$ is the *softmax* weight of the path from routine block B_i to routine block B_j, m is the number of subsequent blocks, and $cost_{align}^{i,j}$ is the cost of the shape-alignment layer in block B_j with input from block B_i. This cost is then integrated into the loss function

$$\mathcal{L}(w, \alpha, \beta) = \mathcal{L}_{CE} + \lambda \log_{\mathcal{T}}(\tilde{cost}^1). \tag{3.44}$$

DenseNAS overperformed previous methods on ImageNet (+2.8 % top 1 accuracy compared to DARTS) while being able to constrain the number of FLOPS and the latency.

The authors of **FBNetV2** [187] (first presented in Section 3.2.3) pointed out that the main issue in previous DNAS works (DARTS [117] and FBNet [197] primarily) is related to their severely restricted search space. Therefore, they crafted a novel search space encompassing two new hyperparameters (i.e., the number of channels and the input resolution). This novel search space comprises 10^{35} candidate architectures and is 10^{14} times larger than FBNet's. By leveraging this expanded search space, FBNetV2 overperforms FBNet on ImageNet significantly (e.g., +1.9 % top 1 accuracy for FBNetV2-F4 compared to FBNet-B).

Building upon what has been laid by previous FBNet works [197, 187, 38], **FBNetV5** [198] is an interesting work as it did not simply seek to lift search space restrictions but more specifically focused on a less-explored subject: improving the transferability of DNAS architectures between different computer vision tasks. The authors addressed this challenge by combining a differentiable NAS process with FBNetV3, a NAS approach that stepped out of DNAS and instead proposed its own novel paradigm where architectures and

training hyperparameters ("recipe") are matched to reach optimal performance. More precisely, they created a supernet trained on a multitask dataset (generated from ImageNet [39]) to disentangle the search process from the training pipeline of the target. FBNetV5 performs topology search to find the optimal backbone architecture for each task (i.e., semantic segmentation, object detection, and image classification). This is done simultaneously for all tasks. Furthermore, the authors designed a search algorithm that produces architectures for each task at a constant computational agnostic to the number of tasks. At the task level, they leverage a DNAS process following [197] where they browsed a block-based search space $\mathcal{A} = \{0, 1\}^B$ comprising B blocks. An architecture $a \in \mathcal{A}$ is hence a set of binary masks a_b sampled independently from a Bernoulli distribution. When searching on multiple tasks, the DNAS problem is relaxed as follows:

$$\min_{\pi^1,\ldots,\pi^T,w} \sum_{t=1}^{T} \mathbb{E}_{a^t\ p_{\pi^t}} \{\updownarrow^t(a^t, w)\}, \tag{3.45}$$

where a^t are architectures sampled from task-specific distributions p_{π^t}, l^t are task-specific losses, T is the number of tasks, and w denotes the supernet weights. In addition, to restrict computational cost, the authors adopt the RL algorithm REINFORCE [194] and importance sampling [67] to reduce the number of forward and backward passes of the search algorithm from T to 1. Fig. 3.13 summarizes the search process of FBNetV5. FBNetV5 successfully overperforms all previous NAS methods on datasets (ImageNet [39], COCO [113], and ADE20k [228]) corresponding to the three tasks this method focused on.

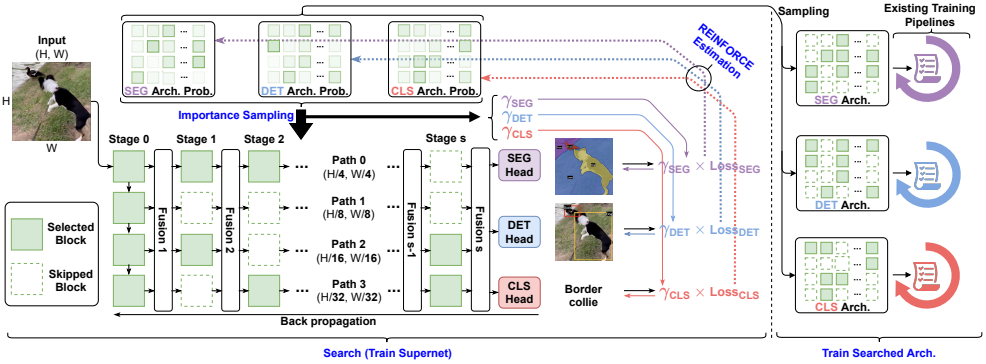


Figure 3.13: **Layout of the FBNetV5 search process.** Figure from Wu et al [198].

3.3 . Applications

It is important to note that the majority of significant DNAS works, such as DARTS [117], ProxylessNAS [15], and FBNet [197], primarily focus on enhancing Convolutional Neural Network (CNN) architectures for computer vision tasks. Consequently, these works predominantly evaluate their approaches using image classification datasets, such as CIFAR-10/100 [100] and ImageNet [39], as well as object detection and semantic segmentation datasets like MS-COCO [113], Pascal-VOC [51], and Cityscapes [35]. Nonetheless, several studies have explored DNAS in various other application domains. In addition to the major approaches discussed in the following paragraphs and Section 3.2, we have also included less prominent works that specifically focus on particular applications of DNAS. Table 3.2 provides a summary of these approaches.

Remote Sensing: Several applications have been developed to address image-oriented tasks in remote sensing, including radar image analysis [44, 224] and scene classification [152]. For example, Zhang et al. [224] employed a DARTS-based method to search for optimized AutoDL detector CNN backbones for sonar image and maritime radar image analysis.

Natural Language Processing: DARTS [117] and its variants, such as β-DARTS [210], have been utilized to search for Recurrent Neural Network (RNN) [162] architectures for Natural Language Processing (NLP) tasks, such as language modeling on datasets like the Penn Tree Bank [128] and WikiText-2 [136]. Additionally, in the same domain, AdaBERT [19] employed Differentiable NAS to compress large language models like BERT [98] into smaller task-specific models.

Medical Applications: Several works have proposed DNAS-based approaches for analyzing medical data, including MRIs [108, 122] and volumetric images [230, 215, 83]. As most DNAS approaches primarily target CNNs and computer vision tasks, their application to medical imaging is straightforward. For example, Guo et al. [65] developed a DNAS approach to modulate the composition of CNNs used for organ-at-risk segmentation in patients treated for head and neck cancer.

Reinforcement Learning: RL-DARTS [138] utilized DARTS to search for backbone architectures for on-policy and off-policy Deep Reinforcement Learning algorithms. They demonstrated that Differentiable NAS with DARTS is effective for both discrete action and continuous control environments. In the Procgen [34] environment, RL-DARTS achieved a 250 % improvement in performance over the baseline IMPALA-CNN [50].

Audio Processing: Several studies have focused on speech recognition, aiming to discover novel convolutional neural network architectures specialized in audio pattern extraction [227, 80, 43]. For instance, Hu et al. [80] employed a DARTS-based search strategy to discover Time Delay Neural Network (TDNN) architectures optimized for automatic speech recognition.

Hardware Optimization: Matching architectures with hardware has been

Table 3.2: **Summary of Differentiable NAS works according to their field of application.** Some articles are referenced under multiple categories.

Field	Subfield	References
Computer Vision	Image Classification	[117, 187, 197, 198, 32, 223, 111, 76, 31, 205, 23, 22]
	Object Detection	[130, 110]
	Video Processing	[155]
	Semantic Segmentation	[230, 115, 52]
	Image Super-Resolution	[81, 192, 200]
	Image Denoising	[60, 219]
	Pose Estimation	[225]
	Image Generation	[57]
	Facial Recognition	[109]
Remote Sensing	Radar Image Analysis	[44, 224]
	Scene Classification	[152]
Natural Language Processing	Language Modeling	[117, 210, 94]
	Keyword Spotting	[142]
Medical Applications	Medical Image Analysis	[108, 230, 65, 122, 83, 193, 215]
Reinforcement Learning	Deep Reinforcement Learning Backbone	[138]
Audio Processing	Speech Recognition	[227, 80, 43]
Hardware Optimization	Embedded Systems and Latency Reduction	[120, 15, 93, 214, 197, 207, 99, 126, 144, 216, 16, 153, 221]
	Predictive Maintenance	[220]

a major focus in DNAS. Multiple works, including ProxylessNAS [15] (discussed in detail in Section 3.2.3), have aimed to optimize computational cost and latency on various platforms, such as GPUs [15, 197, 207], CPUs [15], and embedded systems [15, 99, 120]. Ensuring the real-world deployment of Deep Learning is paramount, as consumer-grade devices are not as powerful as high-end or data-center GPUs/TPUs typically used by researchers.

3.4 . Discussion and Conclusion

In Section 3.2, we reviewed 30 recent DNAS approaches that targeted 4 different challenges (presented in Section 2.7). More than half (62 %) of these approaches are based on DARTS [117] due to the high popularity this method enjoyed from the moment it was first published (2019) until now, with novel DARTS-based methods still being proposed in 2022 [210, 212]. Each reviewed DNAS method addressed at least one of the four challenges we identified in Section 2.7.

One noteworthy fact (clearly shown in Fig.3.1) is that there is a clear partition between DARTS-based and non-DARTS-based works when considering the challenges they tackled. The vast majority (81 %) of DARTS-based methods addressed either challenge **I** (gradient approximation discrepancies) or challenge **II** (over-representations of *skip connections*) while the rest mostly targeted challenge **III** (computational efficiency and latency reduction) and challenge **IV** (search space restrictions). This can be explained as **I** and **II** are DARTS' most prominent issues and thus constitute the main leads to pursue any follow-up work. On the other hand, non-DARTS-based DNAS saw those issues are inherent to DARTS and proposed a change of paradigm that allowed

them to focus on other, more global, problems (i.e., **III** and **IV**). Let us dive into the main conclusions of each category.

(I) A large subset of methods [189, 32, 64, 209, 212, 15] agreed that the final discretized architecture is dissimilar to the proxy model used during the search process, thus resulting in the optimization gap. However, these works differ in their proposed solutions to that analysis. Some replaced the discretization process to yield models that better fit the proxy network, while others designed a novel search process when the proxy network is more closely tied to the final model (or even a proxyless search process [15]). These DNAS works yielded improved results that show the relevance of their respective contributions. However, these improvements are often marginal (i.e., less than 1 % top 1 accuracy improvement on ImageNet [39] compared to previous state-of-the-art), and there is no general consensus among all recent articles on how to reduce the optimization gap. Thus, this may indicate that, despite efforts to provide mathematical background, we still lack a formal model that would bring an optimal solution to this problem. The DNAS optimization gap is not closed yet.

(II) One interesting fact to note is that nearly every paper that addresses the *skip connection* issue provides an analysis of why DARTS fails and draws similar conclusions: the non-parametric operations have an unfair advantage as they accelerate gradient descent in the early stage by forming structures akin to residual blocks [69]. Eventually, this unfair competition suppresses parametric operations and leads to architectural "overfitting". Furthermore, most of the reviewed works (e.g., [32, 210, 23]) proposed to add regularization on the search space to prevent this phenomenon. This regularization is generally applied either before (α weights) or after (β weights) the *softmax* relaxation. This proved relevant as adding regularization prevented *skip connections* from becoming dominant and improved performance. This outcome is logical as a search space mixing parametric and non-parametric operations is inherently unbalanced, and regularization is a well-explored solution to overfitting and rebalancing ill-formed problems [11, 177]. Finally, as an alternative solution, other works [111, 217] devised early-stopping mechanisms to stop the search process before the architecture overfits. However, these approaches are based on arbitrary or empirical criteria that are less formal than regularization-based approaches, hence explaining the popularity of the latter.

(III) Some approaches [205, 207, 191] managed to reduce the search cost drastically (up to 43 times for VIM-NAS [191]) compared to DARTS. This made it possible to launch the search process on low-end, consumer-grade GPUs (and even CPUs in some cases). Hence, it contributed to making DNAS a very accessible process to automate neural network design. However, other methods [187, 15, 144] chose to trade computational cost for reduced latency at infer-

ence, hence helping Deep Learning to deploy on low-resource devices, such as mobile phones. They did so by adding the inference latency as a differentiable objective so that raw performance is no longer the only goal of the search process. To save computational resources, the latency values for a specific platform are often retrieved from a latency lookup table.

(IV) As previously stated, the fact that most DNAS methods that address search space restrictions are not DARTS-based highlights that it is an issue more closely associated with DARTS. Thus, those methods had to craft search algorithms and/or search spaces that deeply diverge from DARTS. Most notably, all approaches in that area abandoned the cell-based building block paradigm as it is one of the main elements that restrict the search space. For instance, the FBNet family [197, 187, 38, 198] relied on a novel MobileNet-like search space that is block-based rather than cell-based. ProxylessNAS [15], and DenseNAS [53] also leveraged a similar search space. Finally, the low number of studies tackling the search restrictions (i.e., 6) means that researchers mostly focused on other issues judged more urgent. In addition, using a larger search space leads to a drastic increase in computational cost, hence making it necessary to design mechanisms to save resources. This might explain why proposing a novel method addressing this issue is difficult. Nevertheless, it is still paramount to find ways to address this issue.

Overall, over the past few years, all of these works contributed to making DNAS more and more viable, with the ability to discover architectures that can now far surpass the performance of handcrafted ones (e.g., CDARTS [212] reaches 78.2 % top 1 accuracy on ImageNet [39] vs. 74.7 % for MobileNetV2 [164]). In addition, the computational cost (i.e., the number of FLOPs) and the latency can be restrained to deploy models on embedded platforms such as mobile phones [15, 197, 144]. This makes DNAS (and Deep Learning by extension) easier to deploy to solve real-world tasks and accessible to a wider audience. Thus, one could argue that DNAS is now a maturing field, with some methods being included in major AutoML libraries such as Microsoft NNI [140] and NASLib [161].

As discussed in Section 3.3, DNAS has already been applied to a wide range of applications (mainly related to computer vision). In the near future, with DNAS becoming more and more robust, we can expect it to be applied to an ever-increasing number of fields. For instance, works on transformers improvements [27] and self-supervised learning [75] have already started to emerge. Other already explored fields, such as Generative Adversarial Networks (GANs) design [57], could be further expanded to other applications, such as face generation [97].

Nevertheless, DNAS still suffers from limitations as none of the proposed methods could solve all four identified challenges simultaneously. Hence, no DNAS method could impose itself as a novel standard. This may substantially

explain how DARTS withstood the test of time and remains popular despite its age. This also indicates that DNAS has room for improvement and has not reached its full potential yet.

Hence, based on this taxonomy, a novel method called D-DARTS is introduced in Chapter 4 to address challenge **(IV)**, as we observed that only a limited number of studies have focused on tackling this particular issue. Furthermore, we endeavored to extend the application of DNAS to novel fields, such as self-supervised learning, as discussed in Chapter 5, and novel architectures like Vision Transformers (ViTs), as explored in Chapter 6.

4 - Improving DARTS: Distributed Differentiable Neural Architecture Search

As discussed in Chapter 3, the most popular approach in Neural Architecture Search (NAS) is currently Differentiable ARchitecTure Search (DARTS) [117]. Its key advantage lies in significantly reducing the search cost compared to earlier approaches [231, 232], thanks to its 2-cell search space. However, the limitation of searching for only two types of cells significantly restricts the search space and hampers the diversity of candidate architectures (i.e., challenge **(IV)**, see Chapter 3). To address this issue, we propose a novel approach that directly searches for a complete network with individualized cells, greatly increasing the size of the search space. This network delegates the search process to the cell level in a distributed manner. Moreover, to further optimize the super network architecture, we leverage Game Theory and introduce a novel loss function based on the concept of Shapley value [166]. Additionally, the new distributed supernet structure allows us to directly encode existing architectures as architectural weights within the search space. Thus, we also propose implementing and optimizing state-of-the-art handcrafted architectures (e.g., ResNet50 [69] or Xception [29]). Finally, we introduce a novel metric for computing distances between architectures in the search space, which can be considered as a manifold. We refer to this method as Distributed Differentiable ARchiTecture Search (D-DARTS). We demonstrate that D-DARTS can reach competitive results on multiple computer vision datasets.

This chapter is structured as follows: In Section 4.1, we detail the proposed method. Section 4.2 presents the results of a set of experiments conducted on popular computer vision datasets. Finally, Section 4.3 discusses the results of the experimental study and brings a conclusion to this chapter while giving some insights on promising directions for future work.

4.1 . Proposed Approach

In this section, we present the main contributions of D-DARTS: a novel distributed DNAS approach with a Shapley-value-based loss and a method to optimize existing handcrafted architectures using DNAS.

4.1.1 . Delegating Search to Cell-Level Subnets

Our method's key idea is to increase architectural diversity by delegating the search process to subnets nested in each cell. Each subnet is itself a full neural network with its own optimizer, criterion, scheduler, input, hidden, and output layers, as shown in Fig. 4.1. This way, each cell that composes

the global supernet is individualized. Thus, instead of searching for building blocks as DARTS [117] do, we increase the number of searched cells to an arbitrary n and directly seek for a full n-layer convolutional neural network where each cell is highly specialized (contrary to generic building blocks). Thus, we trade the weight-sharing process introduced in DARTS for greater flexibility and creativity. Nonetheless, cells still belong either to the *normal* or *reduction* class, depending on their position in the network.

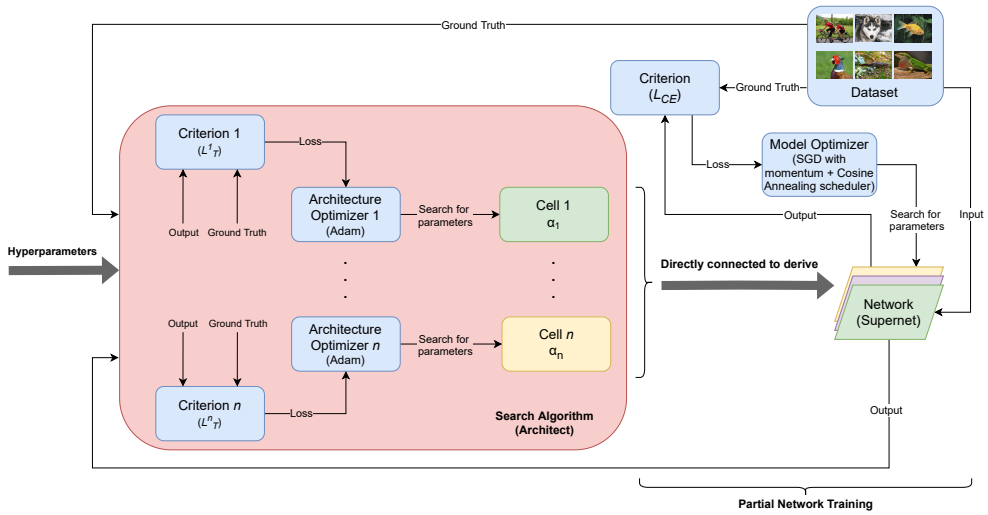


Figure 4.1: **Layout of the search process used in D-DARTS.** Each cell i is independent with its own optimizer, scheduler, and criterion (our proposed ablation loss $\mathcal{L}_T$). Each cell searches for its architectural parameters α_i, which makes the entire search process distributed. The searched cells are directly connected to each other to build a supernet trained to validate their performance.

As explained in Section 2.7 of Chapter 4, DARTS only searches for two types of cells and stacks them multiple times to form a network as deep as needed. This weight-sharing process has the advantage of reducing search space to a limited set of parameters (i.e., α_{normal} and α_{reduce}), thus saving time and hardware resources. However, this approach limits both the search space size and the originality of the derived architectures as all the underlying structure is human-designed. In particular, the search space size $s(K, L)$ of a single cell with K primitive operations to select from (within a maximum of 2 from different incoming edges) and L steps (i.e., nodes standing for intermediary data representations) can be computed as follows:

$$s(K, L) = \prod_{i=1}^{L} \frac{(i+1)i}{2} K^2. \tag{4.1}$$

Following Eq. (4.1), using DARTS default parameters ($K = 7$ and $L = 4$), the

search space of a single cell comprises around 10^9 possible configurations. Thus, as both *normal* and *reduction* cells share the same K and L, the total search space size of DARTS is around 10^{18} possibilities. This number is comparable to other differentiable NAS works [15, 187], but far lower than those of Reinforcement Learning based NAS methods [231, 232] that describe architecture topologies using sequential layer-wise operations, which are also far less efficient. Our approach, dubbed *D-DARTS*, effectively expands the search space by a factor of $10^{(n-2)*9}$ (according to Eq. (4.1)) where n is the total number of searched cells. Thus, the total size of D-DARTS search space reaches around 10^{72} when considering $n = 8$. In Section 4.2, we show that smaller (e.g., 4 or 8 layers) D-DARTS architectures can achieve similar or higher performance than larger (e.g., 14 or 20 layers) architectures on popular datasets.

4.1.2 . Adding a New Cell-Specific Loss

In addition to the new network structure introduced in Section 4.1.1, we have designed a novel cell-specific loss function dubbed "ablation loss" (denoted as $\mathcal{L}_{AB}$). As the number of searched cells increased, we faced a greater learning challenge with a substantial number of additional parameters to consider. Consequently, the global loss functions used in DARTS [117] and FairDARTS [32] cannot accurately assess the performance of each cell, but instead only consider the global performance of the supernet. In contrast, our new loss function is specific to each cell and acts as an additive loss, building upon the global loss function $\mathcal{L}_F$ introduced in FairDARTS (see Eq. 3.2), which has demonstrated significant improvements over the original one [117].

The main concept behind this ablation loss function is to conduct a limited ablation study on the cell level. Ablation studies have long been crucial in validating the effectiveness of neural network architectures [137]. By calculating the difference in the supernet loss $\mathcal{L}_{CE}$ (i.e., the cross-entropy loss) with and without each individual cell activated, we can obtain a measure of their respective contributions, which we refer to as their "marginal contributions," denoted as $M_C = \{M_C^1, ..., M_C^n\}$ for an n-cell network. This method draws inspiration from Shapley values [166], a game theory technique widely used in Explainable Artificial Intelligence (XAI) to assess the contributions of model features to the final output [124, 74]. Thus, cell C_i marginal contribution M_C^i is computed as follows:

$$M_C^i = \mathcal{L}_{CE}^{(C)} - \mathcal{L}_{CE}^{(C \setminus \{C_i\})}, \tag{4.2}$$

where C is the set containing all cells such as $C = \{C_1, ..., C_n\}$. Once we obtained all the marginal contributions M_C, we apply the following formula to compute the ablation loss $\mathcal{L}_{AB}^i$ of cell C_i:

$$\mathcal{L}_{AB}^i = \begin{cases} \frac{M_C^i - mean(M_C)}{mean(M_C)} & \text{if } mean(M_C) \neq 0, \\ 0 & \text{else.} \end{cases} \tag{4.3}$$

$\mathcal{L}^i_{AB}$ expresses how important the marginal contribution of cell i is w.r.t. the mean of all the marginal contributions. Finally, $\mathcal{L}^i_{AB}$ is then added to FairDARTS global loss $\mathcal{L}_F$ (see Eq.(3.2)) to form the total loss $\mathcal{L}^i_T$, weighted by the hyperparameter w_{AB}:

$$\mathcal{L}^i_T = \mathcal{L}_F + w_{AB}\mathcal{L}^i_{AB}. \tag{4.4}$$

Finally, when expanding Eq. (4.4), we obtain:

$$\mathcal{L}^i_T = \mathcal{L}_{CE} + w_{0-1}\mathcal{L}_{0-1} + w_{AB}\mathcal{L}^i_{AB}, \tag{4.5}$$

where $\mathcal{L}_{01}$ is the Zero-One loss introduced in FairDARTS [32] (see Eq. (3.1)) and w_{0-1} is an hyperparameter weighting $\mathcal{L}_{0-1}$.

In Section 4.2, we show that $\mathcal{L}^i_T$ can warm start the search process and substantially increase performance. However, it also increases GPU memory usage, as shown in Section 4.2.2. Combined with the cell individualization process introduced in Section 4.1.1, $\mathcal{L}^i_{AB}$ constitutes the core of the D-DARTS approach as described in Algorithm 1.

4.1.3 . Building Larger Networks from a Few Highly Specialized Cells

As presented in Section 4.1.1, we adopt a different approach to Neural Architecture Search (NAS) by directly searching for a "full" network consisting of multiple individual cells, as opposed to searching for building block cells as done in DARTS [117]. However, this method has a downside: searching for many cells requires a significant amount of memory, as each cell must have its own optimizer, criterion, and parameters. Hence, finding a way to use a larger number of cells without additional search time can be advantageous, especially when dealing with highly complex datasets like ImageNet [163]. To address this, we have developed a novel algorithm that derives larger architectures from an already searched smaller one, drawing inspiration from the approach used in DARTS [117].

The key idea behind this concept is to retain the global layout of the smaller architecture, with the reduction cells positioned at $1/3$ and $2/3$ of the network, similar to DARTS and FairDARTS [32]. We then replicate the searched structure of *normal* cells in the intervals between the *reduction* cells until we achieve the desired number of cells. This enables us to obtain a larger architecture without launching a new search, thus avoiding any overhead. The process is summarized in Algorithm 2.

In Section 4.2, we show that doubling the number of layers this way can result in a 1% increase of top1 accuracy when evaluating on CIFAR-100 [100]. However, the gain is more limited (around 0.15 %) for simpler datasets such as CIFAR-10 [100], where the base model already performs very well.

4.1.4 . Encoding Handcrafted Architectures in DARTS (DARTOpti)

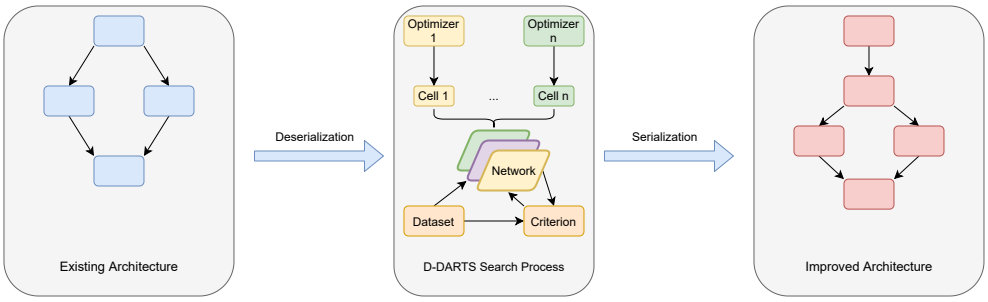


Figure 4.2: **Overall description of the process used in DARTOpti.**

Unlike the original DARTS [117] or one of its derivatives [23, 210, 64], D-DARTS individualizes each cell and allows for the encoding of large handcrafted architectures [29, 69], which typically contain multiple types of layers (e.g., 13 for Xception [29]). The primary motivation behind this is to utilize existing architectures as initial points for the optimization process. Since these handcrafted architectures have been carefully optimized, they can be considered local minima of the search space. This process involves several key procedures, which are as follows:

(i) The architecture is manually encoded as a D-DARTS-compatible *genotype* (i.e., a data structure that describes each cell design, the location of reduction cells in the architecture, and the maximum number of steps in a cell). When searching from or training this architecture, the corresponding *genotype* is automatically loaded and deserialized into α weights (see Section 2.7), which can be optimized by the search process of D-DARTS.

(ii) A weight-sharing mechanism is introduced to reduce the search cost and redundancy in cells, especially when starting from architectures with many layers. Every identical cell in the baseline architecture will share the same weights. For instance, Xception [29] consists of 13 cells, but only 5 of those cells are different. Therefore, in this case, the number of optimizers will be reduced from 13 to 5.

(iii) The supernet is warm-started for 5 epochs before the actual search starts. This allows the performance of the baseline architecture to be assessed and taken into account by the search algorithm.

(iv) 5 new operations are added to the original search space S of DARTS: (1) `conv_3x1_1x3`, (2) `conv_7x1_1x7`, (3) `simple_conv_1x1`, (4) `simple_conv_3x3`, and (5) `bottleneck_1x3x1`. This brings the total number of operations to 12, unlocking new possibilities at the cost of further increasing the already large search space of D-DARTS. This new search space is denoted S_o.

This process, known as DARTOpti, is visually summarized in Figure 4.2. We demonstrate in Section 4.2 that it can successfully optimize handcrafted architectures and achieve competitive results on ImageNet [101].

4.1.5 . Implementing a Metric to Quantify the Distance Between Architectures

In this section, we propose a metric M that effectively asserts the distance between two architectures that belong to the search space S_o. This was motivated by the need to quantify how much DARTOpti (see Section 4.1.4) improved handcrafted architectures. Another motivation for M is the possibility of drawing statistics on S_o such as the average distance between starting and final architectures, the maximum distance between starting and final architectures, or finding if the amelioration of performance in the final architectures is correlated to the number of changes made to the original ones.

The metric M is defined as follows. We follow a hierarchical approach. First, we need to compare (i) architectural operations with each other, then (ii) cell edges, (iii) cells, and finally (iv) architectures.

(i) A way to compare operations is needed, as they are the most elementary components of a *genotype*. Naturally, we cannot consider all operations as having an equal value. They are all different, with some seeming to share more similarities. For instance, one would think that `sep_conv_3x3` has more in common with the other convolution operations than with the pooling operations. To that end, we established an experimental protocol that would allow us to assign a performance score to each operation, assuming that similar operations would also obtain similar scores. In particular, each of the 12 operations in S_o (see Subsection 4.1.4) was benchmarked for each edge of each cell of a small 3-cell proxy network based on ResNet18 [69] architecture (i.e., residual blocks). This way, the proxy network is similar to an existing CNN architecture. All edge operations were disabled except for the benchmarked one. We recorded the maximum top-1 accuracy reached by the proxy network among 4 training runs of 10 epochs each (to counteract the stochasticity of the gradient descent algorithm) on CIFAR-10 [100]. Training the proxy network took 1 GPU day on a single Nvidia RTX A6000 with a batch size of 256. Final scores for each operation were obtained by taking the median value observed across all edges of all cells. Results are presented in Table 4.1. They conform to expectations with analogous operations obtaining comparable scores (e.g., `dil_conv_3x3` and `dil_conv_5x5` are only 0.03 % apart).

(ii) The distance between two edges, each encoded as a binary vector, can be computed using the Hamming distance [66]. This widely used function measures the distance between vectors by evaluating the minimum number of changes needed to translate one vector into the other according to given weights. These weights correspond to the operation score values featured in Table 4.1. The Hamming distance is described in Algorithm 3.

(iii) The Hausdorff distance [68] is utilized to evaluate the distance between cells. It is described in the following equation:

$$d_H(X, Y) = \max\{\sup_{x \in X} d(x, Y), \sup_{y \in Y} d(X, y)\}, \tag{4.6}$$

Table 4.1: **Benchmark scores obtained for each of the 12 operations in DARTOpti search space S_o.** Standard deviation across all cell edges is reported for each operation.

Operation	Score (in %)
`conv_3x1_1x3`	82.76 ± 1.82
`conv_7x1_1x7`	82.72 ± 1.14
`max_pool_3x3`	82.96 ± 2.19
`avg_pool_3x3`	82.51 ± 2.13
`skip_connect`	82.15 ± 2.33
`simple_conv_1x1`	82.27 ± 1.99
`simple_conv_3x3`	83.12 ± 2.21
`sep_conv_3x3`	83.19 ± 0.89
`sep_conv_5x5`	84.87 ± 0.73
`dil_conv_3x3`	82.96 ± 1.56
`dil_conv_5x5`	82.99 ± 1.24
`bottleneck_1x3x1`	83.06 ± 0.77

where X and Y are two subsets of S_o (i.e., cells in this case), and d is a distance metric able to quantify the distance between points of the subsets. Here, d corresponds to the Hamming distance. The Hausdorff distance measures the closeness of every point in one set to those in the other set. Thus, it better considers structural similarity, which is paramount for comparing topologies. However, since the Hausdorff distance of Eq. (4.6) is directive, we need to enforce symmetry in order to create a metric M_H:

$$M_H(X, Y) = \max(d_H(X, Y), d_H(Y, X)). \tag{4.7}$$

(iv) We can compare two different architectures A and B by evaluating the average distance between their pairwise cells with the architectural distance metric M:

$$M(A, B) = \sum_{i=0}^{n} \frac{M_H(C_i^A, C_i^B)}{n}, \tag{4.8}$$

where n is the number of cells in both architectures. In fact, since every cell is individual and linearly linked to the others, it is not relevant to compare cells that are not in the same position in both architectures. Moreover, as M is a composition of metrics (Hausdorff, Hamming), it is a metric itself and so satisfies the basic properties of metric functions:

- Identity of indiscernibles: $M(X, Y) = 0 \Leftrightarrow X = Y$
- Symmetry: $M(X, Y) = M(Y, X)$
- Triangle inequality: $M(X, Y) \leq M(X, Z) + M(Y, Z)$

In Subsection 4.2.4, we show that, in addition to simply comparing two architectures, M can also be used to find the optimal number of epochs for the search process, thus actively reducing search cost.

4.2 . Experiments

In this section, we present the results of image classification and object detection experiments we conducted on various datasets including CIFAR-10 and CIFAR-100 [100], ImageNet [163], MS-COCO [113], and Cityscapes [35]. These datasets are well-known and widely used in the computer vision and pattern recognition community.

4.2.1 . Experimental Settings

All experiments were conducted using Nvidia GeForce RTX 3090 and Tesla V100 GPUs. We mostly used the same data processing, hyperparameters, and training tricks as FairDARTS [32] and DARTS [117]. We searched for 8-cell networks and used Algorithm 2 to derive 14-cell networks. We set the batch size to 128 when searching on CIFAR-10/100 and to 96 when searching on ImageNet. When starting from an existing architecture, the number of initial channels is increased to 64 when training to match the designs of the baseline architectures. However, this resulted in DARTOpti architectures having a significantly larger number of parameters than D-DARTS architectures (see Tables 4.3, 4.4 and 5.4). Hence, we reduced the number of channels to 32 while searching to save memory. Naturally, we searched for networks whose number of cells matches the number of different layers in the original architecture (e.g., 4 in ResNet50 [69]). Finally, we selected the architectural operations using FairDARTS *edge* (i.e., 2 operations maximum per edge) or *sparse* (i.e., 1 operation maximum per edge) method with a threshold of 0.85. We chose $w_{01} = 8$ and $w_{abl} = 0.5$ for the hyperparameters of total loss $\mathcal{L}_T$ (see Eq. (4.4)) as discussed in Section 4.2.2. The parsing method of DARTS is referred to as *darts* in Table 4.3 and Table 5.4.

4.2.2 . Analysis of the Ablation Loss $\mathcal{L}_{AB}$

Hyperparameter Choice

In this experiment, we made the hyperparameter weights w_{abl} and w_{01} from Eq. (4.4) vary in order to choose their optimal value w.r.t. the global loss. Thus, in Fig. 4.3 we made w_{abl} vary from 0 to 2 while keeping L_{01} deactivated (i.e., $w_{01} = 0$) in order to analyze its impact. We can observe that an optimal value seems to be attained around $w_{abl} = 0.5$, with the global loss mainly increasing when w_{abl} reaches higher or lower values.

Moreover, we varied the value of w_{01} while searching on CIFAR-100, with $w_{abl} = 0.5$ fixed, and reported the number of dominant operations (i.e., oper-

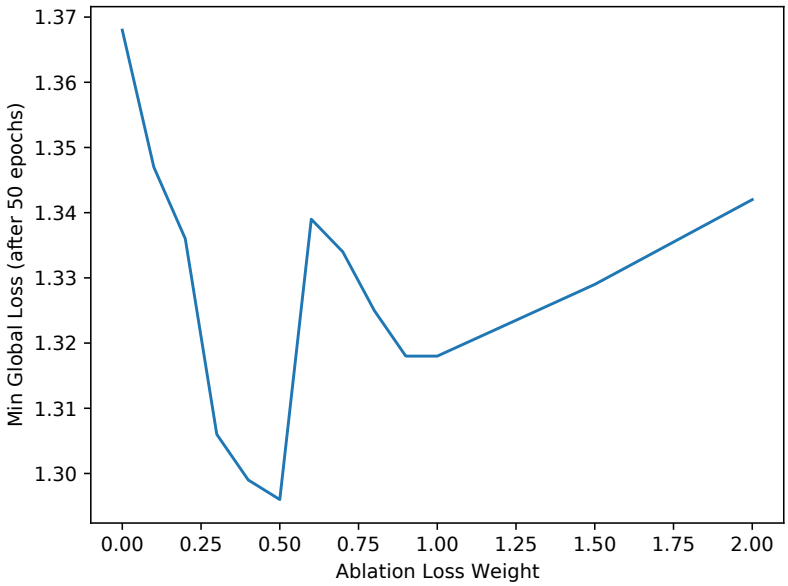


Figure 4.3: **Line plot of the minimal global loss obtained by searching for a model on CIFAR-100 [100] w.r.t. the sensitivity weight w_{abl} used for the ablation loss $\mathcal{L}_{AB}$.** We searched for 50 epochs and deactivated L_{01} (i.e., $w_{01} = 0$) to prevent interference from occurring.

ations with $\sigma(\alpha) > 0.9$). This experiment was conducted to select a relevant value for w_{01} since the one used in FairDARTS [32] ($w_{01} = 10$) is no longer valid as we altered the search process. Fig. 4.4 shows that the proportion of dominant operations steadily increases from $w_{01} = 0$ to $w_{01} = 5$ where it reaches a plateau and stabilizes. It is worth noting that for $w_{01} = 5$ and higher, nearly all sigmoid values $\sigma(\alpha)$ are either greater than 0.9 or inferior to 0.1. Finally, we chose $w_{01} = 7$ as it offers both a high number of dominant operations and an equilibrium between operations with $\sigma(\alpha) > 0.9$ and those with $\sigma(\alpha) < 0.1$.

Ablation Study

We conducted an ablation study on our proposed ablation loss $\mathcal{L}_T$ of Eq. (4.4). Specifically, we compared the performance of architectures with similar characteristics searched either with $\mathcal{L}_T$ or $\mathcal{L}_F$ (see Eq. (3.2)). Tables 4.3 and 4.4 show that $\mathcal{L}_T$-searched architectures (DD-1, DD-3, DD-4, DD-5) outperform their $\mathcal{L}_F$-searched counterparts by an average of 0.6 % across all datasets, confirming the advantage procured by this new loss function. Concretely, when considering the 14-cell *edge* parsed models evaluated on CIFAR-10, DD-3 reached a top-1 accuracy of 97.58 %, thus outperforming $\mathcal{L}_F$-searched DD-2 by 0.48 %. Moreover, it is worth noting that $\mathcal{L}_T$-searched DD-1 (8-cell model) reached a similar score as DD-2 (around 97.1 %), despite featuring significantly fewer parameters (1.7M vs. 3.3M).

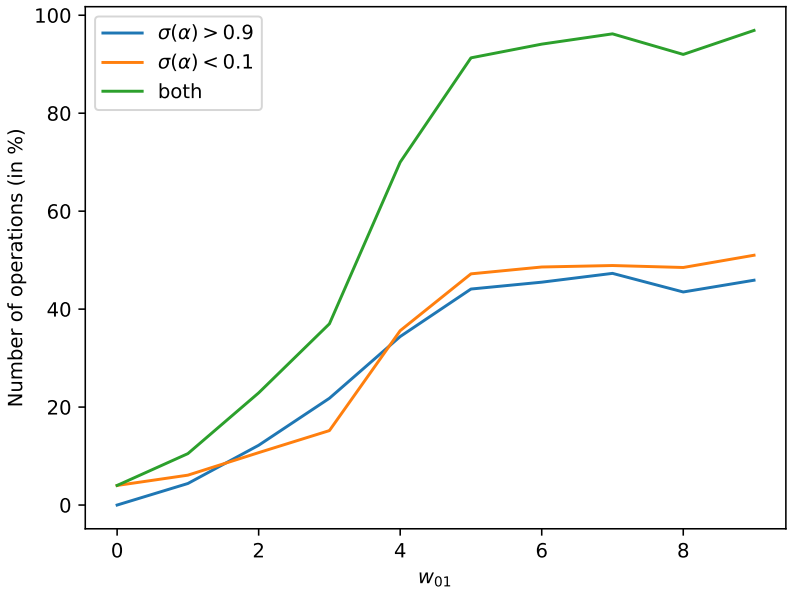


Figure 4.4: **Line plot showing the percentage of dominant operations obtained in the final architecture α while searching on CIFAR-100 w.r.t. the sensitivity weight w_{01} used for L_{01}.** We can see that the proportion of both types of operations stabilizes after $w_{01} = 5$ and reaches an equilibrium at $w_{01} = 7$.

Another important observation is that $\mathcal{L}_T$ seems to provide a larger increase in performance for CIFAR-100. For example, the gain in performance is around 1 % between the 8-cell versions of DD-6 and DD-4. However, when considering models that leveraged Algorithm 2 to increase their number of cells (e.g., DD-4 and DD-6), the performance gain is limited (e.g., around 0.1 % on CIFAR-100). This could be due to Algorithm 2 that may have a leveling effect by disturbing the cell sequence and increasing the number of model parameters.

Convergence Speed

We conducted an experiment on our D-DARTS method's search process convergence speed compared to previous baselines [117, 32]. Fig. 4.5 shows a plot of the best validation top-1 accuracy w.r.t. the number of epochs. One can notice that D-DARTS converges way faster than the other baselines. D-DARTS outperforms both DARTS and FairDARTS respectively by 9% and 14%.

4.2.3 . Memory Efficiency

When searching using $\mathcal{L}_T$ (see Eq. (4.4)), additional tensors need to be stored on GPU memory due to the computations required to obtain the marginal contribution of each cell. As a result, the memory usage with $\mathcal{L}_T$ is higher

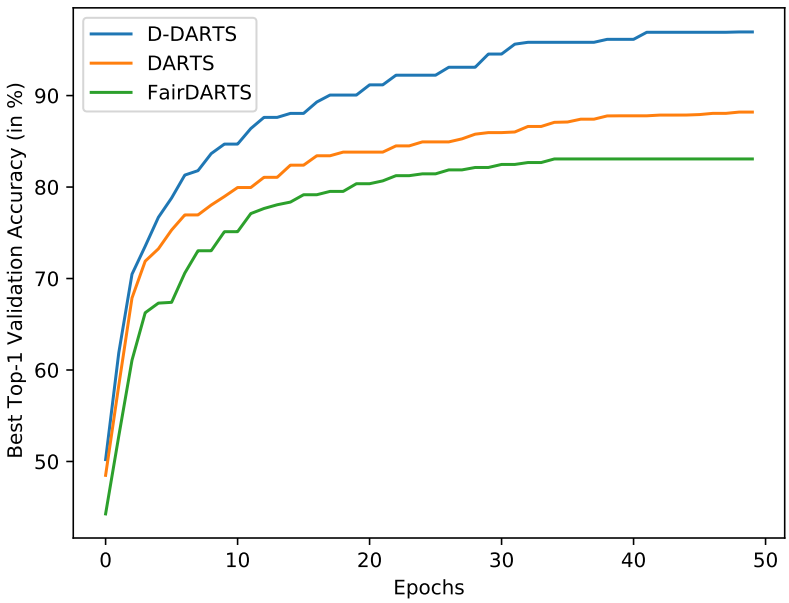


Figure 4.5: **Line plot showing the best validation top-1 accuracy while searching on CIFAR-10[100] w.r.t. the current epoch.** D-DARTS clearly outperforms both DARTS and FairDARTS by a large margin.

compared to an $\mathcal{L}_F$-based search. In fact, a $\mathcal{L}_T$-based search may increase memory consumption by up to 100%. For example, when searching with an Nvidia RTX 3090 GPU on CIFAR-100 [100] with a batch size of 72, the memory usage increases from 11100 MB with $\mathcal{L}_F$ to 21911 MB of memory with $\mathcal{L}_T$. Consequently, this can be identified as an issue of our method: we have to find means to reduce memory consumption to be able to search for deeper networks and make our method available to lower-end GPUs.

To address this, we decided to use an optimization technique: Automatic Mixed Precision (AMP). AMP automatically converts tensors to half-precision (16-bit floats) when full precision (32-bit floats) is not required. This functionality is directly available in PyTorch starting from version 1.5. Enabling AMP significantly reduced memory consumption to 13,000 MB when using $\mathcal{L}_T$. Therefore, AMP offers more flexibility to D-DARTS, allowing it to run on GPUs with less video memory and increase the batch size to achieve better memory utilization.

4.2.4 . Leveraging the Architectural Distance Metric

The architectural distance metric M (given by Eq. (4.8)) allowed us to gain insightful information on the model and search process. First, we plotted M associated with the current epoch w.r.t. the original architecture while running the optimization process. This way, we obtained plots such as Fig. 4.6, featuring M computed for ResNet18 [69] optimized on CIFAR-10 [100], and Fig.

4.7, presenting the validation accuracy reached by the same model during search. In Fig. 4.6, we can see that M quickly rises between epochs 5 and 15, then stabilizes. This would indicate that the hyperparameter of 50 search epochs (also used by DARTS [117]) is way too large and could be reduced.

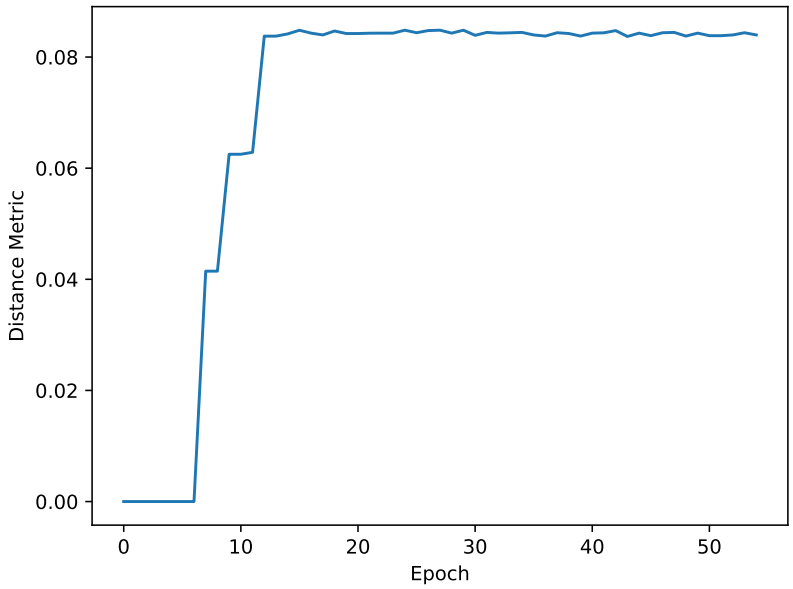


Figure 4.6: **Line plot of the distance metric between the original ResNet18 [69] architecture and the one being optimized by DARTOpti on CIFAR10 [100] according to the current epoch.** The distance quickly rises to around 0.084 DU (distance units) at epoch 15 and then stabilizes. This indicates that no additional major changes are applied to the architecture after this point.

This intuition is further confirmed by the fact that this model quickly converged to a very high top-1 accuracy (90 %) around the same epoch (15) that the distance metric reached a plateau (see Fig. 4.7). After epoch 20, the validation accuracy slowly converges toward 100 %. This is most likely due to the optimization of network weights rather than architectural modifications since M remains stable.

Following the facts presented above, we launched a new optimization of ResNet18 on CIFAR10 with the number of epochs reduced from 50 to 15. We trained the models obtained from the 50-epoch and 15-epoch optimizations for 600 epochs. As shown in Table 4.2, the performance level is marginally degraded, while still being a considerable improvement over the original ResNet18 implementation (93.75%, see Table 4.3). However, the search cost diminished significantly from 0.3 GPU-day to 0.1 GPU-day. This highlights the usefulness of our novel distance metric M.

Furthermore, based on this analysis, we designed and implemented a mechanism that automatically terminates the search process if the distance

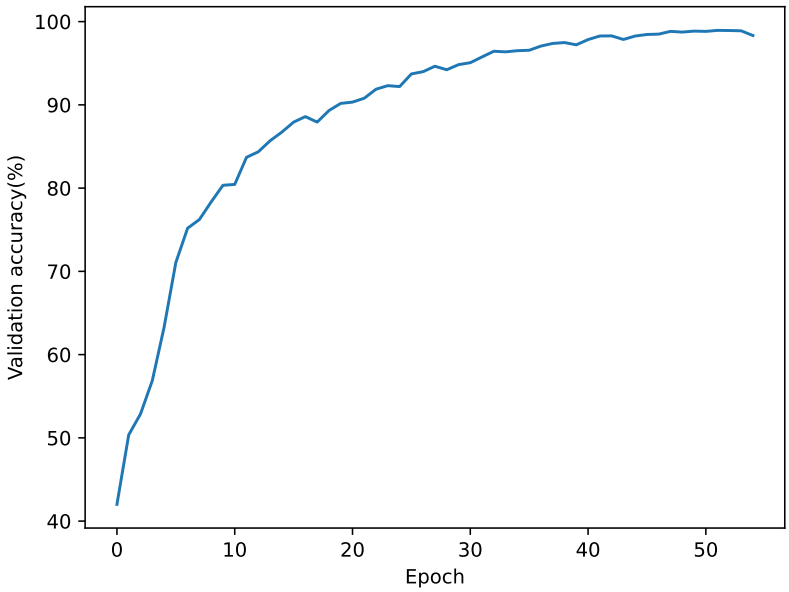


Figure 4.7: **Line plot of the validation accuracy reached at each epoch on CIFAR-10 [100] while optimizing ResNet18 [69] with DARTOpti.** The accuracy rapidly reaches a high value (around 90 %) at epoch 15 and then smoothly progresses towards 100 %.

metric remains stable for 5 consecutive epochs, starting from epoch 10 (considering the 5 pretraining epochs). Empirically, we noticed that this mechanism was triggered more often when searching on CIFAR-10 than on CIFAR-100 and never on ImageNet. The reason behind this is that ImageNet is more challenging, and the model makes better use of the 50 pre-allocated search epochs, resulting in a less stable distance metric during the search process.

4.2.5 . Searching Architectures on CIFAR

On CIFAR-10/100 [100], we searched for several 8-layer models arbitrarily dubbed DD-1, DD-2, DD-3, DD-4, DD-5 and DD-6. These models were searched using varying hyperparameters such as using loss $\mathcal{L}_T$ or $\mathcal{L}_F$, or using the *sparse* or *threshold* parsing method (presented in Section 4.2.1). We used Algorithm 2 to increase the depth to 14 layers to test how smaller architectures compete with larger ones. All results are presented in Tables 4.3 and 4.4. Overall, D-DARTS models reach competitive results in both datasets. The smaller ones, such as DD-1 or DD-5, can match the performance of previous baselines despite possessing fewer parameters (e.g., 1.7M against 2.8M for the smallest model of [32]), although the largest achieve better results (e.g., 84.15% top-1 accuracy for DD-4 on CIFAR-100). Moreover, Algorithm 2 effectively provides a performance gain in both datasets (e.g., around 0.3% for DD-4 when using 14 cells instead of 8), thus asserting its usefulness. This

Table 4.2: **Results of training for 600 epochs two DARTOpti models optimized from ResNet18 [69] on CIFAR-10 [100] with different numbers of search epochs.** Drastically reducing the number of search epochs according to the metric M only marginally degrades the performance.

Starting Architecture	Search Epochs	Train Epochs	Validation Top-1 (%)	Search Cost (GPU-days)
ResNet18	50	600	97.39	0.3
ResNet18	15	600	96.93	0.1

impact is more important on CIFAR-100 (around 2%). Hence, deeper architectures might be less relevant on simpler datasets such as CIFAR-10, where *sparse* models already achieve very high top-1 scores (greater or equal to 97%), than with more challenging datasets like CIFAR-100 or ImageNet [163]. We also compared the performance of models using FairDARTS [32] loss function $\mathcal{L}_F$ with ones using our new ablation-based loss function $\mathcal{L}_T$ to assert its effectiveness (see Section 4.2.2 for details).

4.2.6 . Searching and Transferring to ImageNet

To test our approaches on a more challenging dataset, we transferred our best models searched on CIFAR-100 [100] to ImageNet [163]. We also searched directly on ImageNet using the same training tricks and hyperparameters as the authors of DARTS [117]. Table 4.5 shows that model DD-7 (searched directly on ImageNet) reached a top-1 accuracy of 75.5 %, outperforming PC-DARTS [205] and P-DARTS [23] by 0.6 %. It is important to note that all of these approaches (and ours) use DARTS search space S while FairDARTS [32] uses its own custom search space (mainly composed of inverted bottlenecks) as they argue that S is too limited for ImageNet. Nevertheless, DD-7 still reached a near-identical score as FairDARTS-D despite using this simpler search space. In addition, the DARTOpti versions of ResNet18 and ResNet50 reached a competitive top-1 accuracy of respectively 77.0 % and 76.3 %. They critically improve on the original architectures, increasing top-1 accuracy by an average of 5.1 %. Notably, DO-2-ResNet50 achieves the same score as FBNetV2 [187] while requiring nearly a hundred times less search cost (i.e., 0.3 GPU days versus 25 GPU days) and not even having been searched directly on ImageNet but instead transferred from CIFAR-100. Finally, a notable gap (around 0.5 %) between DD-4 (transferred from CIFAR-100) and DD-7 shows that searching directly on ImageNet significantly impacts performance.

4.2.7 . Detecting objects on MS-COCO and Instance Segmentation

Table 4.3: **Comparison of models on CIFAR-10 [100].** Each reported Top-1 accuracy is the best of 4 independent runs. For previous baselines, results are the official numbers from their respective articles. The search cost is expressed in GPU days. All models have been searched on CIFAR-10 except for $\diamond$ which have been searched on CIFAR-100.

Models		Params (M)	Loss	Top-1 (%)	Layers	Cost
DARTS[117]		3.3	$\mathcal{L}_{CE}$	97.00	20	1.5
PC-DARTS[205]		3.6	$\mathcal{L}_{CE}$	97.43	20	3.8
P-DARTS[23]		3.4	$\mathcal{L}_{CE}$	97.50	20	**0.3**
FairDARTS-a[32]		2.8	$\mathcal{L}_{F}$	97.46	20	0.4
C-DARTS[212]		3.9	$\mathcal{L}_{CE}$	97.52	20	**0.3**
U-DARTS[82]		3.3	$\mathcal{L}_{CE}$	97.41	20	N.D.
DOTS[64]		3.5	$\mathcal{L}_{CE}$	97.51	20	**0.3**
DARTS-[31]		3.5	$\mathcal{L}_{CE}$	97.41	20	0.4
β-DARTS[210]		3.75	$\mathcal{L}_{CE}$	97.47	20	0.4
Ours	DD-1	**1.7**	$\mathcal{L}_{T}$	97.02	8	0.5
	DD-2	3.3	$\mathcal{L}_{F}$	97.10	**8**	0.5
	DD-3	6.55	$\mathcal{L}_{T}$	97.58	14	0.5
	DD-4$^{\diamond}$	3.9	$\mathcal{L}_{T}$	97.48	8	0.5
	DD-4$^{\diamond}$	7.6	$\mathcal{L}_{T}$	**97.75**	14	0.5

on Cityscapes

We transferred our best model trained on ImageNet (DO-2-ResNet18) to MS-COCO [113] in order to test our approach on tasks other than image classification. We used our model as the backbone of RetinaNet [112] and fine-tuned for 12 epochs, similarly to FairDARTS [32]. Table 4.6 shows that our approach reached a box AP score of 34.2 % hence outperforming DARTS- [31], FairDARTS [32] and the original ResNet18 [69]. We also performed instance segmentation on Cityscapes [35], a dataset that focuses on semantic understanding of street scenes. We used DO-2-ResNet18 as the backbone of Mask R-CNN [70] and compared it against other baselines (DARTS, FairDARTS, ResNet18). Table 4.7 features results similar to MS-COCO, with D-DARTS outperforming previous approaches. This way, the performance advantage of D-DARTS is confirmed when transferring to computer vision tasks other than image classification.

4.2.8 . Statistics on the Search Space

The metric described in Subsection 4.1.5 allows us to quantify the distance between architectures and thus can also be employed to draw statistics on the metric space. This way, we can determine the average amount of changes made by DARTOpti to the initial architectures.

Table 4.4: **Comparison of models on CIFAR-100 [100].** Each reported Top-1 accuracy is the best of 4 independent runs. For previous baselines, results are the official numbers from their respective articles. The search cost is expressed in GPU days. All models have been searched on CIFAR-100 except for $\diamond$ which have been searched on CIFAR-10.

Models		Params (M)	Loss	Top-1 (%)	Layers	Cost
DARTS[117]		3.3	$\mathcal{L}_{CE}$	82.34	20	1.5
P-DARTS[23]		3.6	$\mathcal{L}_{CE}$	84.08	20	**0.3**
FairDARTS[32]		3.5	$\mathcal{L}_{F}$	83.80	20	0.4
DOTS[64]		4.1	$\mathcal{L}_{CE}$	83.52	20	**0.3**
DARTS-[31]$^{\diamond}$		3.4	$\mathcal{L}_{CE}$	82.49	20	0.4
β-DARTS[210]$^{\diamond}$		3.83	$\mathcal{L}_{CE}$	83.48	20	0.4
Ours	DD-1$^{\diamond}$	**1.7**	$\mathcal{L}_{T}$	81.10	8	0.5
	DD-4	3.9	$\mathcal{L}_{T}$	83.86	8	0.5
	DD-4	7.6	$\mathcal{L}_{T}$	**84.15**	14	0.5
	DD-5	**1.7**	$\mathcal{L}_{T}$	81.92	8	0.5
	DD-6	3.3	$\mathcal{L}_{F}$	82.90	**8**	0.5
	DD-6	6.1	$\mathcal{L}_{F}$	84.06	14	0.5

Table 4.5: **Comparison of models on ImageNet [163].** For previous baselines, the results are the official numbers from their respective articles. The search cost is expressed in GPU days. †: Our implementation in search space S_o, results might vary from the official one.

Models		Params (M)	$+\times$ (M)	Parsing Method	Loss	Top-1 (%)	Layers	Cost	Search Space	Searched On
FBNetV2-L1[187]		8.49	326	N.A.	N.A.	**77.0**	N.A.	25	custom	ImageNet
DARTS[117]		4.7	574	*darts*	$\mathcal{L}_{CE}$	73.3	14	4	S	CIFAR-100
PC-DARTS[205]		5.3	586	*darts*	$\mathcal{L}_{CE}$	75.8	14	3.8	S	ImageNet
P-DARTS[23]		5.1	577	*darts*	$\mathcal{L}_{CE}$	75.9	14	**0.3**	S	ImageNet
FairDARTS-D[32]		4.3	440	*sparse*	$\mathcal{L}_{F}$	75.6	20	3	custom	ImageNet
DOTS[64]		5.3	596	*sparse*	$\mathcal{L}_{CE}$	76.0	20	1.3	S	ImageNet
DARTS-[31]		4.9	467	*sparse*	$\mathcal{L}_{CE}$	76.2	20	4.5	S	ImageNet
C-DARTS[212]		6.1	701	*darts*	$\mathcal{L}_{CE}$	76.3	14	1.7	S	ImageNet
U-DARTS[82]		4.9	N.D.	*darts*	$\mathcal{L}_{CE}$	73.78	14	N.D.	S	ImageNet
β-DARTS[210]		5.4	597	*sparse*	$\mathcal{L}_{CE}$	75.8	20	0.4	S	CIFAR-100
ResNet18[69]†		14.17	2720	N.A.	N.A.	69.2	4	N.A.	N.A.	N.A.
ResNet50[69]†		24.36	4715	N.A.	N.A.	73.9	4	N.A.	N.A.	N.A.
Xception[29]†		14.7	31865	N.A.	N.A.	74.1	13	N.A.	N.A.	N.A.
Ours	DD-4	7.6	617	*sparse*	$\mathcal{L}_{T}$	75.0	14	0.5	S	CIFAR-100
	DD-7	6.4	828	*edge*	$\mathcal{L}_{T}$	75.5	8	3	S	ImageNet
	DO-2-ResNet18	53.4	8619	*sparse*	$\mathcal{L}_{T}$	**77.0**	**4**	**0.3**	S_o	CIFAR-100
	DO-2-ResNet50	73.23	10029	*sparse*	$\mathcal{L}_{T}$	76.3	**4**	**0.3**	S_o	CIFAR-100

Fig. 4.8 shows the distribution of changes between ResNet-based archi-

Table 4.6: **Comparison of backbone models for RetinaNet [112] on MS-COCO [113].** Compared to previous works, DARTOpti consistently obtains better values for nearly all metrics.

Models	AP (%)	AP_{50} (%)	AP_{75}(%)	AP_s(%)	AP_m(%)	AP_l(%)
FairDARTS[32]	31.9	51.9	33.0	17.4	35.3	43.0
DARTS-[31]	32.5	**52.8**	34.1	18.0	36.1	43.4
ResNet18[69]	31.7	49.6	33.4	16.2	34.2	43.0
DO-2-ResNet18 (Ours)	**34.2**	52.1	**36.6**	**19.1**	**38.3**	**45.3**

Table 4.7: **Comparison of backbone models for Mask R-CNN [70] on Cityscapes [35].** Compared to previous works, DARTOpti consistently obtains better values for nearly all metrics.

Models	AP(%)	AP_{50}(%)	AP_{75}(%)	AP_s(%)	AP_m(%)	AP_l(%)
FairDARTS[32]	41.1	69.3	N.A.	18.9	42.4	61.8
DARTS-[31]	41.7	**70.4**	N.A.	19.5	43.4	62.3
ResNet18[69]	40.9	66.2	N.A.	17.6	41.1	61.8
DO-2-ResNet18 (Ours)	**44.0**	69.6	N.A.	**20.2**	**46.0**	**65.1**

tectures. For instance, the distance between *ResNet18C10* and *ResNet50C10* is around 0.17 distance unit. The distances between architectures are relatively uniform, with the greatest difference (0.2 distance units) observed between *ResNet18C100* and *ResNet50C10*. This finding is logical since they evolved from two different initial architectures (ResNet18 and ResNet50) and were trained on two different datasets (CIFAR10 and CIFAR100). Additionally, ResNet18 and ResNet50 are very close to each other as they both only consist of a few operations per cell (compared to DARTOpti architectures), and these operations are very similar, mainly comprising *skip-connections* and 3×3 convolutions.

These observations indicate that both the starting architecture and the dataset used during the search process have an impact on the composition of the final architecture.

4.3 . Discussion and Conclusion

In this chapter, we tackled challenge **(IV)** by proposing a novel paradigm for DARTS [117] with individualized cells that significantly increases the size of its search space. To accompany this approach, we also proposed a novel cell-level loss $\mathcal{L}_T$, an algorithm that is able to automatically derive deeper architectures, and a distance metric M that is able to assess the closeness of two architectures in search space S.

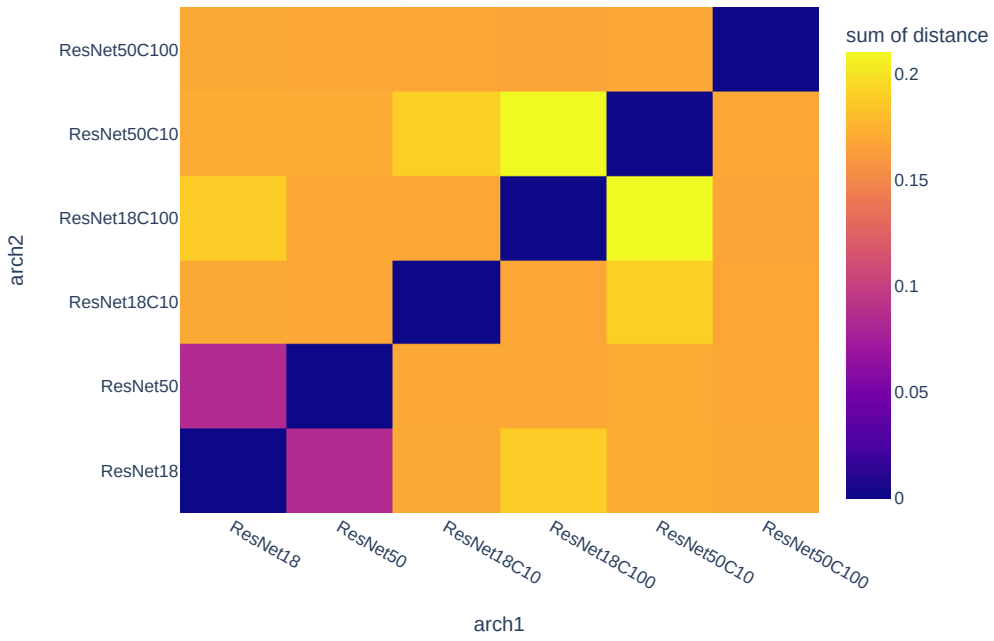


Figure 4.8: **Heatmap representing the distances between the different architectures obtained from ResNet [69] on CIFAR [100] datasets.** We used the *edge* parsing method. The models using different starting points (ResNet18/ResNet50) and searched on different datasets (C10/C100) are logically the farthest apart from each other.

However, in Section 4.2, we showed that our proposed concepts perform well but are not exempt from limitations. Increasing the search space size to such an extent (e.g., 10^{72} with 8 cells) makes the optimization process significantly more challenging. Nevertheless, as shown in Fig. 4.5, D-DARTS outperforms both DARTS and FairDARTS during the search phase, indicating that expanding the search space provides benefits that outweigh the optimization difficulty. To further address this optimization challenge, future work could explore enhancing cell optimizers.

Moreover, we show that despite using a data-dependent search process, following the practice of all previous DARTS-based works [117, 32, 31, 210], that should decrease robustness w.r.t. other datasets, our D-DARTS models still demonstrate generalizability to other tasks. In fact, transfer learning experiments show that models searched on the small CIFAR-100 dataset still achieve competitive performance on the large-scale ImageNet dataset (see Table 4.5).

Furthermore, we proved that D-DARTS could be adapted into DARTOpti to successfully optimize top-performing handcrafted architectures such as ResNet50 [69], resulting in a significant gain in performance (i.e., a 3.9 % average increase in top-1 accuracy across all datasets). However, this approach also has limitations, as the optimization process becomes more challenging

for architectures with many cells (e.g., Xception [29] with 13 cells). This leads to an increased search cost and limited performance gains, as the standard 50 search epochs may not be enough for architectures of that size.

Finally, by combining the *sparse* parsing method with our distributed design, we were able to discover unprecedentedly small architectures (around 1.7 million parameters for the tiniest) that still yield competitive results. Additionally, when using the *edge* parsing method, we can search for larger models that achieve state-of-the-art results. This demonstrates the flexibility and utility of our approach.

However, as seen in Chapter 3, most DNAS works focus on improving CNNs and tackling image classification or object recognition tasks in the classic supervised learning paradigm. In the next chapter, we show that DNAS can be adapted to improve other types of neural networks architectures (e.g., Multi-Layer Perceptrons) in other learning paradigms such as Self-Supervised Learning (SSL).

Algorithm 1 Algorithm describing the differentiable neural architecture search process of D-DARTS

Require: List: C, list of cells each containing architectural weights α
Require: List: O, list of operation
Require: List: D_t, train dataset
Require: List: D_v, validation dataset
Require: Object: $model$, supernet
Require: Object: opt, supernet optimizer
Require: List: $search\_opts$, list containing the search optimizers of each individual cells
Require: Integer: E, number of epochs
Require: Float: w_{0-1}, weight for $\mathcal{L}_{0-1}$ in $\mathcal{L}_T$
Require: Float: w_{AB}, weight for $\mathcal{L}_{AB}$ in $\mathcal{L}_T$
Require: Float: σ_t, sigmoid threshold for architecture derivation

for e in [0, E[**do**
  for (x, t) in D_t **do**
    $x_v, t_v \leftarrow$ `next_batch`(D_v)
    $p_v \leftarrow model(x_v)$
    $\mathcal{L}_{CE} \leftarrow$ `cross_entropy`(p_v, t_v)
    $M_C \leftarrow$ `empty_list()`
    for C_i in C **do**
      C_i.`deactivate`()
      $p_v \leftarrow model(x_v)$
      $\mathcal{L}^{(}C \setminus \{C_i\})_{CE} \leftarrow$ `cross_entropy`(p_v, t_v)
      C_i.`activate`()
      $M_C^i \leftarrow \mathcal{L}_{CE} - \mathcal{L}^{(}C \setminus \{C_i\})_{CE}$
    end for
    for o in $search\_opts$ **do**
      $\mathcal{L}_{AB}^i \leftarrow \frac{M_C^i - \overline{M_C}}{\overline{M_C}}$
      $\mathcal{L}_T^i \leftarrow \mathcal{L}_{CE} + w_{AB}\mathcal{L}_{AB}^i + w_{0-1}\mathcal{L}_{0-1}^i$
      o.`optimization_step`$(\mathcal{L}_T^i, C_i.weights)$
    end for
    $p \leftarrow model(x)$
    $\mathcal{L}_{CE} \leftarrow$ `cross_entropy`(p, t)
    opt.`optimization_step`$(\mathcal{L}_{CE}, model.weights)$
  end for
end for
$A \leftarrow$ `empty_list()`
for C_i in C **do**
  $C_f \leftarrow$ `empty_list()`
  $n \leftarrow |C_i|$
  for i in $[0, n[$ **do**
    $op \leftarrow$ `argmax`$(\sigma(C.weights[i]))$
    if $op \geq \sigma_t$ **then**
      `append`$(O$.`find`$(op), C_f)$
    end if
  end for
  `append`(C_f, A)
end for
return A

Algorithm 2 Algorithm describing the larger architecture derivation process for D-DARTS

Require: List: C, list of searched cells
Require: Integer: n, desired number of cells
Ensure: List: C_f, list of cells that compose the derived architecture

$C_f \leftarrow$ empty_list()
$m \leftarrow$ euclidean_division$(|C|, 3)$
$m2 \leftarrow$ euclidean_division$(2 \times |C|, 3)$
for i in $[0, n]$ **do**
  if $n > |C|$ **then**
    if $i <$ euclidean_division$(n, 3)$ **then**
      $c \leftarrow$ modulo(i, m)
    else if $i =$ euclidean_division$(n, 3)$ **then**
      $c \leftarrow m$
    else if $i >$ euclidean_division$(n, 3)$ **and** $i <$ euclidean_division$(2 \times n, 3)$ **then**
      $c \leftarrow$ modulo$(i, m2 - 1 - m) + m + 1$
    else if $i =$ euclidean_division$(2 \times n, 3)$ **then**
      $c \leftarrow m2$
    else
      $c \leftarrow$ modulo$(i, |C| - 1 - m2) + m2 + 1$
    end if
  else
    $c \leftarrow i$
  end if
  append(c, C_f)
end for

Algorithm 3 Algorithm describing the Hamming distance

Require: List: U, first vector to compare
Require: List: V, second vector to compare
Require: List: W, weights associated with vector indices

$n \leftarrow |U|$
$dist \leftarrow$ empty_list()
for i in $[0, n]$ **do**
  if $U[i]! = V[i]$ **then**
    $dist[i] \leftarrow W[i]$
  else
    $dist[i] \leftarrow 0$
  end if
end for
return $mean(dist)$

5 - Tackling Self-Supervised Learning: Efficient Representation Learning using Neural Architecture Search for Siamese Networks

Self-Supervised Learning (SSL) has experienced rapid growth in the past few years. SSL aims to make DL models learn strong representations from unlabeled data. This is especially useful when considering that manual data labeling is often a costly and laborious process. One of the most common approaches to self-supervised visual representation learning is Siamese networks [14]. Siamese networks consist of two weight-sharing branches (referred to as "twins") applied to two or more inputs. The output feature vectors of the two branches are compared to compute a loss (e.g., a “contrastive” loss). In the case of SSL, the inputs are usually different augmentations of the same image, with the objective of maximizing the similarity between the output feature vectors of the two branches of the Siamese networks [20, 24, 25].

This chapter explores the application of differentiable NAS to discover encoder (projector) and predictor architectures (i.e., Multi-Layer Perceptrons) that enable backbone Convolutional Neural Networks (CNNs) to efficiently learn strong representations from unlabeled data in a contrastive SSL context. To the extent of our knowledge, this is the first time that NAS has been applied to enhance the architecture of Siamese networks (excluding the backbone). We improved the design of several Siamese network frameworks such as SimSiam [24], SimCLR [20], or MoCo [25] by using an encoder-predictor pair discovered by a meta-learner inspired by DARTS [117]. We dubbed our approach NASiam (“Neural Architecture Search for Siamese Networks”). Our experiments demonstrate that NASiam achieves competitive results on small-scale (CIFAR-10, CIFAR-100 [100], INRIA Holidays [92]) and large-scale (ImageNet [163]) datasets.

Section 5.1 highlights our following contributions: A novel way to design encoder/predictor pairs for Siamese networks using DNAS and a novel search space specifically designed for the Multi-Layer Perceptron (MLP) heads of encoder/predictor pairs. We also present how NASiam can be adapted to perform content-based image retrieval, a task where Siamese networks present an advantage due to their innate ability to assess the similarity between two inputs. The remainder of the chapter is structured as follows: Section 5.2 presents the results of various computer vision experiments, and 5.3 provides a discussion on the composition of the discovered encoder/predictor pair architectures and brings a conclusion to our work while offering some insights for future work.

5.1 . Proposed Approach

The proposed NASiam approach focuses on searching for the MLP components of the encoder/predictor pair and involves creating an original search space specifically designed for contrastive learning with Siamese networks. Additionally, we adapted NASiam to be applicable to content-based image retrieval.

5.1.1 . Searching for an Encoder/Predictor Pair

First, we focused on SimSiam [24] as a simple baseline to build our approach upon. SimSiam uses a Siamese architecture consisting of an encoder f and a predictor h. The encoder f is composed of a baseline CNN (e.g., ResNet50 [69]) and of a projector head, which is a three-layer MLP duplicated on twin branches that take different augmentations of the same image as input. A two-layer MLP h is then added on top of one of the branches to act as the predictor head. The discrepancy between the output feature vectors of the two branches is computed using the following contrastive loss:

$$\mathcal{L}_C = \frac{1}{2}(\mathcal{D}(p_1, \texttt{stopgrad}(z_2)) + \mathcal{D}(p_2, \texttt{stopgrad}(z_1)), \tag{5.1}$$

where $z_1 = f(x_1)$, $z_2 = f(x_2)$, $p_1 = h(z_1)$, $p_2 = h(z_2)$ for input images x_1 and x_2, `stopgrad` is a mechanism that stops gradient backpropagation (i.e., the argument inside `stopgrad` is detached from the gradient computation), and $\mathcal{D}$ is the negative cosine similarity defined as follows:

$$\mathcal{D}(p, z) = -\frac{p}{||p||_2}.\frac{z}{||z||_2}, \tag{5.2}$$

where $||.||_2$ is the l_2 norm.

In our proposed approach, we retained most of the global structure of the baseline Siamese framework. However, we used a DNAS method to search for an encoder projector head architecture up to n layers and a predictor architecture up to m layers. Specifically, we considered a set $O = \{o_1, ..., o_K\}$ of candidate operations. We searched for two cells (see Section 2.7), denoted C_e and C_p, for the encoder and decoder respectively. Unlike DARTS [117], each cell is structured as a linear sequence of layers where each layer is a mixed output of $|O| = K$ operations. Each operation o in each layer i is weighted by a parameter α_i^o. The sets of architectural parameters for C_e and C_p are denoted α_e and α_p respectively. Similarly to Eq. 2.7, operation values in each layer are discretized as follows:

$$\overline{o}_i(x) = \sum_{k=1}^{K} \sigma_{SM}(\alpha_i^k) o_k(x), \tag{5.3}$$

where $\overline{o}_i$ is the mixed operation of layer i, α_i^k is the architectural weight assigned to $o_k \in O$ for layer i, and σ_{SM} denotes the *softmax* operation. The

supernet encompassing f and h is trained on a portion of a dataset while C_e and C_p are simultaneously searched on another portion of the same dataset. Therefore, NASiam aims to solve a bi-level optimization problem formulated as follows:

$$\begin{aligned} &\min_{\alpha_e,\alpha_p} \mathcal{L}_{val}(w^*(\alpha_e,\alpha_p),\alpha_e,\alpha_p), \\ \text{s.t.} &w^*(\alpha_e,\alpha_p) = \operatorname*{argmin}_{w} \mathcal{L}_{train}(w,\alpha_e,\alpha_p), \end{aligned} \tag{5.4}$$

where w denotes the weights of the supernet, $\mathcal{L}_{train}(w,\alpha_e,\alpha_p) = \mathcal{L}(w,\alpha_e,\alpha_p)$ is the training loss, and $\mathcal{L}_{val}(w^*,\alpha_e,\alpha_p) = \mathcal{L}(w^*,\alpha_e,\alpha_p)$ is the validation loss.

Once the search phase was completed, we selected the best-performing operation for each layer i of each cell, based on the discretized weights α_e and α_p, to form the encoder/predictor architecture genotype G. The entire DNAS process is detailed in Algorithm 4.

Our approach, dubbed NASiam (Neural Architecture Search for Siamese Networks), is summarized in Fig. 5.1. In addition to SimSiam, we experimented with NASiam on other Siamese frameworks such as SimCLR[20] and MoCo V2 [25]. However, those frameworks do not rely on a predictor. Hence, in those cases, we only performed NAS for the MLP projector head of the encoder (i.e., only searching for cell C_e).

In Section 5.2, we demonstrate that NASiam can consistently improve the performance of popular siamese frameworks (SimSiam, SimCLR, and MoCoV2) in both small-scale (CIFAR-10 and CIFAR-100 [100]) and large-scale (ImageNet [163]) image classification datasets, as well as for content-based image retrieval on INRIA Holidays [92].

5.1.2 . Crafting a Contrastive Learning-Specific Search Space

In order to support our NASiam approach (see Section 5.1.1), an original search space S has been created. This search space is specifically designed for MLPs.

The design of search space S is important for the success of the NASiam approach, as it defines the set of possible solutions that the algorithm can consider. The originality of the search space lies in the fact that it is tailored for MLPs, meaning that it takes into account the specific characteristics and constraints of this type of network. By crafting an original search space for MLPs, the NASiam approach can effectively find the optimal components for the encoder/predictor pair in a Siamese neural network for contrastive learning.

In our work, S comprises the following 7 operation blocks: (1) `linear` + `batch_norm` + `ReLU`, (2) `linear` + `batch_norm` + `Hardswish`, (3) `linear` + `batch_norm` + `SiLU`, (4) `linear` + `batch_norm` + `ELU`, (5) `max_pool_3x3` (1-dimensional) + `batch_norm`, (6) `avg_pool_3x3` (1-dimensional) + `batch_norm`, and (7) `Identity` (*skip connection*).

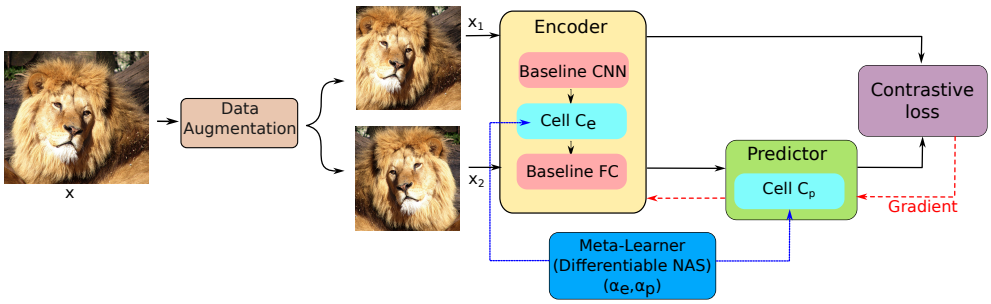


Figure 5.1: **Layout of the NASiam architecture.** Siamese network encoder/predictor (projection MLPs) architectures are searched using differentiable NAS wrapped around a Siamese framework such as SimSiam [20], which is the baseline used in the present figure. First, an input image x is augmented to produce two variations x_1 and x_2. Each of these two inputs is then fed into one of the two branches of the Siamese network. While both x_1 and x_2 go through an encoder equipped with an MLP projection head, x_2 is further processed by an MLP predictor. Finally, a negative cosine contrastive loss is computed and backpropagated to minimize the similarity between the two branches' output feature maps. Both the encoder and the decoder contain cells (i.e., C_e and C_p, respectively) that are designed using a differentiable NAS approach. Architectural parameters for C_e and C_p are denoted α_e and α_p respectively.

Therefore, the search space S includes several types of fully connected layers, each with a unique activation function. The activation function determines the output of a neuron in the neural network given its input. Different activation functions can lead to different network behaviors, positively or negatively impacting performance. Note that the inclusion of various activation functions in the search space is motivated by the desire to increase diversity among the candidate architectures and explore alternatives to the classic ReLU function (e.g., Hardswish [79], or SiLU [73]). To that end, it is also possible to mix different activation functions according to the type of network (i.e., projector or predictor) and the location of the activation function inside that network. In contrast, previous baselines [20, 24, 25] only relied on a single activation (ReLU) for both networks regardless of their respective architectures.

While unconventional, including pooling layers in the search space is helpful as we show in Section 5.2 that they can help prevent collapsing. Moreover, the authors of SimSiam [24] indicated that an insufficient or excessive number of Batch Normalization (BN) layers could cause the model to underperform severely or become unstable. They empirically demonstrate that the optimal setting for SimSiam is to place BNs after every layer except for the predictor's output layer. Hence, we follow this assertion by adding BNs after every `linear` and `pooling` operation except for the predictor's final layer. Finally, we also

included the `identity` operation in the search space so that the search algorithm can modulate the number of layers in the architecture. This way, we can indicate a maximum number of layers n, and the search process can craft an architecture of size $m < n$ by "skipping" layers.

5.1.3 . Adapting the Siamese contrastive learning framework to perform content-based image retrieval

To evaluate our approach on a content-based image retrieval task (see Section 5.2), we replaced the contrastive loss $\mathcal{L}_C$ used for pretraining the model (see Eq. 5.1) with the simple cosine similarity function CS defined as follows:

$$CS(x_1, x_2) = \frac{x_1 \cdot x_2}{max(||x_1||_2 \cdot ||x_1||_2, \epsilon)}, \tag{5.5}$$

where $\cdot$ represents the dot product between two vectors, $||.||_2$ is the ℓ_2 norm, and ϵ is a small positive number (e.g., 10^{-8}) to avoid division by zero.

This switch is motivated by the fact that we no longer perform contrastive learning and hence do not need a contrastive loss anymore. Furthermore, cosine similarity is a standard metric often encountered in content-based image retrieval literature[182, 188].

When performing the evaluation task, we computed the cosine similarity between the output feature vectors of every query image. We selected only the images whose similarity value was above a threshold τ we arbitrarily set and ranked them accordingly.

5.2 . Experiments

In this section, we conducted experiments to validate the effectiveness of the proposed NASiam approach. First, we conducted image classification experiments on small-scale (CIFAR-10, CIFAR-100) and large-scale (ImageNet) datasets [163]. Furthermore, we conducted content-based image retrieval experiments on INRIA Holidays [92]. Finally, we conducted object detection and instance segmentation experiments on the MS COCO dataset [113]. These experiments allowed us to assess the performance of NASiam on various computer vision datasets and compare it to other state-of-the-art approaches.

Alongside the main experiments, we conducted ablation studies to further validate the effectiveness of the NASiam method. These studies focused on evaluating the importance of pooling layers and the impact of data augmentation on the performance of NASiam.

5.2.1 . Experimental Settings

We used Nvidia RTX 3090 and Tesla V100 GPUs to conduct our experiments. We searched for predictor/encoder pairs for 100 epochs on CIFAR-10, and CIFAR-100 [100] using the `SGD` optimizer with $lr = 0.06$, $wd = 5e-4$, and a

batch size of 512. We set a maximum of 6 layers for the encoder. For baseline Siamese frameworks relying on a predictor, we searched for a 4-layer predictor architecture. The whole search process with these settings takes around 2.3 hours on a single GPU. We did not search on the full ImageNet [163] dataset as it is prohibitively expensive (i.e., it takes around 12 days on a single GPU). Instead, we transferred our best CIFAR-searched architecture to ImageNet. We kept the same settings for the pre-training and linear classification phases as SimSiam [24].

5.2.2 . Ablation Study on the Importance of Pooling Layers

We conducted an ablation study on the importance of including pooling layers in our novel space search S (see Section 5.1.2). To this end, we simply removed `max_pool_3x3` and `avg_pool_3x3` from S to form S'. When comparing the results in Table 5.1, we can observe that when searching on S' rather than S, the validation top-1 accuracy of NASiam drops significantly (by around 3 %). Moreover, when analyzing the genotypes searched on CIFAR-10 and CIFAR-100 using S, it appears that the predictor architectures always contain pooling layers (making up to 40 % of the total architecture). Additionally, Fig. 5.2 shows that the contrastive learning model achieved better similarity and faster convergence when searched on S rather than S'. Thus, these findings highlight the critical role of pooling layers in ensuring high performance and preventing collapse, especially concerning the encoder architecture.

Table 5.1: **Results on CIFAR-10 linear classification of two NASiam models using search space S and S' respectively.** Both models were pre-trained for 100 epochs. The baseline framework is SimSiam with a ResNet18 backbone.

Search Space	Search Epochs	Pre-Train Epochs	Validation Top-1 (%)	Validation Top-5
S	50	100	**66.6**	91.3
S'	50	100	63.7	86.1

5.2.3 . Incidence of Data Augmentations on the NAS process

In self-supervised learning, data augmentations are paramount to prevent the model from overfitting and the contrastive loss (see Eq. 5.1) from saturating to -1. In contrast, DNAS methods [117, 32, 76] scarcely employ data augmentation as they only train the supernet for a small number of epochs (e.g., 50). Thus, a legitimate question is how the strong data augmentation policy used in SSL frameworks can interfere with the differentiable search process. we conducted a search for two different SimSiam [24] models: one with the data augmentation policy activated and the other without it, on CIFAR-10. Then, we compared the resulting architectures.

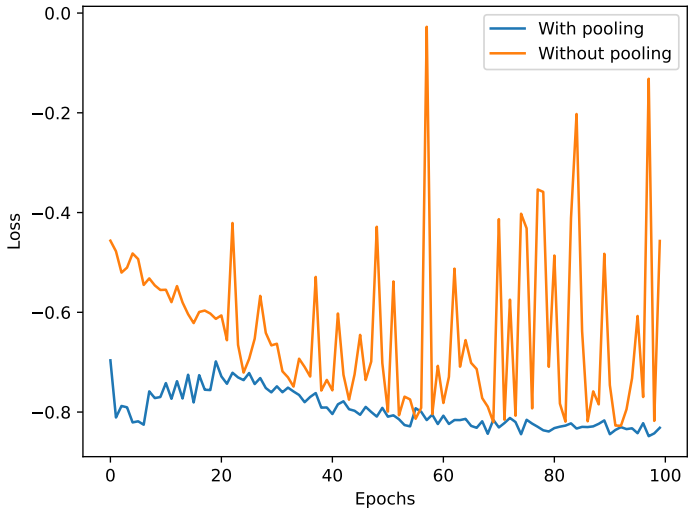


Figure 5.2: **Plot of the negative cosine contrastive loss while pretraining two NASiam models on CIFAR-10.** The baseline framework is SimSiam with a ResNet18 backbone. The two models are searched on search spaces S (blue line) and S' (red line) respectively. The model searched on S achieves better similarity, thus making the relevance of pooling layers clear.

Table 5.2 shows that deactivating the data augmentation policy leads to a degenerated architecture with a dominance of `skip connections` (50 % of the architecture) associated with performance collapse. Furthermore, Fig. 5.3 shows that, during the search phase, the similarity loss converges significantly faster towards -1, thus presenting a collapsing behavior. This observation correlates with the architectural collapse described in numerous DNAS studies [32, 217, 210]. This collapsing behavior is akin to overfitting for NAS and is caused by the high prominence of *skip connections* due to their unfair advantage compared to parametric operations. Thus, data augmentations clearly positively impact the differentiable search process and should not be deactivated, unlike in supervised learning.

Table 5.2: **Results on CIFAR-10 linear classification of two NASiam models using either SimSiam data augmentation policy or no data augmentation.** Both models were pre-trained for 800 epochs. The baseline framework is SimSiam with a ResNet18 backbone.

Data Augmentations	Search Epochs	Pre-Train Epochs	Validation Top-1 (%)	% of Skip Connections
Yes	100	800	91.2	20
No	100	800	10.0	50

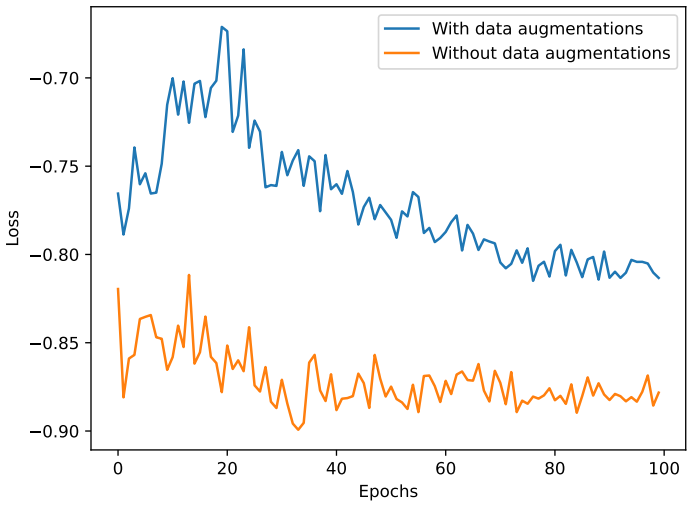


Figure 5.3: **Plot of the negative cosine contrastive loss while pretraining two NASiam models on CIFAR-10.** The baseline framework is SimSiam with a ResNet18 backbone. The two models are searched with and without data augmentation respectively.

5.2.4 . Preliminary Results on CIFAR

To quickly assess the behavior of our novel approach NASiam, we first conducted preliminary experiments on small-scale CIFAR datasets [100]. We searched NASiam architectures for 100 epochs on CIFAR-10 and CIFAR-100 using the CIFAR version of ResNet18 [69] as the encoder backbone. Then, we performed unsupervised pretraining for 800 epochs with a cosine annealing schedule before training a linear classifier using frozen features for 100 epochs. In these settings, NASiam overperforms SimSiam by 1.4 % and 0.4 % on CIFAR-10 and CIFAR-100 respectively (see Table 5.3). In addition, Fig. 5.4 shows that NASiam can achieve better similarity than SimSiam without saturating the contrastive loss to -1 (i.e., avoiding a "collapsing" behavior). Furthermore, results were also positive when using alternative Siamese frameworks, with NASiam overperforming both MoCo V2 [25] and SimCLR [20].

5.2.5 . Results on ImageNet

We conducted image classification experiments on ImageNet [163] as a standard practice to evaluate the performance of our novel approach on large-scale datasets. As stated in Section 5.2.1, we transferred our best CIFAR architecture instead of searching directly on ImageNet to save computational resources. Then, we performed unsupervised pretraining on ImageNet for 100 epochs before training a linear classifier with frozen features for 100 epochs. The results are presented in detail in Table 5.4. As with CIFAR (see 5.2.4), the results of the experiments on ImageNet showed that the NASiam consistently outperforms the baseline frameworks in terms of linear classification results.

Table 5.3: **Results of pre-training for 800 epochs on CIFAR-10 and CIFAR-100 linear classification with SGD.** The backbone is the CIFAR version of ResNet18. †: Result obtained by running the official implementation with the hyperparameters suggested by the authors for CIFAR-10.

Model		Batch Size	Pre-Train Epochs	Train Epochs	C10 Validation Top-1 (%)	C100 Validation Top-1 (%)
MoCo V2 [25]†		256	800	100	89.8	62.9
SimCLR [20]†		256	800	100	91.1	63.6
SimSiam [24]†		256	800	100	89.5	63.7
	NASiam (SimSiam)	256	800	100	**91.2**	**64.1**
Ours	NASiam (MoCo V2)	256	800	100	**90.4**	**65.0**
	NASiam (SimCLR)	256	800	100	**92.1**	**68.9**

This indicates that NASiam is a useful and effective approach for improving the performance of Siamese networks in image classification tasks.

Table 5.4: **Results of training for 100 epochs on ImageNet linear classification with SGD.** The backbone is ResNet50. Models were pre-trained for 100 epochs on ImageNet. †: Score obtained by the authors of SimSiam by using an improved version of the model.

Model		Batch Size	Pre-Train Epochs	Train Epochs	Validation Top-1 (%)
MoCo V2 [25]†		256	100	100	67.4
BYOL [63]†		4096	100	100	66.5
SimCLR [20]†		4096	100	100	66.5
SimSiam [24]		256	100	100	67.1
	NASiam (SimSiam)	256	100	100	**67.4**
Ours	NASiam (MoCo V2)	256	100	100	**67.4**
	NASiam (SimCLR)	4096	100	100	**67.2**

5.2.6 . Content-based image search evaluation on Holidays

We evaluated our approach on content-based image retrieval using the Holidays dataset [92]. We used SimCLR as the backbone Siamese framework and adapted it according to what is presented in Section 5.1.3. The results are reported in Table 5.5. The proposed method outperforms the baseline SimCLR method by around 5 % AP_m, hence validating the relevance of our approach. Another noticeable fact is that using a deeper projector head does not affect the query time, which remains around 1 minute.

Furthermore, the proposed method also surpasses the performance of Jegou et al.'s method [92]. It is worth noting that this previous work uses a highly sophisticated method for matching descriptors, which involves deriving a more precise representation using a suboptimal approach based on

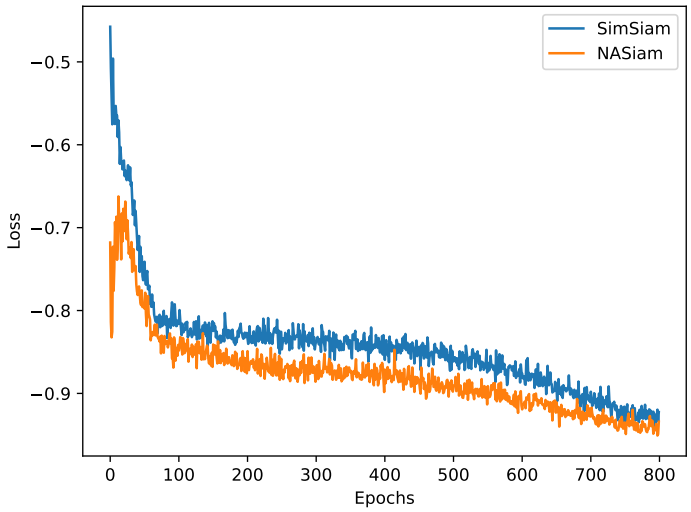


Figure 5.4: **Plot of the negative cosine contrastive loss when pretraining SimSiam and NASiam for 800 epochs on CIFAR-10.** NASiam converges faster without collapsing and achieves better similarity than SimSiam.

Hamming embedding and weak geometric consistency constraints. This technique could also be investigated to further enhance the performance of the proposed method. However, in this study, we only demonstrated the potential of our searched Siamese network without any postprocessing (e.g., fine-tuning or computing descriptors).

Model	AP_m (%)	Query Time (s)	Inference Time per Image (ms)
Jegou et al.[92]	75.07	N.D.	N.D.
SimCLR[20]	70.1	73	6.3
Ours	**75.4**	73	6.9

Table 5.5: **Results of content-based image retrieval on the INRIA Holidays dataset[92].** SimSiam performs better than both SimCLR and the baseline method of Jegou et al. [92] that uses precomputed descriptors.

5.2.7 . Object Detection and Instance Segmentation Results on COCO

To further evaluate the performance of the NASiam method, we conducted additional experiments on Object Detection and Instance Segmentation using the Microsoft Common Objects in Context (MS COCO) dataset [113]. The results of these experiments, reported in Table 5.6, provide additional insight

into the performance and effectiveness of the NASiam method for these tasks and allow for a comparison to other state-of-the-art methods. Table 5.6 displays the results of transferring our NASiam models pretrained on ImageNet [163] to MS COCO. These results show that NASiam consistently outperforms handcrafted SSL architectures in both object detection and instance segmentation tasks. Combined with the evaluation on content-based image retrieval in Section 5.2.6, this demonstrates the ability of NASiam to generalize to computer vision tasks beyond image classification and suggests that it is a promising approach for improving the performance of Siamese networks in a variety of computer vision tasks.

Table 5.6: **Comparison of backbone models for MaskRCNN [70] on COCO [113] using a 1x schedule and ResNet50 [69] as the baseline CNN.** All models are pretrained for 200 epochs on ImageNet, finetuned for 12 epochs on COCO 2017 train set, and evaluated on COCO 2017 val set.

Models	AP_{50} (%)	AP (%)	AP_{75} (%)	AP^{mask}_{50} (%)	AP^{mask} (%)	AP^{mask}_{75} (%)
ImageNet supervised	58.2	38.2	41.2	54.7	33.3	35.2
SimCLR	57.7	37.9	40.9	54.6	33.3	35.3
SimSiam	57.5	37.9	40.9	54.2	33.2	35.2
MoCo V2	58.8	39.2	42.5	55.5	34.3	36.6
BYOL	57.8	37.9	40.9	54.3	33.2	35.0
NASiam (SimSiam)	**58.6**	**39.0**	**42.1**	**55.2**	**34.1**	**36.3**

5.2.8 . Analysis of the composition of NAS-discovered architectures

Some facts are noteworthy when comparing encoder/predictor architectures discovered on CIFAR-10 by our novel NASiam approach (see Section 5.1.1) with those of SimSiam [24].

First, in Fig. 5.5, we can see that both ResNet50-based and ResNet18-based [100] NASiam architectures are significantly deeper than the original SimSiam architecture. Furthermore, a remarkable fact is that the standard `ReLU` activation function is in minority in the discovered architectures (and even completely absent from the ResNet50-based one). Instead, a mix of different activation functions is preferred, with `SiLU` and `Hardswish` having a high prominence. Thus, this may indicate that `ReLU`, despite its popularity, is not the optimal activation function for performing contrastive learning. In addition, the optimizer always selected at least one `AvgPool3x3+BN` layer to be part of the predictor architecture, hence validating the relevance of including pooling layers in the search space (as already highlighted in Section 5.2.2).

Furthermore, when comparing both NAS-discovered architectures, we can observe that the ResNet50-based one possesses a deeper encoder than the ResNet18-based architecture (i.e., 6 vs. 4 layers), with additional `Linear+BN+Swish`

and `Linear+BN+SiLU` blocks. However, the two predictor architectures retain the same depth and a similar composition. This is coherent with the recommendations of the authors of SimSiam [24], where they selected a shallower architecture when training on CIFAR-10 with ResNet18 rather than ResNet50. One hypothesis to explain this discrepancy in architectural sparsity is that ResNet18, being a shallower model than ResNet50, has a less powerful innate ability to extract representations and hence produces less complex feature maps that would not require a deep projector to be analyzed. Using a deeper architecture could even lead to adverse effects. To confirm this hypothesis, we tried to fit a ResNet18-based model on CIFAR-10 with the deeper encoder/predictor pair discovered for ResNet50. Fig. 5.6 clearly shows that this architectural setting quickly led to a collapsing behavior, with the contrastive loss rapidly saturating to -1 as soon as epoch 350 and a higher variance than the ResNet18-searched architecture. Hence, this validates the ability of our NASiam approach to discover backbone-specific architectures.

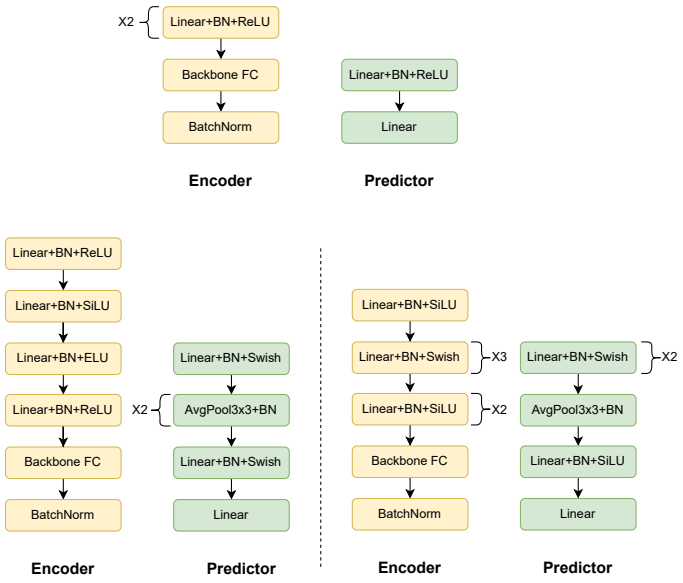


Figure 5.5: **Composition of encoder/predictor pair architectures.** (**Top**) SimSiam model. (**Bottom left**) NASiam model searched for 100 epochs on CIFAR-10 using SimSiam as the baseline framework with ResNet18 as the backbone CNN. (**Bottom right**) NASiam model searched for 100 epochs on CIFAR-10 using SimSiam as the baseline framework with ResNet50 as the backbone CNN. ResNet18-searched and ResNet50-searched architectures are clearly different, with ResNet50 needing a deeper encoder.

5.3 . Discussion and Conclusion

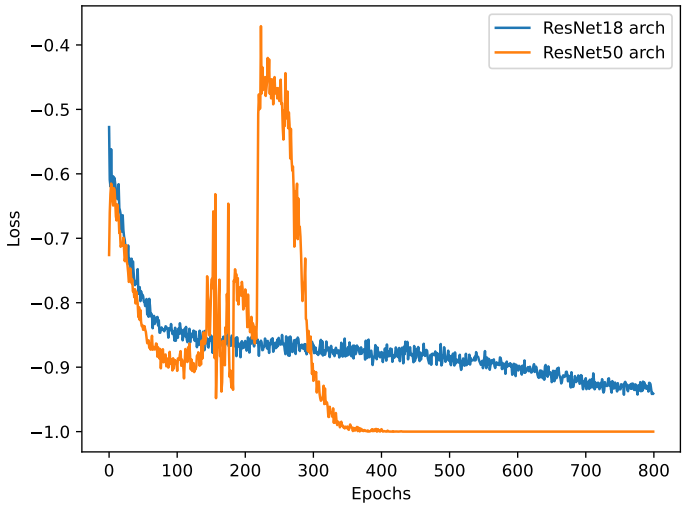


Figure 5.6: **Plot of the negative cosine similarity loss while pretraining NASiam with ResNet18 using architectures searched either with ResNet18 or ResNet50 as backbone.** The ResNet50-searched architecture quickly collapses towards -1 and has high variance while the ResNet18-searched one converges as expected.

This chapter presented NASiam, a novel approach for contrastive learning with Siamese networks that searches for efficient encoder/predictor pairs using differentiable neural architecture search (see Section 5.1.1).

NASiam is a universal approach that can improve the performance of many existing Siamese frameworks while maintaining their original structure. Therefore, it can be easily integrated with existing Siamese networks without the need for major modifications. Furthermore, the NASiam method is computationally efficient, as it requires only a few GPU hours to run. This makes it a practical and accessible solution for enhancing the performance of Siamese networks in a variety of applications.

Section 5.2 showed that NASiam discovers encoder/predictor pair architectures that efficiently learn robust representations and overperform previous baselines in small-scale and large-scale image classification datasets. These empirical results support our intuition that the encoder and predictor architectural designs play a decisive role in representation learning.

However, NASiam is not exempt from limitations. Most notably, the performance gain is somewhat limited, especially for ImageNet (see Table 5.4). This indicates that there is still room for improvement on large-scale datasets, perhaps by expanding the search space. Additionally, NASiam could be combined with more typical DNAS methods (such as DARTS [117]) to jointly search for the MLP predictor/projector heads and the CNN backbone. These ideas could be explored in future work.

Nevertheless, we hope this work will pave the way to further improvements in MLP-headed Siamese networks.

Algorithm 4 Algorithm describing the differentiable neural architecture search process of NASiam

Require: Object: C_e, encoder cell containing architectural weights
Require: Object: C_p, predictor cell containing architectural weights
Require: List: O, list of operations
Require: List: D_t, train dataset
Require: List: D_v, validation dataset
Require: Object: $model$, backbone CNN model
Require: Object: opt, model optimizer
Require: Object: $search\_opt$, search optimizer
Require: Integer: E, number of epochs
  for e in [0, E[**do**
    for (x_1, x_2) in D_v **do**
      $(x_1, x_2) \leftarrow model(x_1, x_2)$
      $(z_1, z_2) \leftarrow C_e(x_1, x_2)$
      $(p_1, p_2) \leftarrow C_p(x_1, x_2)$
      $\texttt{stop\_grad}(z_1, z_2)$
      $loss \leftarrow -0.5(\texttt{cosine\_similarity}(p_1, z_2) + \texttt{cosine\_similarity}(p_2, z_1))$

      $search\_opt.\texttt{optimization\_step}(loss, C_e.weights, C_p.weights)$
    end for
    for (x_1, x_2) in D_t **do**
      $(x_1, x_2) \leftarrow model(x_1, x_2)$
      $(z_1, z_2) \leftarrow C_e(x_1, x_2)$
      $(p_1, p_2) \leftarrow C_p(x_1, x_2)$
      stop_grad(z_1, z_2)
      $loss \leftarrow -0.5(\texttt{cosine\_similarity}(p_1, z_2) +$ cosine_similarity$(p_2, z_1))$
      $opt.\texttt{optimization\_step}(loss, C_e.weights, C_p.weights)$
    end for
  end for
  $A \leftarrow \texttt{empty\_list()}$
  for C in $\{C_e, C_p\}$ **do**
    $C_f \leftarrow \texttt{empty\_list()}$
    $n \leftarrow |C|$
    for i in $[0, n[$ **do**
      $op \leftarrow \texttt{argmax}(C.weights[i])$
      $\texttt{append}(op, C_f)$
    end for
    $\texttt{append}(C_f, A)$
  end for
  return A

6 - Applications of Differentiable NAS

In previous chapters, we focused on Computer Vision (CV) as the prevailing trend in DNAS literature revolves around this field and CNN architectures more specifically (see Chapter 3). However, DNAS methods are based on mathematical concepts that are generic enough to be applied to other fields and to a wide variety of neural network architectures.

Therefore, in this chapter, we broaden our horizons to explore the practical utility of DNAS in two real-world scenarios: **(I)** pruning Vision Transformers (ViT) models with low-cost proxies in a few minutes to increase the inference throughput (see Section 6.1) and **(II)** using DNAS-designed networks to perform torque control of Permanent Magnet Synchronous Motors (PMSM) (see Section 6.2).

6.1 . Exploring Differentiable NAS for Cost-Effective Vision Transformers: Differentiable Vision Transformer Pruning with Low-Cost Proxies

As recalled in Chapter 2, Vision Transformers (ViTs) [46] are becoming prominent in many Computer Vision fields of application such as image classification, object detection, or image segmentation.

However, a major issue of ViTs is that they often comport a very large number of parameters (e.g., around 630 million parameters for ViT-H [46]) making them especially hard to train and conferring a high inference latency. Consequently, it is necessary to find ways to reduce the computational requirements of ViTs and decrease their inference latency to make them deployable on embedded devices such as mobile phones. Using less computational resources will also save energy and thus engage in the ecological transition (i.e., Green Deep Learning [102, 59]). Pruning a ViT is one way of achieving this goal. Pruning a neural network consists in reducing its size by removing redundant or unuseful parts. Hence, our objective is to substantially decrease the inference latency while preserving comparable performance.

As seen in Chapters 2 and 3, DNAS aims to find the optimal architecture within a defined search space according to an objective function (usually a performance metric). DNAS usually starts from a blank slate but it is possible to use an existing architecture as the initial state (as seen in Chapter 5). Hence, by constraining the search algorithm to only removing existing connections and not adding new operations, it is possible to turn DNAS into a pruning method, as shown in this chapter.

In Section 6.1.1, we present an approach that performs efficient ViT prun-

ing within a few minutes on a single GPU. This way, we aim to promote frugal and low-cost Deep Learning. We dubbed our method DARIO (**D**ifferenti**A**ble vision transformer p**R**un**I**ng with low-cost pr**O**xies). DARIO employs a differentiable and data-agnostic search process that leverages low-cost proxies. These proxies enable the estimation of classification performance for pruned ViTs using a single forward and backward pass on a mini-batch of dummy data. By combining the differentiable search process with low-cost proxies, DARIO can effectively accelerate inference speed (up to 69 % faster) while maintaining comparable classification performance and fine-tuning speed.

The main logic behind DARIO is summarized in Fig. 6.1.

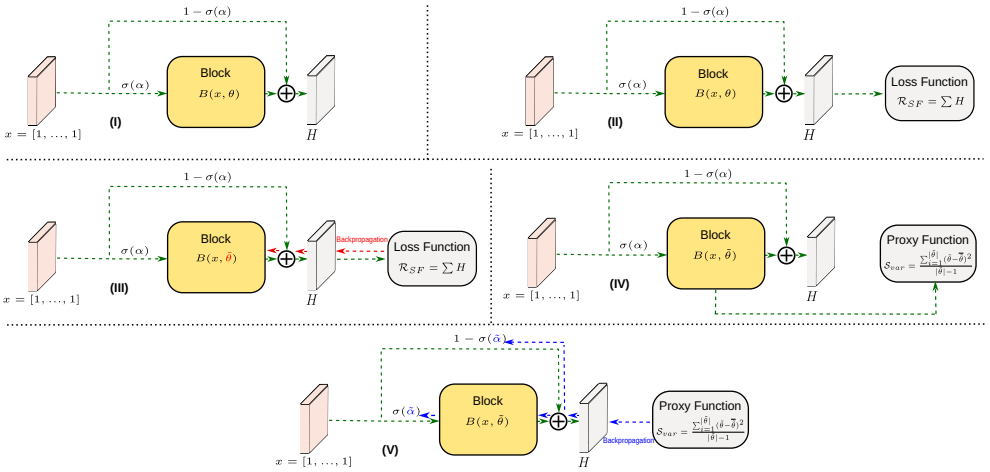


Figure 6.1: **Flowchart of the DARIO pruning process.** This example features a small toy-like neural network composed of only a single block that undergoes the five steps of the DARIO **data-agnostic** pruning process: **(I)** Meta-architecture parameters α are assigned to each path linking neurons together, thus forming a gating mechanism. A tensor of ones is fed as input to the neural network. **(II)** A loss function $\mathcal{R}_{SF}$ [175] is computed from the output H of the network. **(III)** $\mathcal{R}_{SF}$ is backpropagated to update the network parameters θ to $\tilde{\tilde{\theta}}$. **(IV)** The performance proxy function $\mathcal{S}^{var}$ is computed from $\tilde{\tilde{\theta}}$. **(V)** $\mathcal{S}^{var}$ is backpropagated to update α to $\tilde{\alpha}$ and the network parameters are reset to their original values θ. These steps are repeated until convergence of $\mathcal{S}^{var}$. Yellow nodes represent neurons, the red cylinder represents the input data, and the green rectangles stand for output-related data (e.g., output vector, loss value, proxy value).

The rest of this section is organized as follows: Section 6.1.2 presents the results of the experiments we conducted to validate our DARIO method, and Section 6.1.3 discusses the limitations of our method and gives insights for future work.

6.1.1 . Proposed Approach

In this section, we explain the design of our DARIO approach for pruning pre-trained vision transformers. We detail the four key concepts upon which DARIO is based: **Search space:** Meta-architecture spanning several levels of granularity, with continuous gating parameters; **Action mechanism:** Iterative, differentiable pruning using gradient descent in the gating parameters; **Reward mechanism:** Low-cost classification performance proxy; **Information:** Data agnostic (the input is a tensor fully composed of ones).

Meta-architecture and levels of granularity of pruning

Our goal is to create a search space that is both easily navigable and highly configurable. To achieve this, we draw inspiration from the efficiency of Differentiable NAS methods [117, 210] when exploring large search spaces.

In our proposed DARIO method, a crucial aspect is the inclusion of a meta-architectural space that spans multiple levels of granularity. We focus on model pruning for vision transformer (ViT) models [46] in this study. The fundamental building blocks of ViT models are the multi-head attention blocks. Typically, ViTs consist of a sequence of these multi-head attention blocks, each comprising a multi-head attention layer, two fully-connected layers (MLP), normalization layers, and skip-connections. The ability to prune at various levels of granularity is essential, as the topology constraints of the parts being removed can vary depending on the context.

As a result, our exploration of vision transformer pruning involves two levels of granularity: (1) attention-head and (2) block. Pruning at the block level entails removing entire multi-head attention blocks at once, made feasible by the presence of skip-connections within each multi-head attention block [69, 46, 172] in the pre-trained ViT models. In contrast, attention-head-wise pruning involves removing specific heads within a multi-head attention block, making it a more fine-grained approach compared to block-wise pruning. Moreover, we also investigate bi-granularity pruning, where pruning is performed at both levels. This means that certain blocks are entirely pruned, while some heads are pruned within the remaining blocks. The choice of pruning granularity is discussed in Section 4.2.

In our approach, these levels of granularity modulate a meta-architecture, also referred to as the "supernet", which encompasses the original pre-trained ViT model M. Specifically, the model M is decomposed into n modules $\{m_1, m_2, ..., m_n\}$, where each module may represent either an attention head within a block or an entire block in the model, depending on the desired level of pruning granularity. The supernet is parameterized by meta-architectural parameters $\alpha = \{\alpha_1, ..., \alpha_n\}$, enabling continuous exploration of the search space. The meta-architectural parameters α^A for attention heads and α^B for blocks allow for flexible pruning by gating the output features H_m of each module $m \in M$. This approach relaxes the categorical choice of modules and

enables "soft" model pruning of the supernet, providing greater flexibility in the pruning process.

At the block level, gating is performed as follows:

$$\tilde{H_m} = \sigma(\alpha_m^B)H_m + (1 - \sigma(\alpha_m^B))H_{m-1}, \tag{6.1}$$

where $\tilde{H_m}$ represents the gated value of H_m. At the attention-head granularity, it is performed as:

$$\tilde{H_m} = (\sigma(\alpha_m^A)A_m) \cdot V_m, \tag{6.2}$$

where A_m is the attention probability matrix of block m, V_m is the transposed value matrix, and σ represents the sigmoid function defined as:

$$\sigma(x) = \frac{1}{1 + exp(-x)}. \tag{6.3}$$

We deliberately opted against pruning at the individual parameter level to preserve the meta-architectural vision we presented earlier. Implementing gating on every parameter in our model would have been impractical and computationally intensive, leading to a significant increase in computational complexity. Furthermore, the potential gains in terms of inference speed would have been limited, as pruning a few parameters from a layer has a relatively minor impact compared to removing entire layers. Considering individual parameters would have also resulted in a large cardinality for α, equal to the total number of parameters in the model M, which could have hindered the effectiveness and interpretability of our approach. Thus, by focusing on attention-head and block-level pruning, we strike a balance between computational efficiency and model performance while maintaining the flexibility and interpretability of the meta-architectural framework.

Figure 6.2 illustrates the gating mechanisms for block and attention-head-level granularities.

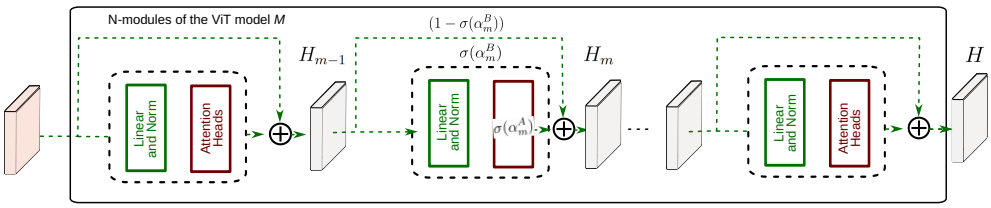


Figure 6.2: **Illustration of DARIO's meta-architecture for a N-block ViT model M.** The gating mechanisms for both block-level and attention-head-level granularities are represented and can be independently deactivated.

Performance proxies

Our goal is to find pruned models that exhibit both strong generalization capabilities and high performance. Therefore, our DARIO pruning approach cannot directly rely on a groundtruth-based, data-specific loss as its objective function, as doing so would limit the resulting pruned model to a narrow, task-specific context. Instead, the objective function of the pruning process must be a generic data-agnostic performance estimation function, which we refer to as a "proxy". By using such a proxy function, we ensure that the pruning process remains independent of any specific dataset and can efficiently search for models that excel across various tasks and domains.

DARIO leverages proxies to evaluate the potential of a given pre-trained model $M \in \mathcal{M}$ in achieving high accuracy. A performance proxy is formally defined as a function denoted $\mathcal{S} : \mathcal{M} \mapsto \mathbb{R}$, where $\mathcal{M}$ represents the space encompassing all potential neural networks that can be obtained by pruning a pre-trained neural network. We consider proxies that adhere to specific constraints:

Data-agnostic: Our aim is to achieve data-agnostic pruning, meaning that the proxies should solely depend on the neural network M and not on any specific data.

Low-cost: Efficiency is vital in our approach to promote resource-efficient Deep Learning. All candidate proxies only require a single forward pass and backward pass over a single minibatch of dummy data.

Scale invariant: The proxies must be normalized to account for model size variations. While larger models may exhibit stronger classification performance, we want the proxies to indicate performance per unit, avoiding model size as a confounding factor.

Differentiable: The proxies need to be differentiable so that they can serve as the objective function for the differentiable search process (see Section 6.1.1). By ensuring differentiability, we enable efficient optimization during the neural architecture search process.

In line with Abdelfattah et al. [1], we adapt importance scores originally designed for finer granularity (i.e., individual parameters) to serve as performance proxies for entire models. Their empirical experiments demonstrate that *synflow* [175] (denoted $\mathcal{S}^s$) performs well in maintaining rank consistency between randomly initialized models and their classification performance post-training. To meet the predefined constraints, *synflow* requires normalization, resulting in the normalized version $\mathcal{S}^{ns}$. Additionally, we consider L_n norms ($\mathcal{S}^{L_0}$, $\mathcal{S}^{L_1}$, $\mathcal{S}^{L_2}$) as potential performance proxies, adapting magnitude-based pruning scores. Moreover, we propose a new variance-based proxy, denoted as $\mathcal{S}^{\text{var}}$, and develop several composite proxies for evaluation. In total, we assess 9 candidate proxies, as listed in Table 6.1. For comparison purposes, we also include $\mathcal{S}^s$.

Table 6.1: **List of candidate performance proxies.** With the exception of *synflow*, all proxies satisfy the 4 constraints: data-agnosticism, scale-invariance, low-cost, and differentiability. $\odot$ stands for the Hadamard product. $\mathcal{S}^{\text{var}}$ is our proposed proxy used in DARIO (see Section 6.1.2).

Proxy name	Proxy definition
L_0 norm	$\mathcal{S}^{L_0}(\tilde{\theta}) = \frac{1}{T}\sum_t^T \lvert\lvert\tilde{\theta}_t\rvert\rvert_0$
L_1 norm	$\mathcal{S}^{L_1}(\tilde{\theta}) = \frac{1}{T}\sum_t^T \lvert\lvert\tilde{\theta}_t\rvert\rvert_1$
L_2 norm	$\mathcal{S}^{L_2}(\tilde{\theta}) = \frac{1}{T}\sum_t^T \lvert\lvert\tilde{\theta}_t\rvert\rvert_2$
synflow	$\mathcal{S}^{\text{s}}(\tilde{\theta}) = \sum_n^N \left(\frac{\partial H^a}{\partial \lvert\theta\rvert} \odot \lvert\tilde{\theta}^a\rvert \right)_n$
normalized synflow	$\mathcal{S}^{\text{ns}}(\tilde{\theta}) = \frac{1}{N}\sum_n^N \left(\frac{\partial H^a}{\partial \lvert\theta\rvert} \odot \lvert\tilde{\theta}^a\rvert \right)_n$
parameter variance	$\mathcal{S}^{\text{var}}(\tilde{\theta}) = \frac{1}{N-1}\sum_n^N (\tilde{\theta}_n - \overline{\theta})^2$
L_0 + parameter variance	$\mathcal{S}^{L_0+\text{var}}(\tilde{\theta}) = \mathcal{S}^{L_0}(\tilde{\theta}) - c_1\mathcal{S}^{\text{var}}(\tilde{\theta})$
L_1 + parameter variance	$\mathcal{S}^{L_1+\text{var}}(\tilde{\theta}) = \mathcal{S}^{L_1}(\tilde{\theta}) - c_2\mathcal{S}^{\text{var}}(\tilde{\theta})$
L_0 + L_1	$\mathcal{S}^{L_0+L_1}(\tilde{\theta}) = \mathcal{S}^{L_0}(\tilde{\theta}) + c_3\mathcal{S}^{L_0}(\tilde{\theta})$

Table 6.1 presents the definitions of various terms used in the calculation of the candidate proxies. Here, T represents the total number of parameter tensors in the neural network, and N is the total number of parameters (scalars) in the network. The symbol $\overline{\theta} = \frac{1}{N}\sum_n^N \tilde{\theta}_n$ denotes the average parameter value. Additionally, c_1, c_2, and c_3 are empirical coefficients used to scale both terms consistently in the composite proxies. The term H^a is similar to H, but the forward pass is computed after transforming all model parameters θ into their absolute values θ^a. The inclusion of the $(N-1)$ term in the parameter variance proxy accounts for Bessel's correction.

The composite proxies, namely $\mathcal{S}^{L_0+\text{var}}$, $\mathcal{S}^{L_1+\text{var}}$, and $\mathcal{S}^{L_0+L_1}$, are obtained by weighted sums of other proxies. Through empirical testing in Section 6.1.2, we found that the variance-based proxy $\mathcal{S}^{\text{var}}$ yielded the most promising results.

The rationale behind $\mathcal{S}^{\text{var}}$ lies in its ability to measure the parameter spread within a model. Networks with more uniform parameter sets are often indicative of better performance. Specifically, a high spread in parameters may suggest that a model has captured the residual variation specific to a dataset, leading to potential overfitting [5, 211].

Differentiable search algorithm

Our primary objective is to enhance the efficiency of a pre-trained ViT model M by reducing its inference latency and computational demands while maintaining its performance across various tasks, as discussed in Section 6.1.1. Additionally, we aim to eliminate irrelevant or detrimental components present

in ViTs. To achieve this, we propose DARIO, an approach that optimizes the meta-architectural parameters α of the pre-trained model M based on a proxy function $\mathcal{S}$. By modulating the components of the model according to this performance-based objective function, we can effectively prune the model while preserving its generalization capability and potentially improving or maintaining its performance. Hence, DARIO aims to optimize the values of the gating parameters α to solve the following optimization problem:

$$\min_{\alpha} \mathcal{S}(\tilde{\theta}(\alpha)), \tag{6.4}$$

where $\mathcal{S}$ is a proxy function we assumed to be negatively correlated to the model performance and hence to be minimized. Hence, Eq. 6.4 indicates that we must evaluate the impact of α on $\tilde{\theta}$, which represents the updated parameters θ based on the values of α.

DARIO employs a gradient-based procedure inspired by the Synaptic Flow loss ($\mathcal{R}_{SF}$) introduced by Tanaka et al. [175]. By using $\mathcal{R}_{SF}$, we can update θ in a data-agnostic manner. It is defined as follows:

$$\mathcal{R}_{SF} = \mathbb{1}^T \left(\prod_{m=1}^{n} |\theta^{[m]}| \right) \mathbb{1}, \tag{6.5}$$

where $|\theta^{[m]}|$ is a matrix containing the absolute values of the pre-trained parameters of block m with θ denoting the full set of pre-trained parameters. The underlying concept of the $\mathcal{R}_{SF}$ loss is that it encompasses all potential paths from each input element to each output value, where the path value is obtained by multiplying the parameter values along that path. To compute $\mathcal{R}_{SF}$, following the approach of Tanaka et al. [175] and Abdelfattah et al. [1], we perform a forward pass using a single minibatch x of dummy uniform data (i.e., tensors containing ones), and then compute a backward pass using the sum of the outputs H of the model M. This process allows us to obtain the gradients of $\mathcal{R}_{SF}$ with respect to θ (i.e., to perform the outer and inner products with respect to x).

From this point, it becomes possible to update the gating parameters α by backpropagating the gradient of an objective function $\mathcal{S}$ computed over θ, using the different paths obtained through $\mathcal{R}_{SF}$. For example, *synflow* by Tanaka et al. [175] considers the gradients of $\mathcal{R}_{SF}$ with respect to θ to estimate the trainability of parameters, aiming to find a good network initialization by pruning the less trainable parameters. In contrast, our objective is to assess the impact of the meta-architectural configuration (parameterized by α) of the model on the objective function $\mathcal{S}$. To that end, in addition to the backward pass, we also perform a single optimization step to update θ into $\tilde{\theta}$ in order to obtain a value differentiable w.r.t. α:

$$\tilde{\theta}(\alpha) = \theta - \gamma \frac{\partial \mathcal{R}_{SF}}{\partial \theta} = \theta - \gamma \frac{\partial \sum H(\alpha)}{\partial \theta}, \tag{6.6}$$

where γ is the step size for the gradient descent. This way, we are able to backpropagate the value of $\mathcal{S}(\tilde{\theta})$ to update the α parameters.

While DARIO shares similarities with DARTS [117] (see Chapter 2), it distinguishes itself by adopting a single-level optimization approach. In DARTS, both the architectural parameters α and the model parameters θ are jointly searched, forming a bi-level optimization problem. However, DARIO focuses solely on optimizing α and leaves θ unchanged, resulting in a simpler single-level optimization. As a result, the supernet is reinitialized at each step to revert $\tilde{\theta}$ to its original value.

To enhance the efficiency of the search process, we incorporate a warm start phase involving several iterations of random search. This phase helps identify a set of α parameters that best optimize the objective function $\mathcal{S}$, serving as the starting point for the pruning process. After completing the search, we obtain the final pruned model by removing modules associated with α values below a certain threshold. The determination of this threshold is discussed in Section 6.1.2.

All the proxies introduced in Section 6.1.1 are differentiable with respect to α since they are functions of the classification features H (see Equation 6.6), which are themselves dependent on the search parameters α. The pseudo-code of the differentiable search process is provided in Algorithm 5. We demonstrate the advantages of our proposed differentiable search algorithm over random search in Section 6.1.2.

Algorithm 5 Pseudo-code describing the differentiable search at works in DARIO.

Input: proxy $\mathcal{S}$, pre-trained model M, cardinality of the meta-architectural parameters l, search learning rate λ, model learning rate γ, number of warm start iterations k, number of search iterations n
Output: α

for k iterations **do**
  $\alpha \leftarrow$ warm_start(l)
  {Warm start α by keeping the best of k random samples }
end for
for n iterations **do**
  $x \leftarrow$ create_batch_ones$()$
  $o \leftarrow M(\alpha, x)$
  {$M(\alpha, x)$ means that the model is gated by α; this step contains one forward pass and one backward pass of one minibatch x of data.}
  $\theta \leftarrow$ get_parameters(M)
  $\tilde{\theta} \leftarrow \theta - \gamma \frac{\partial o}{\partial \theta}$
  $s \leftarrow \mathcal{S}(\alpha, \tilde{\theta}(\alpha))$
  $\alpha \leftarrow \alpha - \lambda \frac{\partial s}{\partial \alpha}$
end for

6.1.2 . Experiments

This section presents the experimental results validating our DARIO approach. We start by comparing various classification performance proxies (Section 6.1.1) to identify the most effective proxy for estimating classification performance (Section 6.1.2). Next, we investigate different pruning granularities (Section 6.1.1) in Section 6.1.2. Additionally, we evaluate the effectiveness of the differentiable search process (Section 6.1.1) by comparing it to random search in Section 6.1.2. Finally, we assess the performance of pruned models achieved using the optimal proxy and granularity through classification experiments conducted on various image datasets (Section 6.1.2).

Experimental settings

To assess the effectiveness of DARIO, we conducted experiments using two pre-trained ViT models: MAE-ViT-base [71] and MobileViT-small [132]. These models represent large and small state-of-the-art ViT architectures, respectively. We evaluated the classification performance preservation of the pruned models using the Meta-Album micro meta-dataset [183], which provides a diverse set of datasets from various domains (see Section 2.4.6 of Chapter 2). We compared the accuracy of the pruned models with that of the original pre-trained models. To avoid overfitting on Meta-Album, we conducted all intermediate experiments (e.g., Section 6.1.2) on a separate dataset called ICDAR-micro, following the approach of Sun et al. [172]. ICDAR-micro is derived from an OCR dataset containing images with alphanumeric characters in natural scenes [123]. It shares a similar train-test split and has an equal number of examples per class as Meta-Album micro. Our implementation is based on PyTorch, and all experiments were performed using Nvidia Tesla V100 and RTX 4090 GPUs.

Figure 6.3: **Sample images from ICDAR-micro.**

Comparing classification performance proxies

In this section, we compared the classification performance proxies presented in Table 6.1 of Section 6.1.1 to identify the most suitable one for the differen-

tiable search process (as discussed in Section 6.1.1).

Our goal was to find proxies that effectively estimate the performance of pruned models. To achieve this, we needed to assess the correlation between the proxy values and the ground-truth classification accuracy of fine-tuned pruned models using the ICDAR-micro dataset. Following Abdelfattah et al. [1], we employed the Spearman correlation coefficient (Spearman's ρ) [170] as the metric to select the most reliable proxy. Additionally, we reported the Pearson correlation coefficient (Pearson's ρ) [151].

For each pre-trained model $p \in \{\text{MAE-ViT-base}, \text{MobileViT-small}\}$, a search space $\mathcal{M}_p$ was defined, encompassing all possible models obtained by pruning a given pre-trained model. We considered two pre-trained models: MAE-ViT-base with 12 multi-head attention blocks (each containing 12 attention heads), leading to $|\mathcal{M}_{\text{MAE-ViT-base}}| = 2^{12\times12} > 2 \times 10^{43}$ possible models, and MobileViT-small with 9 multi-head attention blocks (each containing 4 attention heads), leading to $|\mathcal{M}_{\text{MobileViT-small}}| = 2^{9\times4} > 6 \times 10^{10}$ possible models. We used the attention-head granularity (as described in Section 6.1.1) to define the search spaces, which also encompassed all models attainable through block-wise pruning (i.e., an entire block is pruned when all attention heads within that block are pruned).

To compute the correlations, we randomly sampled 100 models from each associated search space $\mathcal{M}_p$. Each sampled model was then fine-tuned to obtain its ground-truth classification accuracy. Both Spearman and Pearson correlation coefficients were computed over the 100 sampled models. The results of the Spearman correlation experiment are shown in Figure 6.4, and the results of the Pearson correlation experiment can be found in Figure 6.5. Tabular results are available in Table 6.2 and Table 6.3.

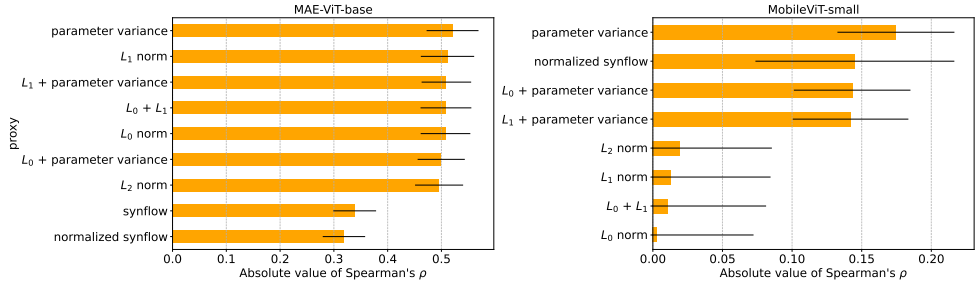


Figure 6.4: **Absolute value of Spearman correlation coefficients on ICDAR-micro.** The 95% confidence intervals are computed with 12 independent runs. The parameter variance proxy consistently has the highest absolute value across both pre-trained models.

To select the best proxy, we considered two criteria: (1) a high absolute value of the Spearman correlation coefficient, and (2) consistent signs of the Spearman correlation coefficient across different pre-trained models. Our

Table 6.2: **Correlation between proxy value and classification accuracy on MAE-ViT-base.** proxies with "+" represent composite proxies. The 95% confidence intervals are computed with 12 independent runs. The parameter variance proxy consistently has the highest correlation (ρ) absolute value for both Pearson and Spearman.

Proxy	Spearman's ρ	Pearson's ρ
Parameter variance	-0.48 ± 0.04	-0.48 ± 0.04
Normalized synflow	0.27 ± 0.06	0.28 ± 0.05
synflow	0.29 ± 0.06	0.29 ± 0.05
L_2 norm	0.45 ± 0.04	0.46 ± 0.03
L_0 norm + parameter variance	0.45 ± 0.04	0.46 ± 0.03
L_0 norm + L_1 norm	0.46 ± 0.04	0.47 ± 0.03
L_0 norm	0.46 ± 0.04	0.46 ± 0.03
L_1 norm + parameter variance	0.46 ± 0.04	0.47 ± 0.03
L_1 norm	0.46 ± 0.04	0.47 ± 0.04

Table 6.3: **Correlation between proxy value and classification accuracy on MobileViT-small.** proxies with "+" represent composite proxies. The 95% confidence intervals are computed with 12 independent runs. The parameter variance proxy consistently has the highest correlation (ρ) absolute value for both Pearson and Spearman.

Proxy	Spearman's ρ	Pearson's ρ
Parameter variance	-0.18 ± 0.04	-0.12 ± 0.04
L_1 norm	-0.07 ± 0.06	-0.08 ± 0.06
L_0 norm + L_1 norm	-0.06 ± 0.05	-0.07 ± 0.06
L_0 norm	-0.05 ± 0.05	-0.06 ± 0.05
Normalized synflow	-0.05 ± 0.08	-0.05 ± 0.06
L_2 norm	-0.04 ± 0.06	-0.05 ± 0.06
L_1 norm + parameter variance	0.15 ± 0.04	0.10 ± 0.04
L_0 norm + parameter variance	0.16 ± 0.04	0.10 ± 0.04

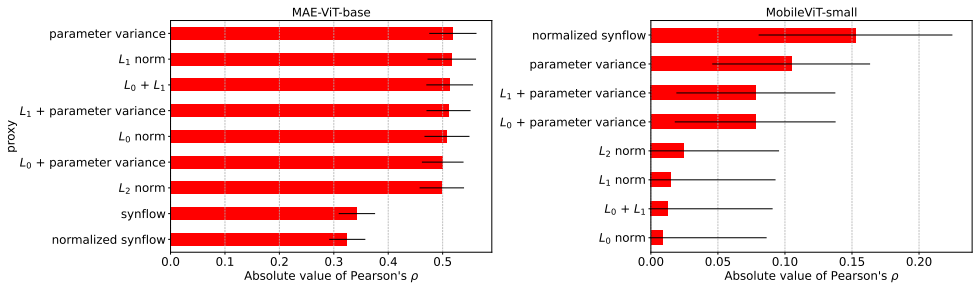


Figure 6.5: **Absolute value of Pearson correlation coefficients.** The 95% confidence intervals are computed with 12 independent runs. The parameter variance proxy consistently has the highest absolute value across both pre-trained models.

newly proposed parameter variance proxy met both criteria, showing consistent signs across different pre-trained models (as shown in Table 6.2 and Table 6.3). Additionally, the parameter variance proxy consistently exhibited the highest absolute values of correlation coefficients for both Spearman and Pearson correlations across different pre-trained models (as depicted in Figure 6.4 and Figure 6.5). As a result, we decided to utilize the parameter variance proxy for the differentiable search process in the subsequent sections. Given that this proxy function exhibits a negative correlation with classification accuracy, we minimize it during the search process to discover models with high classification performance.

Comparing pruning granularities

We conducted a comparison of the three granularities introduced in Section 6.1.1 (i.e., *full*, *attention head*, and *bi-granularity*). The learning curves obtained while minimizing the parameter variance proxy for these granularities can be found in Figure 6.6 and Figure 6.7.

All three granularities yielded positive outcomes by achieving lower values compared to the original model (represented by the blue line). However, the *block* granularity emerged as the predominant choice for both models. When pruning MAE-ViT-base, the average parameter variance proxy value obtained with the block granularity (5.6545×10^{-3}) is lower than that of the other granularities (5.6557×10^{-3} for the *attention head* granularity and 5.6547×10^{-3} for the *bi-granularity*). A similar predominance was observed when pruning MobileViT-small. Moreover, block-wise pruning also offers a more significant inference speedup (see Section 6.1.2). As a result, we conducted all subsequent experiments using the *block* granularity.

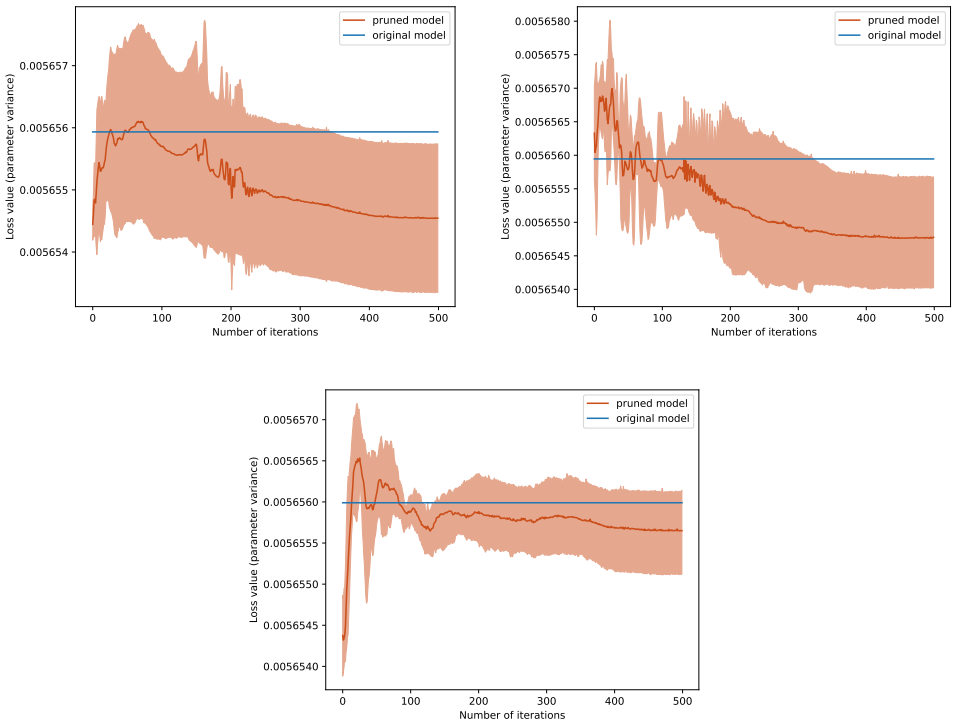


Figure 6.6: **Comparison between the learning curves for *block* (left), *attention head* (bottom), and *bi-granularity* (right) when pruning MAE-ViT-base [71].** We used our proposed parameter variance proxy ($\mathcal{S}^{var}$) and averaged over 5 independent runs. It can be noted that the *block* granularity is leading as it converges towards a minimum of 5.6545×10^{-6} compared to 5.6547×10^{-6} for the bi-level granularity, and 5.6557×10^{-6} for the attention-head granularity (which does not seem to provide any significant gains).

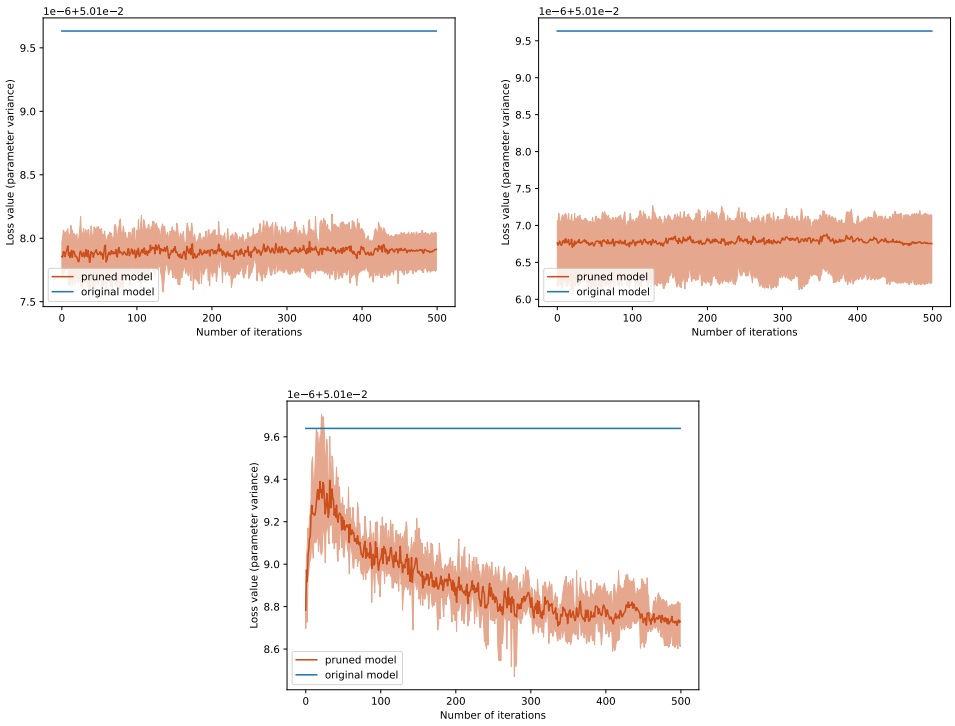


Figure 6.7: **Comparison between the learning curves for *block* (left), *attention head* (bottom), and *bi-granularity* (right) when pruning MobileViT-small[132].** We used our proposed parameter variance proxy ($\mathcal{S}^{var}$) and averaged over 5 independent runs. It can be noted that the attention-head granularity performs poorly, reaching a higher loss value (around 5.01088×10^{-2}) than the other granularities. The bi-granularity reaches the lowest loss value (5.01067×10^{-2}) but suffers from higher variance than the block granularity.

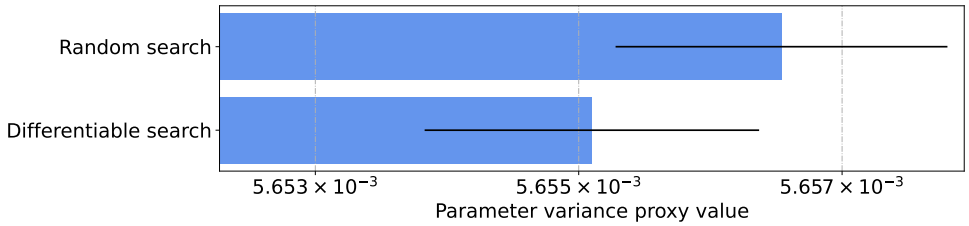


Figure 6.8: **Results of running the random search and our proposed differentiable search when pruning MAE-ViT-base [71] at the block granularity with the parameter variance proxy for 5 runs of 500 iterations.** The error bars represent the 95 % confidence intervals.

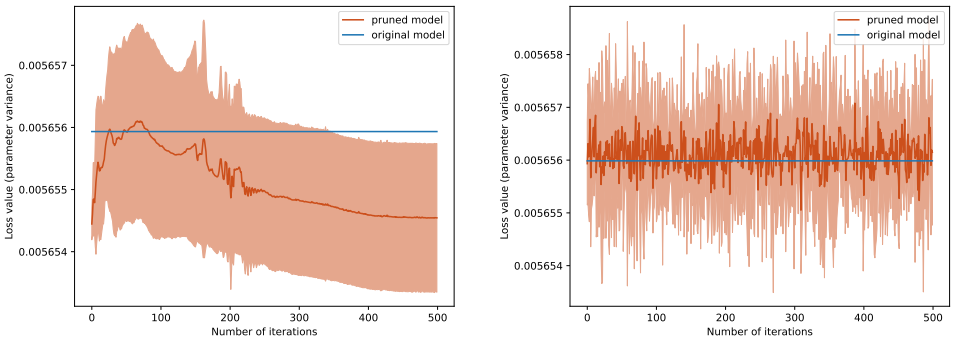


Figure 6.9: **Comparison between the plot of the learning curve of our proposed differentiable search process (left) and the random search (right).** We pruned MAE-ViT-base[71] at the block granularity using our proposed parameter variance proxy ($\mathcal{S}^{var}$) and averaged over 5 independent runs. The blue line represents the proxy value of the original (i.e., unpruned) model. It can be noted that the learning curve of the differentiable search smoothly converges.

Comparing iterative search processes: differentiable search vs random search

To evaluate the effectiveness of our proposed differentiable search process (Section 6.1.1), we compared its results with those obtained through a random search. In the random search, the architectural parameters α were randomly sampled from a normal distribution $\mathcal{N}(0, 1)$. The baseline model used for this comparison was MAE-ViT-base [71], and the search process was conducted over the block granularity.

Figure 6.8 shows that the differentiable process allows reaching a lower (better) parameter variance proxy value. The learning curves of both search processes are presented in Figure 6.9. It should be noted that, although the difference between values is small, it is not negligible as these proxy values

cannot be directly interpreted and instead represent a ranking between models. These results confirm the advantage of our proposed differentiable approach.

Choice of threshold

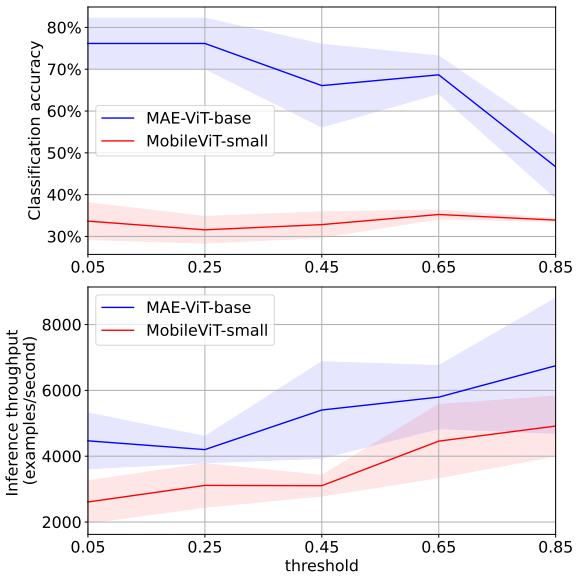


Figure 6.10: **Evolution of classification accuracy and inference speed of the pruned models in function of the threshold.** The thresholding is performed on α parameters shown in Figure 6.11 (block-wise pruning). The inference speed is measured by the number of examples per second. The classification accuracies are from the validation dataset ICDAR-micro. The last block is fine-tuned together with the classification head. Error bars are 95 % confidence intervals.

Within the framework of DARIO, the pruned models emerge through the application of a binary thresholding operation to the α parameters, which are generated during the differentiable search process (outlined in Section 6.1.1). The effects of manipulating the pruning threshold are graphically depicted in Figure 6.10. It is worth noting that elevating the threshold engenders swifter inference times by facilitating the removal of more parameters. Regarding classification accuracy, distinct trends surface for MAE-ViT-base and MobileViT-small. Specifically, as the threshold escalates, MAE-ViT-base experiences a diminishing classification performance, particularly beyond the threshold of 0.65. Conversely, the classification performance of MobileViT-small demonstrates an intriguing pattern, with some instances showcasing improved accuracy as the threshold increases, reaching a pinnacle at 0.65. As a result, we

judiciously opted for the threshold value of 0.65 for the conclusive evaluation conducted on Meta-Album, as expounded upon in Section 6.1.2.

Image classification experiments

We evaluated the performance of our pruned models on Meta-Album datasets. The pruning process involves thresholding the obtained search parameters α (see Figure 6.11), which allows us to control the trade-off between classification accuracy and training/inference speed. Through experiments on the validation dataset ICDAR-micro, we chose 0.65 as the threshold for both MAE-ViT-base and MobileViT-small (see Figure 6.11 and 6.1.2).

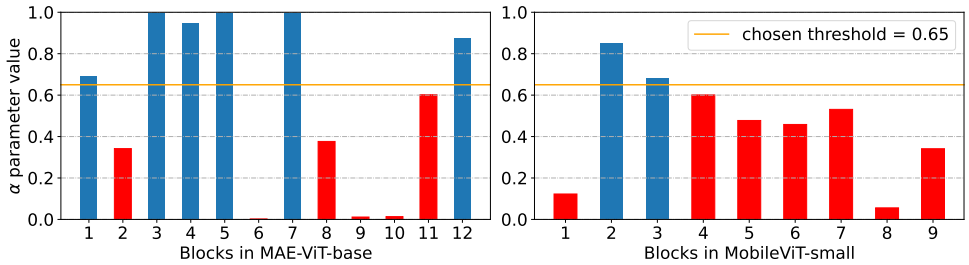


Figure 6.11: **Resulting search parameters α associated with each block in MAE-ViT-base (left) and MobileViT-small (right).** There are 12 multi-head attention blocks in MAE-ViT-base and 9 multi-head attention blocks in MobileViT-small. The orange line represents the threshold chosen empirically (0.65). Blocks with α values lower than the threshold are pruned and represented in red, while blocks with higher α values are kept and represented in blue. We pruned 6 out of the 12 multi-head attention blocks of MAE-ViT-base, reducing the number of parameters from 85.8 to 43.3 million. Similarly, for MobileViT-small, we pruned 7 out of the 9 multi-head attention blocks, reducing the number of parameters from 5.0 to 2.5 million.

Block-wise pruning leads to a drastically decreased training time, which allows us to train an additional block (i.e., the last one) along with the classification head when fine-tuning. Hence, we fine-tuned the last block together with the classification head for pruned models.

Figure 6.12 presents an overview of the classification performance of the pruned models in comparison to their unpruned counterparts. The classification performance of the pruned models varies across different datasets due to the diverse nature of the Meta-Album datasets [183]. On average, the pruned MAE-ViT-base achieves slightly lower classification accuracy than its unpruned counterpart, while the pruned MobileViT demonstrates comparable classification accuracy.

Regarding training and inference efficiency, Figure 6.13 illustrates the results. The pruned MAE-ViT-base model shows a 69% improvement in infer-

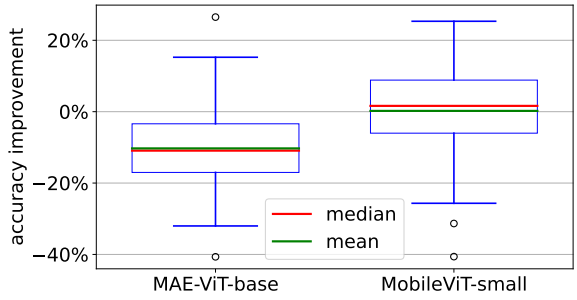


Figure 6.12: **Box plot of accuracy improvement over the 40 Meta-Album datasets on both pre-trained models.** The vertical axis shows the increase in test accuracy from the unpruned pre-trained model to the pruned model (higher is better). DARIO-pruned models consistently reach similar or better performance compared to their unpruned counterparts.

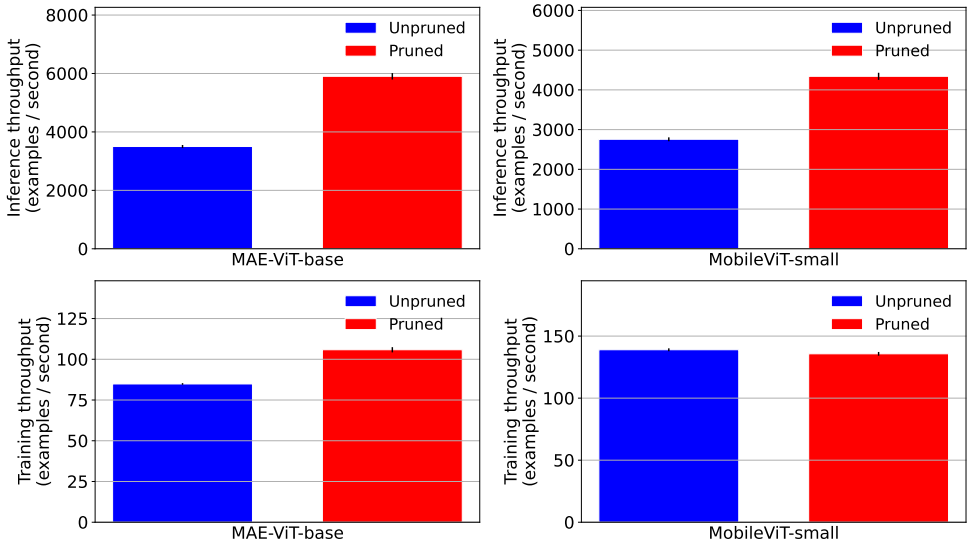


Figure 6.13: **Training and inference throughput.** The vertical axis shows the throughput of training and inference. Throughput is measured by the number of processed examples per second, thus the higher the throughput is, the more efficient the model is. The 95% confidence intervals are computed with repeated runs. DARIO-pruned models consistently achieve better inference throughput compared to their unpruned counterparts.

ence speed compared to its unpruned counterpart, while the pruned MobileViT-small model achieves a 58% improvement in inference speed. In terms of training speed, the pruned MAE-ViT-base is 25% faster than its unpruned counterpart, while the pruned MobileViT-small requires a similar training time.

6.1.3 . Discussion and Conclusion

We introduce DARIO, a data-agnostic pruning method designed specifically for pre-trained Vision Transformers (ViTs). A significant advantage of DARIO is its data-agnostic nature, requiring only a single application per pre-trained model, making it an efficient and fast-pruning solution. By utilizing DARIO, significant gains in inference speed (up to 69%) can be achieved, while still maintaining or even improving classification performance and fine-tuning time.

DARIO focuses on the pruning of entire multi-head attention blocks, which leads to more substantial speed improvements for two main reasons. Firstly, attention-head-wise pruning leaves adjacent structures within each multi-head attention block intact, such as fully-connected and normalization layers [46]. Secondly, modern hardware allows parallelized computations within a layer, while computations between layers and blocks are inherently sequential, limiting the benefits of parallel computing [172].

In our experiments (Section 6.1.2), we observed that models pruned using DARIO can even exhibit performance improvements, as seen in the case of MobileViT-small (+16% top-1 accuracy). This indicates that even in small state-of-the-art ViT models, there is redundancy and potential for further runtime efficiency enhancements. These findings underscore the significance of DARIO and align with previous research like the Lottery Ticket Hypothesis [54, 127].

Furthermore, while our focus in this chapter has been on computer vision applications with ViTs, it is important to note that DARIO is a universal approach that can also be applied to transformer models in Natural Language Processing (NLP) tasks.

As the size of deep learning models, especially transformers, has grown significantly in recent years, from millions (e.g., ELMo [156]) to billions (e.g., Megatron-Turing NLG [169]) of parameters, there is a growing demand to explore methods for reducing their size and inference latency. DARIO shows promise as a pruning approach for these large models, creating smaller versions that can be directly deployed on resource-constrained devices like mobile phones. By democratizing access to these models, DARIO contributes to accessibility while also supporting ecological efforts by preserving computational resources.

However, DNAS is not restricted to computer vision applications. In the next section, we show that DNAS can be successfully applied to discover new

Multi-Layer Perceptron (MLP) designs and activation functions for control theory applications.

6.2 . Torque Control of a Permanent Magnet Synchronous Motor using Differentiable Neural Architecture Search

In addition to classical Computer Vision (CV) tasks such as image classification and object detection (as seen in Chapters 4 and 5), DNAS can also be applied to new domains, especially those that recently started to leverage Deep Learning, such as automation and control theory [218, 62]. These fields employ neural networks but existing works lack analysis of the composition of their architectures, aside from empirical validation. They do not have any certainty about the optimality of the architectures they selected. Hence, DNAS comes into play as a way of discovering better, task-specific, neural network architectures.

Permanent Magnet Synchronous Motors (PMSMs) [157] represent a cutting-edge advancement in electric motor technology. These motors are designed for efficiency and precision, making them highly valuable in various industrial and commercial applications. What sets PMSMs apart is the synchronization between the rotor's magnetic field and the stator's rotating magnetic field, facilitated by permanent magnets embedded in the rotor. This synchronization results in minimal energy loss, reduced heat generation, and smoother operation compared to traditional asynchronous motors. PMSMs offer rapid response times and precise control over speed and torque, making them well-suited for applications requiring high performance and reliability, such as electric vehicles, robotics, industrial automation, and renewable energy systems. With their high efficiency, low maintenance requirements, and ecological attributes, Permanent Magnet Synchronous Motors have reshaped the landscape of modern engineering and contributed to the transition towards a more sustainable future.

In opposition to the traditional PID-based controllers, some neural-network-based controllers for PMSMs were recently proposed with promising results [206, 58, 150]. However, most of these works use "generic" architectures whose composition is never discussed.

In this chapter, we focused on designing a task-specific DNAS-based controller for PMSMs that is able to estimate its torque. We presented our approach in Section 6.2.2. The rest of this section is organized as follows: Section 6.2.1 presents the governing equations of PMSMs. Section 6.2.3 features the results of our experiments, and Section 6.2.4 discusses the limitations of our results and brings a conclusion to our work.

6.2.1 . Preliminaries

The governing equations for PMSMs in the $d-q$ axis are given by:

$$\dot{\lambda}_q = -Ri_q - \omega_e \lambda_d + v_q, \tag{6.7}$$

$$\dot{\lambda}_d = -Ri_d + \omega_e \lambda_q + v_d, \tag{6.8}$$

where v_d, v_q are the d, q axis voltages, i_q, i_d are the d, q axis stator currents, λ_d, λ_q are the d, q axis stator flux linkages, R is the stator resistance and ω_e is the electromagnetic velocity. Furthermore, the electromagnetic torque developed and the velocity ω dynamics can be related as:

$$\tau_e = \frac{3}{2} P(\lambda_d i_q - \lambda_q i_d), \tag{6.9}$$

$$\dot{\omega} = \tau_e - \tau_L - b\omega, \tag{6.10}$$

where τ_L is some load torque, P is the number of pole pairs and b is the coefficient of viscous damping. In order to design controllers, one must first express these equations in terms of the d, q axis currents. It can be readily seen that once expressed in terms of the currents, the equivalent expressions for Eqs. (6.7), (6.8) will form the current loops and Eqs. (6.9), (6.10) would form the velocity control loops. This separation can be justified as the current response is much faster than the mechanical torque response.

In order to express Eqs. (6.7), (6.8) in terms of the currents, one must substitute the corresponding constitutive laws. Assuming linear inductance, we have

$$\begin{aligned} \lambda_q &= L_q i_q, \\ \lambda_d &= L_d i_d + \lambda_f, \end{aligned} \tag{6.11}$$

where L_q, L_d are the inductances and λ_f is the rotor flux linkage. However, this may not always be the case.

Our objective is to find a couple of functions (f_1, f_2) that are able to approximate the flux controlled by the current in each axis respectively:

$$\begin{aligned} \lambda_q &= f_1(I_q), \\ \lambda_d &= f_2(I_d), \end{aligned} \tag{6.12}$$

By substituting these nonlinear constitutive laws, we obtain the following equations:

$$\dot{I}_d = \left(\frac{\partial f_2}{\partial I_d}\right)^{-1} (-RI_d + \omega_e \lambda_q + v_d), \tag{6.13}$$

$$\dot{I}_q = \left(\frac{\partial f_1}{\partial I_q}\right)^{-1} (-RI_q - \omega_e \lambda_d + v_q). \tag{6.14}$$

In Section 6.2.2, we show that we can approximate the (f_1, f_2) pair using a Multi-Layer Perceptron (MLP) neural network whose architecture is designed with DNAS. As the electromagnetic flux (i.e., the ground truth) cannot be directly measured, the partial differential equations 6.13 and 6.14 can be used to compute an approximation of the flux.

6.2.2 . Proposed Approach

We divided our approach into two differentiable steps: **(i)** Firstly, searching for the optimal number of layers and **(ii)** Secondly, searching for the optimal activation function for the network structure found in step (i). This separation is motivated by preventing the numerical instability (i.e., the training loss becoming NaN) that would result from combining these two approaches into a single one.

We also had to design a surrogate criterion for training our neural network as the ground truth was not directly available.

Searching for the optimal number of layers

In this initial phase, we designed a compact search space denoted as S, encompassing solely two operations:

- `Linear` (fully-connected) + `ReLU`
- `Identity`

Here, `ReLU` served as a temporary representation, reserved for the forthcoming activation function uncovered through the process outlined in Section 6.2.2.

The differentiable exploration process closely follows the principles of DARTS [117] (detailed in Section 2.7 of Chapter 2). However, in contrast to the multiple cells with interconnected operations of DARTS, our objective involved identifying a solitary cell, comprising a linear sequence incorporating a maximum of N operations drawn from S. This cell interposed itself between the input and output fully-connected layers, thus determining the depth of our neural network. The formulation encompassed the creation of a collection of $2 \times N$ architectural parameters, collectively labeled as α_L. Each individual parameter within α_N corresponded to a specific candidate operation, denoted as o_i^L, in each respective layer L.

Remarkably, pursuing a single epoch sufficed to unveil a network depth that consistently delivered commendable performance levels. We detail the architecture of this network in Section 6.2.3.

Jacobian-based surrogate criterion

As presented in Section 6.2.1, it is not possible to directly obtain the electromagnetic flux values (λ_q, λ_d). However, it is possible to compute approximations by using Eqs. 6.13 and 6.14 as we have access to the ground truth values of $\dot{I}_d$ and $\dot{I}_q$.

Hence, we designed the following Jacobian-based surrogate criterion C to enable training:

$$C = MSE(J_M^{-1}(-RI + \omega_e M + v), (\dot{I}_d, \dot{I}_q)), \tag{6.15}$$

where MSE is the mean squared error, M is the model output vector such as $M = (\tilde{\lambda}_q, \tilde{\lambda}_d) \approx (\lambda_q, \lambda_d)$, R is the resistance value, I is the current vector such as $I = (I_d, I_q)$, v is the voltage vector such as $v = (v_d, v_q)$, and J_M is the Jacobian matrix of M w.r.t. I, such as:

$$J_M = \begin{pmatrix} \frac{\partial \tilde{\lambda}_q}{\partial I_q} & \frac{\partial \tilde{\lambda}_d}{\partial I_q} \\ \frac{\partial \tilde{\lambda}_q}{\partial I_d} & \frac{\partial \tilde{\lambda}_d}{\partial I_d} \end{pmatrix}. \tag{6.16}$$

Hence, minimizing C enforces the constraint that $M \approx (\lambda_q, \lambda_d)$.

Searching for the optimal activation function

In this second phase, we commenced by working from the network architecture unveiled in Section 6.2.2, and subsequently eliminated the provisional `ReLU` placeholder. Building upon this foundation, we then embarked on a differentiable process to discern the optimal activation function $\mathcal{A}^*$ that best suits this specific architectural configuration.

We define an activation function $\mathcal{A} \in S_{\mathcal{A}}$ as the amalgamation of a binary function $\mathcal{B}$ and two unary functions $(\mathcal{U}_1, \mathcal{U}_2)$, giving rise to $\mathcal{A}(x) = \mathcal{B}(\mathcal{U}_1(x), \mathcal{U}_2(x))$. Consequently, the core objective of this NAS endeavor rests in the identification of $\mathcal{A}^*(x) = \mathcal{B}^*(\mathcal{U}_1^*(x), \mathcal{U}_2^*(x))$. The search space $S_{\mathcal{A}}$ comprises the following binary primitives:

- $add(x, y) = x + y$
- $prod(x, y) = x \times y$
- $sub(x, y) = x - y$
- $max(x, y)$
- $min(x, y)$
- $mask(x, y) = x$

$S_{\mathcal{A}}$ also comprises the following unary functions:

- $max(x, 0)$
- $min(x, 0)$
- e^x

- $tanh(x)$
- x (`identity`)
- $|x|$
- $sin(x)$
- $cos(x)$
- 1
- x^2

Consequently, we formulated a set of $|B| + 2 \times |U|$ parameters, designated as $\alpha_{\mathcal{A}}$. Within a cell framework, these parameters undergo a `softmax` transformation, serving as weighted inputs for the mixed output of all unary and binary functions (see Eq. 2.7 and Section 2.7). This cell is integrated after each fully-connected layer, taking the place of the `ReLU` activation. Notably, since a solitary cell was employed, our exploration focused on determining a singular, global activation function for the entire network.

Drawing a parallel to Section 6.2.2 (and similarly to the optimization process of DARTS described in Section 2.7), we applied stochastic gradient descent over 20 epochs to optimize $\alpha_{\mathcal{A}}$ and consequently ascertain the optimal $\mathcal{A}^*$.

6.2.3 . Experiments

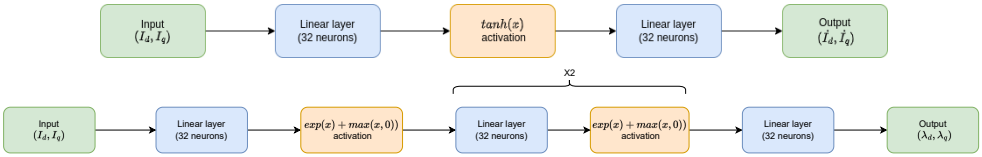


Figure 6.14: **Comparison between the baseline architecture (top) and the DNAS-optimized one (bottom)**. The DNAS-designed architecture is deeper than the original one and the activation function is also different and more complex.

In Fig. 6.14, we compared the original architecture with the one we obtained using differentiable NAS. The baseline architecture is a simple Multi-Layer Perceptron with only a single hidden layer and a standard `Tanh` activation. Our optimized architecture is deeper with two additional hidden layers and a totally different activation function $\mathcal{A}^*(x) = exp(x) + max(x, 0)$.

This newly-discovered architecture greatly overperforms the baseline architecture as shown in Fig. 6.15 where we can note that we get a much closer fit for the target flux linkages (λ_d, λ_q).

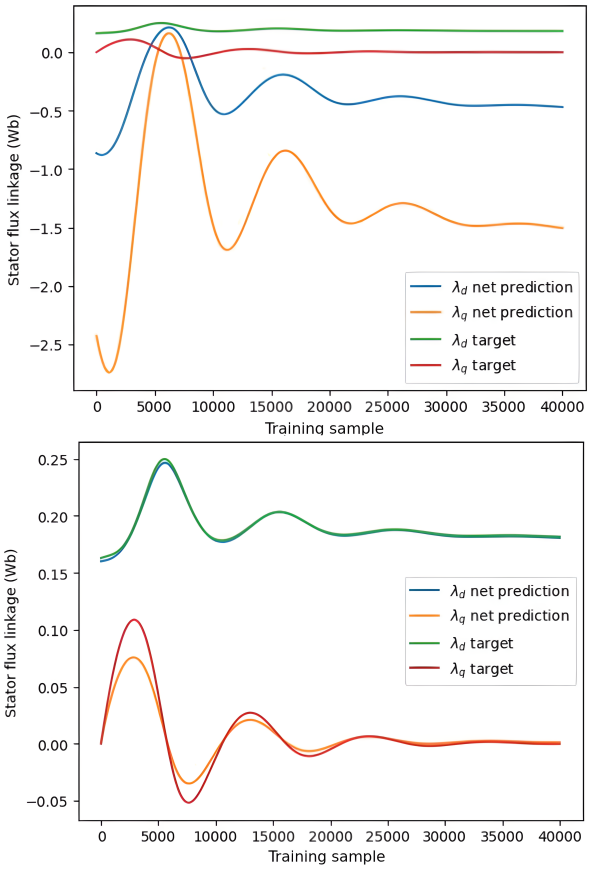


Figure 6.15: **Comparison between the flux linkages prediction plots of the baseline architecture (top) and the optimized architecture (bottom)**. The DNAS-based architecture fits the objective way closer than the baseline architecture.

6.2.4 . Discussion and Conclusion

We were able to leverage DNAS to design novel MLP architectures that perform significantly better than the baseline architecture. We evaluated our approach on simulation (with dummy data) but it still needs to be validated in a real-world scenario (i.e., using a physical motor). Our approach is also generic enough to be applied to other use cases and control theory problems where it could advantageously replace PID-based controllers.

7 - Conclusion and Future Directions

This thesis focused on exploring Differentiable Neural Architecture Search and its application to computer vision tasks. In Chapter 3, we saw that DNAS rapidly imposed itself as the prevailing subfield of NAS, with DARTS [117] becoming the most popular method by far. This high popularity led researchers to produce dozens of follow-up articles trying to tackle the four challenges we identified. However, none could overcome all challenges simultaneously, and no subsequent DNAS method could become the new standard in lieu of DARTS.

Nevertheless, in Chapters 4 to 6, we demonstrate that DNAS still has room for improvement and can be successfully applied to a wide range of topics, such as image classification, object recognition, pruning, content-based image retrieval, and self-supervised learning. DNAS-discovered architectures now consistently outperform handcrafted architectures, such as ResNets [69] or Xception [29].

However, it should be noted that not all of our efforts were successful. DNAS is a challenging research field, and some of our intuitions did not produce satisfactory results. For instance, we included in Appendix A an unpublished article that summarizes our efforts to create a biomimetic DNAS method we dubbed "ColorNAS". We explored color vision theory and highlighted that humans possessing a fourth type of photoreceptor (i.e., tetrachromats) can perceive a weirder color gamut than standard trichromat subjects. In addition, recent color vision research shows that the human brain processes the stimuli from each type of photoreceptor using separate neural pipelines and recomposes the final "mental" image by merging the processed signals. From there, we made the assumption that we could roughly assimilate the color spectrum perceived by each type of photoreceptor to a channel. Following this intuition, we designed an approach that would construct an ensemble model composed of DNAS-designed subnetworks that would each be responsible for processing a single channel. Unfortunately, despite promising initial results on CIFAR-10, we could not generalize these results to the more challenging CIFAR-100. Nonetheless, this failure indicates that pursuing biomimetic designs is not always advisable and highlights the disparities between computer vision and biological optical vision.

Furthermore, a drastic change is currently underway in the computer vision field with the gradual replacement of Convolutional Neural Networks (CNNs) with Vision Transformers (ViTs). ViTs perform significantly better than CNNs, especially when the task at hand involves learning from a very large dataset (e.g., DeiT III [178] outperforms ResNet-50 by around 6 % in top-1 accuracy on ImageNet [163]). However, ViTs are often challenging to train and

suffer from high inference latency due to their large size. Therefore, as shown in Chapter 6, DNAS (and NAS in general) can be useful for performing automatic pruning on ViTs or discovering new highly-optimized architectures capable of addressing these issues.

Additionally, Explainable AI (XAI) is paramount to ensure the deployment of Deep Learning in everyday tasks. In the case of automated decision making (e.g., judicial case analysis or autonomous driving), people affected by those decisions would have trouble accepting them unless the model is able to provide argumentation [7]. Furthermore, some governmental entities such as the European Union plan to require by law that AI models deployed on the market should be explainable [47]. Concerning NAS, only a few approaches have been proposed to directly search for explainable CNN models. However, most of the existing studies leverage evolutionary-based [17, 3] or one-shot probabilistic NAS [116, 143]. To the extent of our knowledge, none targeted DNAS specifically. Nonetheless, one method dubbed SNAS (Saliency-Aware NAS) [78] is noteworthy as it is post-hoc (i.e., agnostic of the backbone) and can compute saliency maps for a wide range of NAS algorithms, including DNAS methods such as DARTS [117]. Thus, Explainable NAS (XNAS) is still an emerging field, and we predict that it will expand greatly in the near future when DNAS methods overcome all their current challenges.

Moreover, DNAS, along with NAS and general and the whole Automated Machine Learning (AutoML) ecosystem, could play a key role in solving the alignment problem [146, 195], which arises when an autonomous system does not match the specific requirements or goal of a given task (it is thus "misaligned"). In the case of a neural network, this leads to suboptimal performance and global inefficiency. DNAS addresses this issue by efficiently exploring a vast space of architectural configurations and identifying network structures that are better suited for the task's objective function. This way, it is possible to find an architecture that can help constrain the neural network to its objective, preventing possible deviations. Hence, its flexibility and adaptability make DNAS a powerful tool to address the alignment problem and enhance the capabilities of neural networks in diverse tasks.

In addition to the alignment problem, DNAS is also very relevant in the current context of ecological transition. The excellent pattern recognition performance of deep neural networks comes at the price of a high carbon footprint that raises concerns about their sustainability [102, 59]. Consequently, it is necessary to find ways to reduce their computational cost and lower their energy consumption. DNAS can be a useful tool to contribute to that goal by designing efficient sparse architectures (i.e., that do not comport any superfluous operations). Ideally, those architectures would retain the performance level of larger architectures despite having a low number of FLOPs, similar to what we achieved in Chapter 6. It is noteworthy to state that some approaches

went further than this and already tried to assess the carbon footprint of NAS itself [226].

Finally, novel trends have recently appeared in the NAS landscape, with promising non-DNAS approaches such as zero-shot (or low-cost) NAS [135, 21, 2]. These works focused on methods able to perform NAS very efficiently (e.g., just using a few minutes of computational time), thus comparing favorably with DNAS which still requires several GPU hours. We also witnessed the revival of previously explored concepts such as Reinforcement Learning or Evolutionary Algorithms [38]. Consequently, we can expect that, in the near future, DNAS will benefit from knowledge/experience transfer in NAS methods that could combine the strengths of several approaches.

Bibliography

[1] Mohamed S Abdelfattah et al. “Zero-Cost Proxies for Lightweight {NAS}”. In: *International Conference on Learning Representations*. 2021.

[2] Mohamed S Abdelfattah et al. “Zero-Cost Proxies for Lightweight NAS”. In: *International Conference on Learning Representations*. 2021.

[3] Andrea Agiollo, Giovanni Ciatto, and Andrea Omicini. “Shallow2Deep: Restraining Neural Networks Opacity Through Neural Architecture Search”. In: *International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems*. Springer. 2021, pp. 63–82.

[4] Shunichi Amari. “A theory of adaptive pattern classifiers”. In: *IEEE Transactions on Electronic Computers* 3 (1967), pp. 299–307.

[5] D Anderson and K Burnham. “Model selection and multi-model inference”. In: *Second. NY: Springer-Verlag* 63.2020 (2004), p. 10.

[6] M Arai. “Adaptive control of a neural network with a variable function of a unit and its application”. In: *Trans. Inst. Electron, Inform. Commun. Engng* 74 (1991), pp. 551–559.

[7] Alejandro Barredo Arrieta et al. “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI”. In: *Information fusion* 58 (2020), pp. 82–115.

[8] Werner G. K. Backhaus et al. *Color Vision: Perspectives from Different Disciplines*. Walter de Gruyter, 1998.

[9] Gabriel Bender et al. “Understanding and simplifying one-shot architecture search”. In: *International conference on machine learning*. PMLR. 2018, pp. 550–559.

[10] Yassine Benyahia et al. “Overcoming multi-model forgetting”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 594–603.

[11] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[12] Leo Breiman. “Arcing classifier (with discussion and a rejoinder by the author)”. In: *The annals of statistics* 26.3 (1998), pp. 801–849.

[13] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[14] Jane Bromley et al. “Signature verification using a" siamese" time delay neural network”. In: *Advances in neural information processing systems* 6 (1993).

[15] Han Cai, Ligeng Zhu, and Song Han. “ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware”. In: *International Conference on Learning Representations*. 2019.

[16] Han Cai et al. “Once for All: Train One Network and Specialize it for Efficient Deployment”. In: *International Conference on Learning Representations*. 2020.

[17] Zachariah J Carmichael, Timothy Y Moon, and Samson A Jacobs. *Explainable Neural Architecture Search (XNAS)*. Tech. rep. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2021.

[18] Chyi-Tsong Chen and Wei-Der Chang. “A feedforward neural network with function shape autotuning”. In: *Neural networks* 9.4 (1996), pp. 627–641.

[19] Daoyuan Chen et al. “AdaBERT: task-adaptive BERT compression with differentiable neural architecture search”. In: *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 2021, pp. 2463–2469.

[20] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[21] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. “Neural Architecture Search on ImageNet in Four GPU Hours: A Theoretically Inspired Perspective”. In: *International Conference on Learning Representations (ICLR)*. 2021.

[22] Xiangning Chen and Cho-Jui Hsieh. “Stabilizing differentiable architecture search via perturbation-based regularization”. In: *International conference on machine learning*. PMLR. 2020, pp. 1554–1565.

[23] Xin Chen et al. “Progressive differentiable architecture search: Bridging the depth gap between search and evaluation”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1294–1303.

[24] Xinlei Chen and Kaiming He. “Exploring simple siamese representation learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 15750–15758.

[25] Xinlei Chen et al. “Improved baselines with momentum contrastive learning”. In: *arXiv preprint arXiv:2003.04297* (2020).

[26] Yukang Chen et al. “Renas: Reinforced evolutionary neural architecture search”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4787–4796.

[27] Krishna Teja Chitty-Venkata et al. “Neural architecture search for transformers: A survey”. In: *IEEE Access* 10 (2022), pp. 108374–108412.

[28] François Chollet. “On the measure of intelligence”. In: *arXiv preprint arXiv:1911.01547* (2019).

[29] François Chollet. “Xception: Deep learning with depthwise separable convolutions”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[30] Xiangxiang Chu and Bo Zhang. “Noisy Differentiable Architecture Search”. In: *British Machine Vision Conference*. 2021.

[31] Xiangxiang Chu et al. “DARTS-: Robustly Stepping out of Performance Collapse Without Indicators”. In: *International Conference on Learning Representations*. 2021.

[32] Xiangxiang Chu et al. “Fair darts: Eliminating unfair advantages in differentiable architecture search”. In: *European conference on computer vision*. Springer. 2020, pp. 465–480.

[33] *CIFAR-10 and CIFAR-100 datasets*. https://www.cs.toronto.edu/~kriz/cifar.html. Accessed: 2023-09-06.

[34] Karl Cobbe et al. “Leveraging procedural generation to benchmark reinforcement learning”. In: *International conference on machine learning*. PMLR. 2020, pp. 2048–2056.

[35] Marius Cordts et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[36] Ekin Dogus Cubuk et al. “AutoAugment: Learning Augmentation Policies from Data”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* ().

[37] Haskell B Curry. “The method of steepest descent for non-linear minimization problems”. In: *Quarterly of Applied Mathematics* 2.3 (1944), pp. 258–261.

[38] Xiaoliang Dai et al. “FBNetV3: Joint architecture-recipe search using predictor pretraining”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 16276–16285.

[39] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[40] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[41] Terrance DeVries and Graham W Taylor. “Improved Regularization of Convolutional Neural Networks with Cutout”. In: *arXiv preprint arXiv:1708.04552* (2017).

[42] Terrance DeVries and Graham W Taylor. “Improved regularization of convolutional neural networks with cutout”. In: *arXiv preprint arXiv:1708.04552* (2017).

[43] Shaojin Ding et al. “Autospeech: Neural architecture search for speaker recognition”. In: *arXiv preprint arXiv:2005.03215* (2020).

[44] Hongwei Dong et al. “Automatic design of CNNs via differentiable neural architecture search for PolSAR image classification”. In: *IEEE Transactions on Geoscience and Remote Sensing* 58.9 (2020), pp. 6362–6375.

[45] Xuanyi Dong and Yi Yang. “NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search”. In: *International Conference on Learning Representations (ICLR)*. 2020. url: https://openreview.net/forum?id=HJxyZkBKDr.

[46] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations*. 2021. url: https://openreview.net/forum?id=YicbFdNTTy.

[47] Martin Ebers. “Regulating Explainable AI in the European Union. An Overview of the Current Legal Framework (s)”. In: *An Overview of the Current Legal Framework (s)(August 9, 2021). Liane Colonna/Stanley Greenstein (eds.), Nordic Yearbook of Law and Informatics* (2020).

[48] Thomas Ebrey and Yiannis Koutalos. “Vertebrate Photoreceptors”. In: *Progress in Retinal and Eye Research* 20.1 (2001), pp. 49–94. issn: 1350-9462.

[49] Bradley Efron. “Bootstrap methods: another look at the jackknife”. In: *Breakthroughs in statistics*. Springer, 1992, pp. 569–593.

[50] Lasse Espeholt et al. “Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures”. In: *International conference on machine learning*. PMLR. 2018, pp. 1407–1416.

[51] Mark Everingham and John Winn. “The PASCAL visual object classes challenge 2012 (VOC2012) development kit”. In: *Pattern Anal. Stat. Model. Comput. Learn., Tech. Rep* 2007 (2012), pp. 1–45.

[52] Zhenkun Fan et al. “Self-attention neural architecture search for semantic image segmentation”. In: *Knowledge-Based Systems* 239 (2022), p. 107968.

[53] Jiemin Fang et al. “Densely connected search space for more flexible neural architecture search”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10628–10637.

[54] Jonathan Frankle and Michael Carbin. “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks”. In: *International Conference on Learning Representations*. 2019.

[55] Yoav Freund, Robert Schapire, and Naoki Abe. “A short introduction to boosting”. In: *Journal-Japanese Society For Artificial Intelligence* 14.771-780 (1999), p. 1612.

[56] Kunihiko Fukushima. “Visual feature extraction by a multilayered network of analog threshold elements”. In: *IEEE Transactions on Systems Science and Cybernetics* 5.4 (1969), pp. 322–333.

[57] Chen Gao et al. “Adversarialnas: Adversarial neural architecture search for gans”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5680–5689.

[58] Jian Gao et al. “A Practical Analytical Expression and Estimation for Average Torque of High Saturation Permanent Magnet Synchronous Motor for Special Vehicles”. In: *IEEE Transactions on Vehicular Technology* 72.1 (2022), pp. 357–366.

[59] Stefanos Georgiou et al. “Green ai: Do deep learning frameworks have different costs?” In: *Proceedings of the 44th International Conference on Software Engineering*. 2022, pp. 1082–1094.

[60] Yuanbiao Gou et al. “Clearer: Multi-scale neural architecture search for image restoration”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 17129–17140.

[61] Klaus Greff, Rupesh K Srivastava, and Jürgen Schmidhuber. “Highway and Residual Networks learn Unrolled Iterative Estimation”. In: *International Conference on Learning Representation*. 2017.

[62] Sorin Grigorescu et al. “A survey of deep learning techniques for autonomous driving”. In: *Journal of Field Robotics* 37.3 (2020), pp. 362–386.

[63] Jean-Bastien Grill et al. “Bootstrap your own latent-a new approach to self-supervised learning”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21271–21284.

[64] Yu-Chao Gu et al. “Dots: Decoupling operation and topology in differentiable architecture search”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 12311–12320.

[65] Dazhou Guo et al. “Organ at risk segmentation for head and neck cancer using stratified learning and neural architecture search”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 4223–4232.

[66] Richard W Hamming. “Error detecting and error correcting codes”. In: *The Bell system technical journal* 29.2 (1950), pp. 147–160.

[67] W Keith Hastings. “Monte Carlo sampling methods using Markov chains and their applications”. In: (1970).

[68] Felix Hausdorff. *Grundzüge der Mengenlehre*. Leipzig: Veit, 1914.

[69] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[70] Kaiming He et al. “Mask r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[71] Kaiming He et al. “Masked Autoencoders Are Scalable Vision Learners”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 16000–16009.

[72] Hermann von Helmholtz. *Treatise on Physiological Optics*. Courier Corporation, 1909. isbn: 978-0486174709.

[73] Dan Hendrycks and Kevin Gimpel. “Gaussian error linear units (gelus)”. In: *arXiv preprint arXiv:1606.08415* (2016).

[74] Alexandre Heuillet, Fabien Couthouis, and Natalia Diaz-Rodriguez. “Explainability in deep reinforcement learning”. In: *Knowledge-Based Systems* 214 (2021), p. 106685.

[75] Alexandre Heuillet, Hedi Tabia, and Hichem Arioui. *NASiam: Efficient Representation Learning using Neural Architecture Search for Siamese Networks*. 2023. url: https://openreview.net/forum?id=apZRm_0VClK.

[76] Alexandre Heuillet et al. “D-DARTS: Distributed Differentiable Architecture Search”. In: *arXiv preprint arXiv:2108.09306* (2021).

[77] Sung-Jin Hong and Oh-Kyong Kwon. “An RGB to RGBY Color Conversion Algorithm for Liquid Crystal Display Using RGW Pixel with Two-Field Sequential Driving Method”. In: *Journal of the Optical Society of Korea* 6.6 (Dec. 2014).

[78] Ramtin Hosseini and Pengtao Xie. “Saliency-Aware Neural Architecture Search”. In: *Advances in Neural Information Processing Systems*. 2022.

[79] Andrew Howard et al. “Searching for mobilenetv3”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1314–1324.

[80] Shoukang Hu et al. “Neural architecture search for LF-MMI trained time delay neural networks”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 30 (2022), pp. 1093–1107.

[81] Han Huang et al. “Lightweight image super-resolution with hierarchical and differentiable neural architecture search”. In: *arXiv preprint arXiv:2105.03939* (2021).

[82] Lan Huang et al. “U-DARTS: Uniform-space differentiable architecture search”. In: *Information Sciences* 628 (2023), pp. 339–349.

[83] Ziyan Huang, Zehua Wang, Lixu Gu, et al. “AdwU-Net: Adaptive Depth and Width U-Net for Medical Image Segmentation by Differentiable Neural Architecture Search”. In: *Medical Imaging with Deep Learning*. 2021.

[84] David H Hubel and Torsten N Wiesel. “Receptive fields and functional architecture of monkey striate cortex”. In: *The Journal of physiology* 195.1 (1968), pp. 215–243.

[85] “IEEE Standard for Floating-Point Arithmetic”. In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), pp. 1–84. doi: 10.1109/IEEESTD.2019.8766229.

[86] *INRIA Holidays Dataset*. https://lear.inrialpes.fr/~jegou/data.php. Accessed: 2023-09-07.

[87] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[88] A.G. Ivakhnenko and V.G. Lapa. *Cybernetic Predicting Devices*. Jprs report. CCM Information Corporation, 1973. url: https://books.google.fr/books?id=FhwVNQAACAAJ.

[89] Gerald H. Jacobs. "The evolution of vertebrate color vision". In: *Advances in experimental medicine and biology* 739 (2012), pp. 156–172.

[90] K. A. Jameson, S.M. Highnote, and L.M. Wasserman. "Richer color experience in observers with multiple photopigment opsin genes". In: *Psychonomic Bulletin & Review* 8 (2001), pp. 244–261.

[91] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical Reparametrization with Gumble-Softmax". In: *International Conference on Learning Representations*. 2017.

[92] Herve Jegou, Matthijs Douze, and Cordelia Schmid. "Hamming Embedding and Weak Geometric Consistency for Large Scale Image Search". In: *Proceedings of the 10th European Conference on Computer Vision: Part I*. 2008, pp. 304–317.

[93] Qian Jiang et al. "EH-DNAS: End-to-End Hardware-aware Differentiable Neural Architecture Search". In: *arXiv preprint arXiv:2111.12299* (2021).

[94] Yufan Jiang et al. "Improved differentiable architecture search for language modeling and named entity recognition". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 3585–3590.

[95] Kun Jing, Luoyu Chen, and Jungang Xu. "An architecture entropy regularizer for differentiable neural architecture search". In: *Neural Networks* 158 (2023), pp. 111–120.

[96] Gabriele Jordan et al. "The dimensionality of color vision in carriers of anomalous trichromacy". In: *Journal of Vision* 10.8 (July 2010), pp. 12–12. issn: 1534-7362.

[97] Amina Kammoun et al. "Generative Adversarial Networks for face generation: A survey". In: *ACM Computing Surveys* 55.5 (2022), pp. 1–37.

[98] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *Proceedings of naacL-HLT*. 2019, pp. 4171–4186.

[99] Sunghoon Kim et al. "MDARTS: Multi-objective differentiable neural architecture search". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2021, pp. 1344–1349.

[100] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[101] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems* 25 (2012).

[102] C-C Jay Kuo and Azad M Madni. "Green learning: Introduction, examples and outlook". In: *Journal of Visual Communication and Image Representation* (2022), p. 103685.

[103] Yann LeCun. "A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27". In: (2022).

[104] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[105] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[106] Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[107] Claude Lemaréchal. "Cauchy and the gradient method". In: *Doc Math Extra* 251.254 (2012), p. 10.

[108] Qing Li, Xia Wu, and Tianming Liu. "Differentiable neural architecture search for optimal spatial/temporal brain function network decomposition". In: *Medical Image Analysis* 69 (2021), p. 101974.

[109] Shiqian Li et al. "Auto-FERNet: A facial expression recognition network with architecture search". In: *IEEE Transactions on Network Science and Engineering* 8.3 (2021), pp. 2213–2222.

[110] Zhiheng Li et al. "Differentiable neural architecture search for sar image ship object detection". In: *IET International Radar Conference (IET IRC 2020)*. Vol. 2020. IET. 2020, pp. 950–954.

[111] Hanwen Liang et al. "Darts+: Improved differentiable architecture search with early stopping". In: *arXiv preprint arXiv:1909.06035* (2019).

[112] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[113] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[114] Seppo Linnainmaa. "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". PhD thesis. Master's Thesis (in Finnish), Univ. Helsinki, 1970.

[115] Chenxi Liu et al. “Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 82–92.

[116] Chia-Hsiang Liu et al. “FOX-NAS: Fast, On-device and Explainable Neural Architecture Search”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 789–797.

[117] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “DARTS: Differentiable Architecture Search”. In: *International Conference on Learning Representations*. 2019.

[118] Ze Liu et al. “Swin transformer: Hierarchical vision transformer using shifted windows”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 10012–10022.

[119] Vasco Lopes et al. “Guided Evolution for Neural Architecture Search”. In: *arXiv preprint arXiv:2110.15232* (2021).

[120] Javier Garcia Lopez, Antonio Agudo, and Francesc Moreno-Noguer. “E-DNAS: Differentiable neural architecture search for embedded systems”. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 4704–4711.

[121] Jonathan Lorraine, Paul Vicol, and David Duvenaud. “Optimizing millions of hyperparameters by implicit differentiation”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 1540–1552.

[122] Qing Lu et al. “RT-DNAS: Real-Time Constrained Differentiable Neural Architecture Search for 3D Cardiac Cine MRI Segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2022, pp. 602–612.

[123] S.M. Lucas et al. “ICDAR 2003 Robust Reading Competitions”. In: *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* 2003, pp. 682–687. doi: 10.1109/ICDAR.2003.1227749.

[124] Scott M Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Proceedings of the International Conference on Neural Information Processing Systems*. 2017, pp. 4768–4777.

[125] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in neural information processing systems* 30 (2017).

[126] Xiangzhong Luo et al. “Lightnas: On lightweight and scalable neural architecture search for embedded platforms”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).

[127] Eran Malach et al. “Proving the Lottery Ticket Hypothesis: Pruning Is All You Need”. In: *Proceedings of the 37th International Conference on Machine Learning*. PMLR, Nov. 2020, pp. 6682–6691. (Visited on 06/02/2023).

[128] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. “Building a Large Annotated Corpus of English: The Penn Treebank”. In: *Comput. Linguist.* 19.2 (June 1993), pp. 313–330. issn: 0891-2017.

[129] Louis Martin et al. “CamemBERT: a Tasty French Language Model”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 7203–7219.

[130] Seyed Mojtaba Marvasti-Zadeh et al. “CHASE: Robust Visual Tracking via Cell-Level Differentiable Neural Architecture Search”. In: *British Machine Vision Conference*. 2021.

[131] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[132] Sachin Mehta and Mohammad Rastegari. “MobileViT: Light-weight, General-Purpose, and Mobile-Friendly Vision Transformer”. In: *International Conference on Learning Representations*. 2022.

[133] Sachin Mehta and Mohammad Rastegari. “MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer”. In: *International Conference on Learning Representations*. 2021.

[134] Sachin Mehta and Mohammad Rastegari. “Separable self-attention for mobile vision transformers”. In: *arXiv preprint arXiv:2206.02680* (2022).

[135] Joe Mellor et al. “Neural architecture search without training”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 7588–7598.

[136] Stephen Merity et al. “Pointer Sentinel Mixture Models”. In: *International Conference on Learning Representations*. 2017. url: https://openreview.net/forum?id=Byj72udxe.

[137] R Meyes et al. “Ablation studies to uncover structure of learned representations in artificial neural networks”. In: *Proceedings on the International Conference on Artificial Intelligence*. The Steering Committee of The World Congress in Computer Science, Computer … 2019, pp. 185–191.

[138] Yingjie Miao et al. “RL-DARTS: differentiable architecture search for reinforcement learning”. In: *arXiv preprint arXiv:2106.02229* (2021).

[139] Paulius Micikevicius et al. “Mixed Precision Training”. In: *International Conference on Learning Representations*. 2018.

[140] Microsoft. *Neural Network Intelligence*. Version 2.0. Jan. 2021. url: https://github.com/microsoft/nni.

[141] *Microsoft Common Objects in Context*. https://cocodataset.org/#home. Accessed: 2023-09-06.

[142] Tong Mo et al. “Neural Architecture Search for Keyword Spotting”. In: *Proc. Interspeech 2020* (2020), pp. 1982–1986.

[143] Niv Nayman et al. “BINAS: Bilinear Interpretable Neural Architecture Search”. In: *arXiv preprint arXiv:2110.12399* (2021).

[144] Niv Nayman et al. “HardCoRe-NAS: hard constrained differentiable neural architecture search”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 7979–7990.

[145] Yuval Netzer et al. “Reading digits in natural images with unsupervised feature learning”. In: (2011).

[146] Richard Ngo. “The alignment problem from a deep learning perspective”. In: *arXiv preprint arXiv:2209.00626* (2022).

[147] Asaf Noy et al. “Asap: Architecture search, anneal and prune”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 493–503.

[148] David Opitz and Richard Maclin. “Popular ensemble methods: An empirical study”. In: *Journal of artificial intelligence research* 11 (1999), pp. 169–198.

[149] Ankur Parikh et al. “A Decomposable Attention Model for Natural Language Inference”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016, pp. 2249–2255.

[150] Abhishek Patkar and Anuradha M Annaswamy. “An adaptive controller for a class of nonlinear plants based on neural networks and convex parameterization”. In: *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE. 2020, pp. 126–131.

[151] Karl Pearson. “Note on Regression and Inheritance in the Case of Two Parents”. In: *Proceedings of the Royal Society of London Series I* 58 (Jan. 1895), pp. 240–242.

[152] Cheng Peng et al. “Efficient convolutional neural architecture search for remote sensing image scene classification”. In: *IEEE Transactions on Geoscience and Remote Sensing* 59.7 (2020), pp. 6092–6105.

[153] Cheng Peng et al. “ReCNAS: Resource-Constrained Neural Architecture Search Based on Differentiable Annealing and Dynamic Pruning”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[154] Junran Peng et al. “Efficient neural architecture transformation search in channel-level for object detection”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[155] Wei Peng, Xiaopeng Hong, and Guoying Zhao. “Video action recognition via neural architecture searching”. In: *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2019, pp. 11–15.

[156] Matthew E. Peters et al. “Deep contextualized word representations”. In: *Proceedings of the NAACL*. 2018.

[157] Pragasen Pillay and Ramu Krishnan. “Modeling, simulation, and analysis of permanent-magnet motor drives. I. The permanent-magnet synchronous motor drive”. In: *IEEE Transactions on industry applications* 25.2 (1989), pp. 265–273.

[158] Esteban Real et al. “Regularized evolution for image classifier architecture search”. In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 4780–4789.

[159] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Laboratory. Report no. VG-1196-G-8. Spartan Books, 1962. url: https://books.google.fr/books?id=7FhRAAAAMAAJ.

[160] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65 6 (1958), pp. 386–408.

[161] Michael Ruchte et al. *NASLib: A Modular and Flexible Neural Architecture Search Library*. https://github.com/automl/NASLib. 2020.

[162] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[163] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. doi: 10.1007/s11263-015-0816-y.

[164] Mark Sandler et al. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[165] Jürgen Schmidhuber. “Learning to control fast-weight memories: An alternative to dynamic recurrent networks”. In: *Neural Computation* 4.1 (1992), pp. 131–139.

[166] Lloyd S Shapley. “A value for n-person games”. In: *Classics in game theory* 69 (1997).

[167] Herbert A Simon. “The new science of management decision.” In: (1960).

[168] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[169] Shaden Smith et al. “Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model”. In: *arXiv preprint arXiv:2201.11990* (2022).

[170] C. Spearman. “The Proof and Measurement of Association between Two Things”. In: *The American Journal of Psychology* 15.1 (1904), pp. 72–101. issn: 00029556. url: http://www.jstor.org/stable/1412159 (visited on 05/23/2023).

[171] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[172] Haozhe Sun et al. “RRR-Net: Reusing, Reducing, and Recycling a Deep Backbone Network”. In: *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2023, pp. 1–9.

[173] Gunnar Svaetichin. “Spectral response curves from single cones”. In: *Acta physiologica Scandinavica. Supplementum* 39.154 (1956), pp. 17–46.

[174] Mingxing Tan and Quoc Le. “Efficientnetv2: Smaller models and faster training”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10096–10106.

[175] Hidenori Tanaka et al. “Pruning Neural Networks without Any Data by Iteratively Conserving Synaptic Flow”. In: *Advances in neural information processing systems* 33 (2020), pp. 6377–6389.

[176] *The data that transformed AI research—and possibly the world*. https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world. Accessed: 2023-09-06.

[177] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[178] Hugo Touvron, Matthieu Cord, and Hervé Jégou. "Deit iii: Revenge of the vit". In: *European Conference on Computer Vision*. Springer. 2022, pp. 516–533.

[179] Hugo Touvron et al. "Training data-efficient image transformers & distillation through attention". In: *International conference on machine learning*. PMLR. 2021, pp. 10347–10357.

[180] Martin J. Tovée. *An introduction to the visual system*. Cambridge University Press, 2008.

[181] Alan M. Turing. *The imitation game*. 1950.

[182] Maria Tzelepi and Anastasios Tefas. "Deep convolutional learning for content based image retrieval". In: *Neurocomputing* 275 (2018), pp. 2467–2478.

[183] Ihsan Ullah et al. "Meta-Album: Multi-domain Meta-Dataset for Few-Shot Image Classification". In: *Thirty-Sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2022.

[184] Arash Vahdat et al. "UNAS: Differentiable Architecture Search Meets Reinforcement Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[185] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[186] J. Von Neumann, P.M. Churchland, and P.S. Churchland. *The Computer and the Brain*. The Silliman Memorial Lectures Series. Yale University Press, 2000. isbn: 9780300084733. url: https://books.google.fr/books?id=Q30MqJjRv1gC.

[187] Alvin Wan et al. "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 12965–12974.

[188] Ji Wan et al. “Deep learning for content-based image retrieval: A comprehensive study”. In: *Proceedings of the 22nd ACM international conference on Multimedia*. 2014, pp. 157–166.

[189] Ruochen Wang et al. “Rethinking Architecture Selection in Differentiable NAS”. In: *International Conference on Learning Representation*. 2021.

[190] Wenna Wang et al. “FP-DARTS: Fast parallel differentiable neural architecture search for image classification”. In: *Pattern Recognition* 136 (2023), p. 109193.

[191] Yaoming Wang et al. “Learning latent architectural distribution in differentiable neural architecture search via variational information maximization”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 12312–12321.

[192] Yu Weng, Zehua Chen, and Tianbao Zhou. “Improved differentiable neural architecture search for single image super-resolution”. In: *Peer-to-Peer Networking and Applications* 14.3 (2021), pp. 1806–1815.

[193] Yu Weng et al. “Nas-unet: Neural architecture search for medical image segmentation”. In: *IEEE Access* 7 (2019), pp. 44247–44257.

[194] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3 (1992), pp. 229–256.

[195] Yotam Wolf et al. “Fundamental limitations of alignment in large language models”. In: *arXiv preprint arXiv:2304.11082* (2023).

[196] David H. Wolpert. “Stacked generalization”. In: *Neural Networks* 5.2 (1992), pp. 241–259. issn: 0893-6080.

[197] Bichen Wu et al. “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10734–10742.

[198] Bichen Wu et al. “Fbnetv5: Neural architecture search for multiple tasks in one run”. In: *arXiv preprint arXiv:2111.10007* (2021).

[199] Dongxian Wu, Yisen Wang, and Shu-tao Xia. “Revisiting loss landscape for adversarial robustness”. In: *arXiv preprint arXiv:2004.05884* (2020).

[200] Yan Wu et al. “Trilevel neural architecture search for efficient single image super-resolution”. In: *arXiv preprint arXiv:2101.06658* (2021).

[201] Gerard Jacques van Wyk and Anna Sergeevna Bosman. "Evolutionary neural architecture search for image restoration". In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

[202] Saining Xie et al. "Aggregated residual transformations for deep neural networks". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.

[203] Sirui Xie et al. "SNAS: stochastic neural architecture search". In: *arXiv preprint arXiv:1812.09926* (2018).

[204] Jingjing Xu et al. "A survey on green deep learning". In: *arXiv preprint arXiv:2111.05193* (2021).

[205] Yuhui Xu et al. "PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search". In: *International Conference on Learning Representations*. 2020.

[206] Yu-Bai Yan et al. "Torque estimation and control of PMSM based on deep learning". In: *2019 22nd International Conference on Electrical Machines and Systems (ICEMS)*. IEEE. 2019, pp. 1–6.

[207] Zheyu Yan et al. "RADARS: Memory Efficient Reinforcement Learning Aided Differentiable Neural Architecture Search". In: *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2022, pp. 128–133.

[208] Tien-Ju Yang, Yi-Lun Liao, and Vivienne Sze. "Netadaptv2: Efficient neural architecture search with fast super-network training and architecture optimization". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2402–2411.

[209] Yibo Yang et al. "Towards improving the consistency, efficiency, and flexibility of differentiable neural architecture search". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 6667–6676.

[210] Peng Ye et al. "$beta$-DARTS: Beta-Decay Regularization for Differentiable Architecture Search". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 10874–10883.

[211] Xue Ying. "An overview of overfitting and its solutions". In: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing. 2019, p. 022022.

[212] Hongyuan Yu et al. “Cyclic differentiable architecture search”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[213] Jiahui Yu and Thomas S Huang. “Universally slimmable networks and improved training techniques”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1803–1811.

[214] Jiahui Yu et al. “Bignas: Scaling up neural architecture search with big single-stage models”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*. Springer. 2020, pp. 702–717.

[215] Qihang Yu et al. “C2fnas: Coarse-to-fine neural architecture search for 3d medical image segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 4126–4135.

[216] Ahmet Caner Yüzügüler, Nikolaos Dimitriadis, and Pascal Frossard. “U-Boost NAS: Utilization-Boosted Differentiable Neural Architecture Search”. In: *arXiv preprint arXiv:2203.12412* (2022).

[217] Arber Zela et al. “Understanding and Robustifying Differentiable Architecture Search”. In: *International Conference on Learning Representations*. 2020.

[218] Hanfeng Zhai and Timothy Sands. “Comparison of Deep Learning and Deterministic Algorithms for Control Modeling”. In: *Sensors* 22.17 (2022), p. 6362.

[219] Haokui Zhang et al. “Memory-efficient hierarchical neural architecture search for image denoising”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 3657–3666.

[220] Kaiyu Zhang et al. “Differentiable neural architecture search augmented with pruning and multi-objective optimization for time-efficient intelligent fault diagnosis of machinery”. In: *Mechanical Systems and Signal Processing* 158 (2021), p. 107773.

[221] Li Lyna Zhang et al. “Fast hardware-aware neural architecture search”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 692–693.

[222] Miao Zhang et al. “Differentiable neural architecture search in equivalent space with exploration enhancement”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 13341–13351.

[223] Miao Zhang et al. “idarts: Differentiable architecture search with stochastic implicit gradients”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12557–12566.

[224] Peng Zhang et al. “Self-trained target detection of radar and sonar images using automatic deep learning”. In: *IEEE Transactions on Geoscience and Remote Sensing* 60 (2021), pp. 1–14.

[225] Wenqiang Zhang et al. “Efficientpose: Efficient human pose estimation with neural architecture search”. In: *Computational Visual Media* 7.3 (2021), pp. 335–347.

[226] Yiyang Zhao and Tian Guo. “Carbon-Efficient Neural Architecture Search”. In: *arXiv preprint arXiv:2307.04131* (2023).

[227] Huahuan Zheng, Keyu An, and Zhijian Ou. “Efficient neural architecture search for end-to-end speech recognition via straight-through gradients”. In: *2021 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2021, pp. 60–67.

[228] Bolei Zhou et al. “Scene parsing through ade20k dataset”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 633–641.

[229] Pan Zhou et al. “Theory-inspired path-regularized differential network architecture search”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 8296–8307.

[230] Zhuotun Zhu et al. “V-NAS: Neural architecture search for volumetric medical image segmentation”. In: *2019 International conference on 3d vision (3DV)*. IEEE. 2019, pp. 240–248.

[231] Barret Zoph and Quoc Le. “Neural Architecture Search with Reinforcement Learning”. In: *International Conference on Learning Representations*. 2017.

[232] Barret Zoph et al. “Learning transferable architectures for scalable image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710.

Appendices

A - ColorNAS

A.1 . Introduction

Color vision is an important aspect of visual perception and is widespread among animals, including humans. It plays a crucial role in allowing organisms to accurately interpret and analyze their visual environment. Biologists and neuroscientists have long emphasized the importance of color vision in visual perception, and it remains a widely researched area in the field of vision science [89, 90, 96]. Color information helps us recognize objects, perceive depth and surface orientation, and make decisions based on our visual input.

The computer vision community is focused on developing algorithms and models that can analyze and interpret visual information like humans. Nowadays, Convolutional Neural Networks (CNNs) are the most efficient approach to achieving complex computer vision tasks such as object recognition or semantic segmentation. Although CNNs have proven to be very efficient, they handle color information differently from the human visual system. In particular, CNNs deal with all the initial channels simultaneously with a single set of operations (i.e., an architecture). Thus, channel differentiation will only occur when assigning weights to feature maps during the training phase. Nonetheless, the channel processing pipeline remains architecturally identical for each initial channel. On the other hand, in the human brain, biological evidence [180] suggests that channel differentiation occurs as soon as the spectral stimuli are received by the photoreceptor cells and turned into electrical signals. Different biological neural networks process these signals before the visual cortex recomposes the final image. Consequently, we wondered if implementing channel differentiation at the architecture level (i.e., a biomimetic channel processing pipeline) could improve the performance of CNNs. This way, we introduced ColorNAS, a novel way of processing colors by searching for convolutional neural networks (CNNs) specific to each input image channel with differentiable Neural Architecture Search (NAS). The cooperation between the different CNNs is achieved through Ensemble Learning.

We structured our paper as follows: Section A.1.1 features a short survey on related work, Section A.1.2 recalls important concepts about color vision and discusses its biological implementation in human brains, Section A.2 describes the ColorNAS approach, Section A.3 showcases the results of ablation studies and image classification experiments conducted on popular computer vision datasets, and Section A.4 finally brings a conclusion to this paper while pointing out some future work directions.

A.1.1 . Related Work

This paper focuses on the implementation of a color-aware CNN model using NAS. In this section, we review the most significant methods involved.

Neural Architecture Search NAS is a subfield of deep learning that focuses on automating the design of neural network architectures using meta-learning techniques. This has made it possible to design and optimize neural network architectures in a more systematic and efficient manner, moving the field of deep learning architecture design towards formal methods. A large number of NAS methods have been developed. Early approaches used Reinforcement Learning (RL) [231, 232], but their search cost was prohibitively expensive (i.e., hundreds or thousands of GPU days). However, an alternative NAS family is currently trending: differentiable (or gradient-based) methods such as Differentiable ARchiTecture Search (DARTS) [117] in which architectures are represented as sets of parameters. These parameters are optimized during the search phase by a gradient descent algorithm.

Channel-Wise Neural Architecture Search The focus in channel-wise NAS, is on searching for the optimal number of channels and the optimal operations to be applied to each channel, rather than searching for the overall architecture of the network. Only a handful of NAS studies attempted to experiment on channels. FBNetV2 [187] employs a masking mechanism to reuse feature maps (*DMaskingNAS*) to reduce computational and memory costs during the search process. This freed memory allowed the authors to search for the ideal input resolution and number of channels at each architecture layer. PC-DARTS [205] samples a subset of channels at each cell edge and bypass the rest in a shortcut. However, this mechanism causes an inconsistency in the edge selection process that the authors alleviate through edge normalization. NetAdaptV2 [208] monitors the number of filters in each layer. If one is removed, an input channel is bypassed to maintain the same number of output channels. This mechanism was dubbed Channel-level Bypass Connections (CBCs). Therefore, CBCs allowed the authors to merge layer width and depth into a single dimension, thus reducing the computational cost of the search process. [154] divides channels into subgroups at the block level and assigns an architectural parameter to each group. This allows them to improve on existing architecture designs focusing on object detection.

Ensemble Learning Ensemble Learning aims to combine the predictions of multiple individually trained classifiers (e.g., multi-layer perceptrons, convolutional neural networks, or decision trees) in order to improve predictive performance [148]. In the past decades, many different Ensemble methods have been proposed. Bootstrap Aggregating (Bagging) [49] consists in generating n uniform samples (called *bootstrap sets*) from the dataset with replacement. In fact, sampling with replacement makes each bootstrap set independent from the others. Then, n models are fitted each with a specific bootstrap set,

and their predictions are combined in a single final output. Random Forests [13] are a common example of a Bagging model. Boosting [12] is based on sequentially building multiple model instances and training each new instance with feedback on the previous model's mistakes. The most common Boosting algorithm is Adaboost [55]. [196] introduced Stacked Generalization as an advanced Ensemble method. In this case, a learning algorithm (i.e., a meta-learner) is trained to combine the predictions of several other predictive models. Hence, the Ensemble process is not explicitly described by an algorithm but rather learned through interaction with the sub-models. In many cases, the meta-learner is a fully-connected neural network. This is the approach we adopted for ColorNAS, as described in Section 4.1.

A.1.2 . Preliminaries: Color Vision

Humans and convolutional neural networks [104] (CNNs) process images in different ways. CNNs (e.g., ResNets [69]) obtain feature maps from RGB images by applying convolution operations in order to extract patterns. This process is loosely inspired by how visual cortex neurons react to specific stimuli and interact together [84].

On the other hand, humans perceive color using three different types of photoreceptor cells (called "cones") that each reacts to specific wavelengths of light (i.e., red, green, and blue) [48, 173]. These photoreceptor cells transmit the color information to the visual cortex through the optical nerve using a process called visual phototransduction [48] (i.e., the biological conversion of photons into electrical signals). This capacity to perceive color through three different channels is called trichromacy [72]. Besides, recent research showed that some humans possess an additional type of photoreceptor cell, thus leading to tetrachromacy [8].

Thus, both CNNs and humans can be considered trichromats. However, this similarity ends here as humans process each channel through separate neuronal mechanisms and then compare the resulting signals to recompose the final image [180], contrary to CNNs, which process all channels through identical operations. This fact led us to question whether biomimicking brain color vision mechanisms could improve the image-processing capacities of CNNs. Hence, we introduce a novel way to process channels through distributed cooperation between different CNNs and present its implementation in Section A.2.1.

A.2 . Proposed Approach

This section presents our proposed approach for achieving channel-aware NAS with cooperating networks. First, we describe the implementation of this mechanism in a differentiable NAS framework. Then, we detail the different

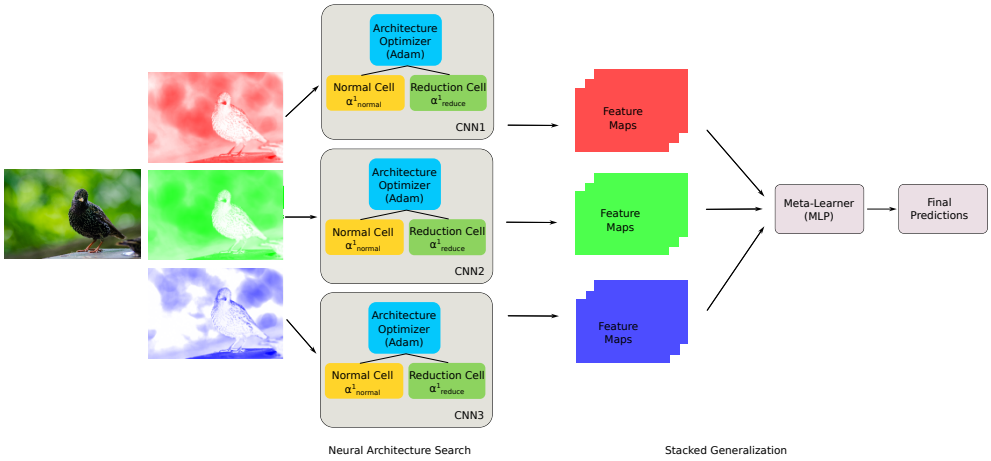


Figure A.1: Layout of the ColorNAS search process with an RGB image as input. Each channel is fed to an independent CNN built using differentiable NAS (DARTS [117]). The output feature maps of all CNNs are then concatenated and fed to a multi-layer perceptron acting as a meta-learner.

color models we implemented.

A.2.1 . Distributed Cooperation between Networks

Inspired by the biological process of color vision exposed in Section A.1.2, we propose a novel mechanism to achieve better channel processing through distributed cooperation between several CNNs whose architectures are obtained using differentiable NAS. More precisely, the input image is split along the channel dimension. Each channel is fed to an independent CNN whose operations are specifically searched for this channel. Then, the outputs (i.e., feature maps) of all CNNs are concatenated to create a single tensor, similarly to the way the different color signals are combined in the visual cortex. Finally, this unified output is passed to a meta-learner (i.e., a multi-layer perceptron) to obtain a final prediction. This combination process is inspired by a widespread Ensemble Learning approach called Stacked Generalization [196]. Hence, instead of searching for a single CNN, we search for a specific, highly specialized architecture to process each channel. The fundamental intuition behind our method is that color channels are of paramount importance for analyzing an image correctly, especially for complex tasks such as object detection or semantic segmentation. In addition, this concept is backed by biological evidence of differentiation in channel processing in the human brain (see Section A.1.2). Fig. A.1 summarizes our approach, dubbed ColorNAS.

We used β-DARTS [210], an evolution of the popular NAS method DARTS [117], as the differentiable NAS backbone of our approach. This choice is motivated by the computational and memory efficiency of DARTS. Furthermore, β-DARTS alleviates some of DARTS' most important limitations with the imple-

mentation of an additional beta-decay loss. This way, each CNN i is fitted with two different sets of architectural weights: α^i_{normal} and $\alpha^i_{\text{reduction}}$. These two sets respectively define *normal* cells (i.e., making up most of the architecture) and *reduction* cells (i.e., performing dimension reduction). Consequently, the search process of ColorNAS involves a total of $2|C|$ sets of architectural parameters, where C represents the channels of the input image.

In addition to standard RGB images, we implemented alternative color models such as RGBY [77] to test whether ColorNAS could leverage these models to boost performance (see Section A.2.2). We also added an RGB to RB conversion algorithm to verify our hypothesis that all color channels are critical for computer vision tasks. Hence, the removal of a color channel should negatively impact performance.

A.2.2 . Colors Models

We used several color models to experiment on the distributed cooperation mechanism presented in Section A.2.1. To this end, we implemented algorithms that convert images from the RGB (Red, Green, and Blue) additive color model, the standard model of digital pictures, to another color model. These are motivated by the fact that the datasets used in the experiments (i.e., CIFAR-10 and CIFAR-100 [100]) only provide RGB images. Besides, it is uncommon to find computer vision datasets where images are encoded in an alternative color model. Thus, the more straightforward approach is to convert existing RGB images before feeding them to our ColorNAS approach.

Firstly, we implemented the RB (Red and Blue) color model by simply deleting the green channel to simulate dichromacy to perform an ablation study on color channels (see Section A.3.2). This conversion mechanism is detailed in Algorithm 6.

Algorithm 6 Algorithm describing the RGB to RB color model conversion.

Require: Matrix: I, the RGB image to convert with shape (3,H,W)
Ensure: Matrix: I_f, the output RB image with shape (2,H,W)

$I_f \leftarrow I$
delete_channel_from_index$(I_f, 1)$

return I_f

Secondly, we implemented the RGBY (Red, Green, Blue, and Yellow) color model in order to assess the performance impact of adding a fourth channel. In particular, several color vision studies [90, 96] showed that tetrachromat humans (i.e., subjects presenting a mutation adding a fourth type of photoreceptor cell to their retina) experience an enhanced perception of the world. Hence, tetrachromats can discriminate spectral stimuli that are oblivious to

trichromats. This biological fact motivated us to feed a fourth channel (i.e., yellow) to our approach to see whether it leads to a performance boost. To this end, we employed the RGB to RGBY algorithm introduced by Hong et al. [77] and featured in Algorithm 7. However, this algorithm computes the yellow channel by interpolation from the red and green channels. Thus, we hypothesize that this artificial channel is not equivalent to one computed from a raw image file or directly obtained from a 4-channel image sensor.

Algorithm 7 Algorithm describing the RGB to RGBY color model conversion described by Hong et al. [77].

Require: Matrix: I, the RGB image to convert with shape (3,H,W)
Ensure: Matrix: I_f, the output RGBY image with shape (4,H,W)

$I_f \leftarrow I$
$R \leftarrow I[0]$
$G \leftarrow I[1]$
$I_f[0] \leftarrow R - \text{minimum}(R, G)$
$I_f[1] \leftarrow G - \text{minimum}(R, G)$
$I_f[3] \leftarrow I_f[0] + I_f[1]$

return I_f

A.3 . Experiments

This section showcases the results of several image classification and object detection experiments conducted on popular computer vision datasets. We also conducted an ablation study on our proposed ColorNAS approach.

A.3.1 . Experimental Settings

All experiments were conducted on CIFAR-10 and CIFAR-100 [100] computer vision datasets using Nvidia RTX 3090 and A6000 GPUs. We mostly used the same hyperparameters as β-DARTS [210] when searching and training. However, we did not use the CIFAR auxiliary tower of DARTS [117]. We also used AutoAugment [36], and Cutout [41] when training. We searched for 8-cell networks for 50 epochs on CIFAR datasets and ImageNet. We trained 8-cell networks for 600 epochs on CIFAR datasets and 300 epochs on ImageNet. Finally, we parsed architectures from architectural weights (α_{normal}, $\alpha_{reduction}$) by selecting the two best incoming operations per node, similar to DARTS. Operations were selected using a threshold of 0.7.

A.3.2 . Ablation Study

We conducted an ablation study on the channel-wise distributed cooperation ColorNAS approach we presented in Section 4.1. Our goal was to as-

sess ColorNAS' performance impact compared to a standard NAS-obtained CNN. To this end, we searched and trained a standard version of FairDARTS [32] on CIFAR-10 and CIFAR-100 [100] using the same hyperparameters as ColorNAS (see Section A.3.1). In Table A.1, we can see that the ColorNAS RGB model obtained 98.1% accuracy, surpassing the standard 8-cell FairDARTS approach by 1.6%. However, when evaluating on CIFAR-100, our RGB approach under-performed the standard FairdDARTS model by 1.3%. One possible explanation for this discrepancy in performance achievements is that the meta-learner in our ColorNAS architecture can correctly classify between the 10 classes of CIFAR-10 but fails to find a good projection onto the larger class distribution of CIFAR-100 (i.e., 100 classes thus a 10x increase factor compared to CIFAR-10). Our suspicion is motivated by the fact that the meta-learner is an essential part of the ensembling process and is responsible for combining the predictions of the different sub-models.

Furthermore, we wanted to highlight the critical role of colors in Deep Learning Computer Vision, as several studies have already pointed out in biological vision [90, 89]. To this end, we implemented the RB color model presented in Section 4.1 to assess the impact on the performance of removing a color channel. In Table A.1, we can notice that the RG color is severely underperforming compared to the RGB color model (92.7% vs. 98.1% top-1 accuracy on CIFAR-10, and 76.2 vs. 77.6 top-1 accuracy on CIFAR-100). Thus, this validates our hypothesis that colors are critical to achieving computer vision tasks.

A.3.3 . Searching and Evaluating on CIFAR

We searched and evaluated on CIFAR-10 [100] our ColorNAS approach (see Section A.2.1) using all the different color models presented in Section A.2.2 (i.e., RB, RGB, and RGBY). All architectures were searched using 8-cell proxy networks. We also searched and evaluated a standard 8-cell FairDARTS model to conduct an ablation study on ColorNAS (see Section A.3.2).

Notably, the RGB model (3 8-layer networks) outperformed the standard FairDARTS approach (a single 8-layer network) by 1.6% while the number of parameters more than doubles but remains modest at 2.9 M. The search cost also modestly increases from 0.25 GPU-day to 0.5 GPU-day. In addition, the RB model matches our expectations as removing the green channel degrades performance to 92.7%, underperforming the RGB model by 5.4%. However, the RGBY model did not achieve to outperform the standard FairDARTS approach and only reached a similar level of performance at 96.4%.

In addition to CIFAR-10, we also evaluated ColorNAS on CIFAR-100 [100] our ColorNAS approach. Unfortunately, all models failed to reach the same level of performance as the standard approaches. In particular, our RGB model only reached 81.0% top-1 accuracy, thus underperforming DARTS by 1.3%.

All experimental results are presented in Table A.1.

Table A.1: **Comparison of models trained on CIFAR-10 and CIFAR-100 [100].** Each reported Top-1 accuracy is the best of 4 independent runs. For previous baselines, results are the official numbers from their respective articles with the search cost expressed with the GPU used by the authors.

Models		Params (M)	C10 Top-1 (%)	C100 Top-1 (%)	Layers	Search Cost (GPU-days)
DARTS[117]		3.3	97.0	82.3	20	0.4
FairDARTS[32]		1.3	96.5	83.8	8	0.25
FairDARTS-b[32]		3.9	97.5	78.9	20	0.25
PC-DARTS[205]		3.6	97.4	83.1	20	0.1
P-DARTS[23]		3.4	97.5	84.1	20	0.3
DARTS-[31]		3.5	97.4	82.5	20	0.4
β-DARTS[210]		3.8	97.5	83.5	20	0.4
Ours	RGB	2.9	**98.1**	81.0	8	0.5
	RGBY	3.2	96.4	76.9	8	1
	RB	1.7	92.7	76.2	8	0.25

A.4 . Conclusion

This paper explored a novel way to better consider the specificity of color channels when searching for convolutional neural network architectures. In Section A.3, we showed that despite positive initial results on CIFAR-10, our ColorNAS approach failed to scale appropriately to the larger CIFAR-100 dataset. Furthermore, we highlighted the paramount importance of color channels through the ablation study presented in Section A.3.2. Additionally, we demonstrated that RGB is still the best color space for image classification compared to other alternatives (namely RB and RGBY). Our goal was not to reach the highest competitive scores in image classification but to point out that a viable alternative to classical CNN channel processing exists. Future work should then focus on improving ColorNAS scaling abilities, especially its meta-learner.