One-Shot Neural Architecture Search for Computer Vision

by

Mehraveh JAVAN ROSHTKHARI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, MAY 26, 2026

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Marco Pedersoli, Thesis supervisor
Department of Systems Engineering, École de technologie supérieure

Mr. Matthew Toews, Thesis Co-Supervisor
Department of Systems Engineering, École de technologie supérieure

Mr. Ismail Ben Ayed, Chair, Board of Examiners
Department of Systems Engineering, École de technologie supérieure

Mr. Jose Dolz, Member of the Jury
Department of Software Engineering and IT, École de technologie supérieure

Mr. Mark Coates, External Independent Examiner
Department of Electrical and Computer Engineering, McGill University

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON APRIL 7, 2026

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my supervisors, Prof. Marco Pedersoli and Prof. Matthew Toews. I thank them for their tremendous support and valuable guidance throughout my PhD journey. I am also very grateful that they provided me with the chance to change my field of research from physics to deep learning and computer vision. I particularly enjoyed the scientific discussions I had with both of them and the valuable insights they provided that shaped the trajectory of my research. I would also like to thank my colleagues at LIVIA lab who have helped me in various ways.

I would like to thank my family for their support, encouragement, and investment in achieving my goals. Your unconditional love and support have been fundamental in my personal and academic achievements. Finally, I thank all my friends who indirectly contributed to this work and have been by my side throughout this journey. Your support and our insightful discussions were essential to my academic and personal growth.

Recherche d'Architecture Neuronale One-Shot pour la Vision par Ordinateur

Mehraveh JAVAN ROSHTKHARI

RÉSUMÉ

L'apprentissage profond a connu un succès remarquable dans le domaine de la vision par ordinateur grâce aux avancées des réseaux de neurones convolutifs (CNN). Cependant, la conception de modèles CNN optimaux (architectures) pour une nouvelle application est une tâche fastidieuse qui nécessite une expertise pointue ainsi qu'un long processus d'essais et d'erreurs. La recherche d'architecture neuronale (NAS) est apparue au cours de la dernière décennie comme une solution pour automatiser ce processus de conception. Les méthodes « one-shot » constituent l'une des principales approches de la NAS. Elles permettent de réduire le coût de calcul en entraînant un unique super-réseau contenant toutes les architectures possibles de l'espace de recherche, et en héritant directement de ces poids pour évaluer les performances. Cependant, un problème fondamental de la NAS one-shot est la dégradation de la qualité de l'estimation des performances du super-réseau, due aux conflits et à la co-adaptation des poids lors de l'entraînement. Ce problème peut être résolu en réduisant le partage de poids, soit en utilisant plusieurs super-réseaux pour différentes parties de l'espace de recherche, soit en se concentrant sur les zones prometteuses via une méthode d'échantillonnage.

Cette thèse se concentre sur l'amélioration des méthodes de NAS one-shot sous divers aspects. Nous présentons d'abord une introduction à nos recherches et à nos contributions. Nous fournissons ensuite un contexte général sur les CNN, les différents types d'architectures, la recherche arborescente de Monte-Carlo (MCTS) et la NAS. Pour notre première contribution, nous abordons l'optimisation de la configuration du sous-échantillonnage des CNN comme un problème de NAS. Nous proposons un mélange équilibré de super-réseaux pour partitionner l'espace de recherche et réduire le partage de poids en attribuant des super-réseaux distincts à chaque partition. Nous proposons d'apprendre le partitionnement et l'association des architectures de manière équilibrée afin de garantir l'équité lors de l'entraînement de plusieurs super-réseaux. Ensuite, nous proposons deux approches pour apprendre la structure hiérarchique (arbre de recherche) de l'espace de recherche NAS pour la MCTS, simultanément à l'entraînement du super-réseau. Notre première approche consiste à apprendre la hiérarchie de l'espace de recherche de manière non supervisée. Nous proposons d'utiliser la similitude fonctionnelle des architectures, basée sur leur vecteur de sortie, pour construire cette hiérarchie. Contrairement aux travaux précédents, notre méthode n'utilise pas de conception hiérarchique par défaut ni de prédictions de performance imprécises issues du super-réseau. La seconde approche est une méthode itérative permettant d'affiner la hiérarchie en s'appuyant sur des estimations de performance de plus en plus précises. Ces deux approches facilitent la NAS one-shot en offrant un meilleur compromis entre exploration et exploitation, améliorant ainsi les performances finales tout en réduisant les coûts de recherche. Enfin, pour notre dernière contribution, nous étudions les conflits et la coopération des gradients des architectures échantillonnées pendant l'entraînement du super-réseau. Étant donné que les conflits de gradients ne sont pas uniformes dans l'espace de recherche, nous nous concentrons sur l'entraînement effectif que chaque architecture reçoit

lors d'une étape d'optimisation. Nous proposons une métrique de densité de gradient qui estime l'entraînement effectif reçu par une architecture en mesurant l'alignement de son gradient avec le reste de l'espace de recherche. Nous proposons une méthode d'échantillonnage sensible à la densité afin de réduire le biais dans l'entraînement effectif reçu lors de l'optimisation du super-réseau. Pour conclure, nous présentons une synthèse de nos travaux ainsi que des perspectives futures pour l'amélioration de la NAS one-shot.

Mots-clés: Vision par ordinateur, Apprentissage automatique automatisé (AutoML), Recherche d'architecture neuronale (NAS), NAS one-shot, partage de poids

One-Shot Neural Architecture Search for Computer Vision

Mehraveh JAVAN ROSHTKHARI

ABSTRACT

Deep learning has achieved remarkable success in the field of computer vision with the advancement of Convolutional Neural Networks (CNN). However, the design of the optimal CNN models (architectures) for a new application is a time-consuming task, requiring expert knowledge and trial and error. Neural Architecture Search (NAS) has emerged in the past decade as a solution to automate the process of finding optimal architecture design. One-shot methods are one of the main NAS approaches that reduce the computational cost of NAS by training a single supernet that contains all possible architectures in the search space and directly inheriting these weights for architecture performance evaluation. However, a fundamental issue in one-shot NAS is the degradation of the quality of architecture performance estimation based on the supernet due to conflict and co-adaptation of weights during supernet training. This issue can be addressed by reducing the weight sharing by using multiple supernets for various parts of the search space or by focusing only on promising parts of the search space during training by using a sampling method.

This thesis focuses on improving one-shot NAS methods from various aspects. We first provide an introduction to our research and contributions. We then provide general background about CNNs, various architecture designs, Monte-Carlo Tree Search (MCTS), and NAS. For our first contribution, we focus on optimizing the downsampling configuration of CNN as a NAS problem. We propose a balanced mixture of supernets to partition the search space and reduce weight sharing by utilizing distinct supernets for each partition. We propose to learn the partitioning and association of architectures to each partition in a balanced manner to ensure fairness in training multiple supernets. Next, we propose two approaches for learning the hierarchical structure (search tree) of the NAS search space for MCTS simultaneously with supernet training. Our first approach is to learn the hierarchy of the search space in an unsupervised manner. We propose to use the functional similarity of architectures based on their output vector to construct the hierarchy. Unlike previous works, our method does not use a default hierarchical design or inaccurate performance predictions from the supernet. The second approach is an iterative method to refine the hierarchy based on increasingly better performance estimations from the supernet. Both approaches facilitate one-shot NAS by providing a better exploration-exploitation trade-off, improving the final performance with reduced NAS cost. Finally, as our last contribution, we investigate the gradient conflict and cooperation of sampled architectures during supernet training. Since gradient conflict is not uniform in the search space, we focus on the effective training each architecture receives in an optimization step. We propose a gradient density metric that estimates the effective training received by an architecture by measuring how aligned its gradient is with the rest of the search space. We propose a density-aware sampling method to reduce the bias in effective training received during supernet optimization. Finally, we provide a conclusion and future directions for improving one-shot NAS.

Keywords: Computer Vision, Automated Machine Learning (AutoML), Neural Architecture Search (NAS), One-Shot NAS, Weight-Sharing

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Page

LIST OF ALGORITHMS

Page

LIST OF ABBREVIATIONS

ASPP	Atrous Spatial Pyramid Pooling
AutoML	Automated Machine Learning
BSE	Boltzmann Softmax Exploration
CNN	Convolutional Neural Network
DAG	Directed Acyclic Graph
DARTS	Differentiable Architecture Search
DW	DepthWise
EA	Evolutionary Algorithm
EMA	Exponential Moving Average
FC	Fully-Connected
FLOPs	Floating Point Operations
GAEA	Geometry-Aware Exponentiated Algorithm
GPU	Graphical Processing Unit
ID	Identity
KT	Kendall's Tau
MBConv	MobileNet Convolutional blocks
MCTS	Monte-Carlo Tree Search
mIOU	mean Intersection Over Union
NAS	Neural Architecture Search

NLP	Natural Language Processing
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SE	Squeeze-and-Excitation
SPOS	Single-Path One-Shot
UCT	Upper Confidence Bound for Trees

INTRODUCTION

0.1 Context

Deep neural networks have shown great success in a wide range of applications in various domains, such as computer vision (Krizhevsky, Hinton *et al.*, 2009), speech recognition (Vaswani *et al.*, 2017), and many others. Increased hardware computational capabilities have significantly increased the use of neural networks in the past decade, enabling deep architectures that approximate the relationship between the input and output, eliminating the need for feature engineering. One of the most important factors in the performance of the neural network is the suitability of the model design or architecture for the given task and dataset. In fact, the introduction of new Convolutional Neural Network (CNN) architecture designs throughout the years has been a key factor in improving the performance of deep learning models.

Traditionally, neural architectures were designed by experts using theoretical understandings and trial and error. Many mainstream architectures, such as VGG (Simonyan & Zisserman, 2014), ResNet (He, Zhang, Ren & Sun, 2016a), and U-Net (Ronneberger *et al.*, 2015) were designed manually. This inherently made the architecture design bound by human intuition and conceptualization. Furthermore, other objectives, such as resource constraints (memory, latency, etc.), which are crucial for edge devices, were difficult to manually incorporate into architecture design. Different datasets and domains (such as medical or satellite images) may also require different model designs. Finally, the expertise required to design an architecture for a specific application is not accessible to many users in various fields.

In the past decade, Neural Architecture Search (NAS) (Zoph & Le, 2016) has emerged as a subfield of Automated Machine Learning (AutoML) (Hutter, Kotthoff & Vanschoren, 2019) to provide automation in model and architecture design. Traditional NAS methods were hugely impractical due to their enormous resource requirements ($\approx 10^3$ GPU hours) to train and evaluate

each candidate architecture in a vast search space from scratch. Since those early days of NAS, more efficient evaluation methods, such as one-shot NAS methods, have made NAS more practical.

In fact, the introduction of one-shot methods based on weight-sharing (Pham, Guan, Zoph, Le & Dean, 2018) has been one of the most important advances in NAS. In this approach, a super-network ("supernet") encompassing the entire search space is trained, and each individual architecture is evaluated by inheriting the weights from this supernet. While this increases the efficiency of evaluation, the quality of the evaluation is compromised due to the entanglement of different paths using the same shared weights (Bender, Kindermans, Zoph, Vasudevan & Le, 2018). Therefore, many subsequent works have focused on improving the supernet training to increase the accuracy of its performance estimations. In this thesis, we focus on one-shot NAS for computer vision tasks and CNN models.

0.2 Challenges and Problem Statement

The core assumption in one-shot NAS is that the evaluations of architectures provided by inheriting the weights from the supernet can accurately enough reflect their true performance. However, the weight sharing mechanism inherently introduces an optimization conflict where shared weights of vastly different architectures are jointly optimized. This co-adaptation of weights leads the supernet evaluations to estimate a poor ranking of architectures relative to the ground-truth performance, hindering the ability to find optimal architectures.

Furthermore, weight-sharing NAS methods have been dominated by gradient-based (differentiable) methods (Liu *et al.*, 2018b). However, while efficient, these methods face several challenges in high memory requirement and in discretizations, often failing to account for joined influence of different operations on the architecture performance.

Sample-based one-shot NAS resolves some of the challenges of differentiable NAS by sampling only one (or a few paths) during training. These methods are sometimes referred to as Single-path one-shot methods and have been widely used in recent years. However, the weight co-adaptation among architectures remains an issue.

Much research in recent years has focused on improving one-shot NAS by directly reducing weight sharing in the search space, reducing gradient conflict, or applying various architecture sampling methods to guide the supernet training towards higher-performing architectures.

In summary, while one-shot NAS significantly reduces the computational cost compared to brute-force approaches, it introduces a critical ranking inconsistency problem. The joint optimization of shared weights across diverse architectures leads to co-adaptation and gradient interference, which is the cause of ranking inconsistency. The search process then often converges to suboptimal architectures. Therefore, there is a need for NAS frameworks that can mitigate these weight-sharing artifacts to provide reliable architecture rankings without the prohibitive costs of brute-force evaluation.

Finally, while weight sharing reduces the computational cost compared to early brute-force approaches, the overall computational cost of one-shot NAS pipeline remains relatively high.

0.3 Objectives and Contributions

The objective of this thesis is to improve the Single-Path One-Shot (SPOS) (Guo *et al.*, 2020b) NAS method from several aspects. One approach is to directly reduce weight sharing in the search space by using multiple supernets (Zhao, Wang, Tian, Fonseca & Guo, 2021a; Hu *et al.*, 2022). The search space is partitioned into a disjoint set, with each partition corresponding to a distinct supernet. This method often incurs additional cost as they need training for more parameters. While random partitioning provides some benefits (Zhao *et al.*, 2021a), to provide the maximum benefit, the partitioning of the search space could be performed in an intelligent

way. A partitioning mechanism is needed that groups architectures with similar optimization dynamics together, as well as ensuring fair and adequate training for each supernet.

Another approach is to promote efficient navigation of the complex and vast discrete NAS search space. To make sampling of new candidate architectures more efficient, it is crucial to identify and prioritize sampling from promising regions of the search space as soon as possible. A hierarchical search space design (tree structure) combined with MCTS (Świechowski, Godlewski, Sawicki & Mańdziuk, 2023) is a classical choice to balance exploration-exploitation (Wang, Zhao, Jinnai, Tian & Fonseca, 2019b). However, the hierarchical structure determines the efficiency of MCTS. Ideally, the hierarchy places a large portion of unpromising architectures in the same early branch, facilitating the sampling method to undersample these regions and focus on promising regions. Manually designed hierarchies based on structural properties (Su *et al.*, 2021a) do not guarantee that property.

Applying prioritized sampling during the supernet training has several benefits. In general, as prioritized sampling increasingly focuses on sampling from certain promising regions, it effectively reduces the search space and the weight sharing. It allows the weights to be optimized and adapt primarily to high-performing architectures. However, as supernet evaluations have low accuracy during training (Bender *et al.*, 2018; Yu, Sciuto, Jaggi, Musat & Salzmann, 2019; Yu, Ranftl & Salzmann, 2020b), they provide inadequate and unreliable estimation to identify promising regions of the search space. Applying hierarchical factorization of the search space and MCTS after supernet training (Wang, Xie, Li, Fonseca & Tian, 2019a; Wang *et al.*, 2019b), while more straightforward, does not provide the benefits of reduced weight sharing. Therefore, designing a hierarchical search space during supernet training is a challenging task that can potentially increase the efficiency and performance of NAS.

Finally, uniform sampling has been extensively used as the default, standard, and bias-free architecture sampling method during supernet training (Guo *et al.*, 2020b). However, its

fairness in training different architectures has been scrutinized. Uniform sampling provides the expectation fairness, ensuring that every operation/architecture has an equal probability of being sampled and updated. FairNAS (Chu, Zhang & Xu, 2021c) enforces strict fairness to ensure that at each window of training (e.g., one epoch), different weights are updated the same number of times. Another aspect is the budget bias in uniform sampling, where models with different degrees of complexity and convergence rate receive the same training opportunities (Zhang, Yu, Zhao & Ou, 2023; Jeon *et al.*, 2025). Therefore, uniform sampling is not truly bias-free and fair, and alternative sampling methods are required to debias sampling from different aspects.

0.3.1 Contributions

As discussed above, this thesis focuses on improving one-shot NAS method. The main contributions of the thesis are summarized as follows:

- **Searching for optimal pooling structure of CNN (Chapter 2)** We analyze and show that the position of downsampling layers in CNN plays a crucial role in performance. We propose a NAS benchmark with a sole focus on downsampling layers. We present the task of finding the optimal CNN downsampling or pooling layers as a NAS problem, and perform extensive experiments to evaluate search space design and various NAS methods. We show that mainstream NAS methods struggle with this task due to the full weight sharing of the architectures. We propose a novel NAS method to reduce weight sharing by using a balanced mixture of specialized supernets. The partitioning of the search space for different supernets is learned by the model. Our method shows improved results compared to several NAS methods, such as SPOS and DARTS.
- **Improving NAS with Monte Carlo Tree Search (MCTS) (Chapters 3 and 4):** We revisit the use of MCTS for NAS and the crucial advantages and challenges of this approach. We highlight the sampling probability landscape of various NAS methods and the advantage of using a hierarchical search space. We propose two methods to tackle the challenging

task of training a supernet and sampling with unreliable estimations simultaneously. First, we propose to learn the tree structure in an unsupervised manner by hierarchical clustering of architecture outputs. Our method shows improved performance and search efficiency compared to the manual (default) hierarchical design. Second, we propose an iterative algorithm to refine and redesign the tree structure during supernet training. We show that with a small number of iterations and negligible overhead cost, our method achieves competitive results for classification and segmentation tasks.

- **Debiasing supernet training with density-aware sampling (Chapter 5) :** We investigate the gradient conflict and co-adaptation among architectures in the search space and re-frame it in terms of gradient density. We propose a novel density-aware sampling approach that balances the effective gradient updates in regions with different gradient densities. We propose a cheaper proxy to estimate gradient density, and provide two sampling algorithms for small and larger search spaces.

0.3.2 Outline

This thesis is organized as follows:

- **Chapter 1:** We provide background and related work. We review the basics of CNN (section 1.1) and various architecture designs for classification and segmentation tasks (section 1.1.2). We then briefly provide background for MCTS in section 1.2. NAS is reviewed in section 1.3. We focus on three main components of NAS with section 1.3.1 focusing on search space design, section 1.3.2 on search algorithms, and section 1.3.3 on evaluation methods. We discuss one-shot NAS separately in section 1.3.4, and more advanced NAS methods in section 1.3.5.
- **Chapter 2:** This chapter investigates the impact of downsampling configurations on CNNs. We frame optimization of downsampling as a NAS problem and establish a Pooling benchmark

(section 2.3.1) that focuses solely on downsampling configurations. We propose a balanced mixture of supernets (section 2.2.2) to reduce weight sharing and show the effectiveness of our proposed method compared of both mainstream NAS and non-NAS methods.

- Related Publication: Balanced Mixture of supernets for Learning the CNN Pooling Architecture, Roshtkhari, M. J., Toews, M., & Pedersoli, M. (2023, December). In International Conference on Automated Machine Learning (pp. 8-1). PMLR.

- **Chapter 3:** We discuss the sampling probability distribution for NAS and highlight the effectiveness of a hierarchical search space design, as well as the node independence assumption present in many NAS sampling methods. We address the limitations of manually designed hierarchies in MCTS for NAS. We propose an unsupervised approach that clusters architectures based on their functional similarity (output correlations). By building a semantically meaningful search tree, we improve the efficiency of MCTS in navigating large search spaces.

  - Related Publication: (Under Review) Neural Architecture Search by Learning a Hierarchical Search Space, Roshtkhari, M. J., Toews, M., & Pedersoli, M. (2025). arXiv preprint arXiv:2503.21061.

- **Chapter 4**: Building upon the concepts of chapter 3, we propose an iterative framework to refine the tree structure used by MCTS for NAS. We demonstrate that allowing the tree structure to evolve alongside the Supernet weights leads to more reliable performance estimation and superior final architectures for both classification and semantic segmentation.

  - Related Publication: Iterative Monte Carlo Tree Search for Neural Architecture Search, Roshtkhari, M. J., Toews, M., & Pedersoli, M. (2025, September). In International Conference on Automated Machine Learning (pp. 3-1). PMLR.

- **Chapter 5:** We propose a novel sampling method to address the fairness of training architectures during training. We provide an analysis of how uniform sampling biases the Supernet toward high-density architectural regions. We then propose Density-Aware

Sampling, a method that uses gradient density metrics to rebalance training updates, resulting in improved ranking correlations and model performance.

- Related Publication: (Accepted) Debiased One-shot NAS via Density-aware Sampling, Roshtkhari, M. J., Toews, M., & Pedersoli, M. (2026), Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Findings.

- **Conclusion and Recommendations:** We provide the conclusion and recommendations for future work.

CHAPTER 1

BACKGROUND

In this chapter, we cover basic concepts related to CNNs and NAS. First, we cover the fundamentals of CNN for image classification and semantic segmentation and revisit some of the most widely used CNN architectures and design concepts. We then provide a brief overview of MCTS (section 1.2) as one of the fundamental methods used in this thesis. We then cover core aspects of the NAS: the search space, search algorithm, and evaluation method. Finally, we focus on one-shot NAS methods.

1.1 Convolutional Neural Networks

Neural networks are a powerful tool to learn the complex function representing the relationship between the inputs and outputs for various tasks (LeCun, Bengio & Hinton, 2015). Deep neural networks with many layers can capture even more complex relationships. The utilization of Graphical Processing Units (GPUs), the availability of large datasets, and the development of better optimization techniques have made deep neural networks more applicable to a wide range of problems. Convolutional Neural Networks (CNNs) are extensively used for computer vision (CV) tasks such as image classification, detection, localization, and semantic and instance segmentation.

Similar to the human visual system, a CNN structures the filtered output in a hierarchical manner. Unlike dense connections, where any neuron is connected to all neurons from the previous layer, CNN uses the assumption that close regions in images are correlated. These shared weights among local regions provide more stable features and reduce the number of parameters. Typical CNNs are usually composed of three types of layers: convolution layers, pooling layers, and fully-connected layers.

Considered a breakthrough, in 2012 AlexNet demonstrated the capabilities of deep CNN in classification on ImageNet (Deng *et al.*, 2009) dataset, winning ImageNet Large Scale Visual Recognition (ILSVRC) Challenge by a large margin. Several factors have contributed to the

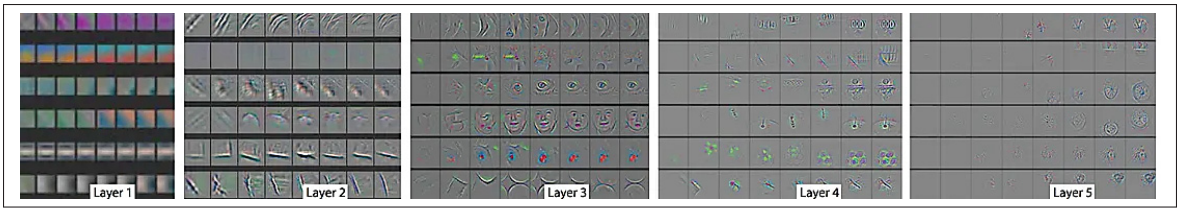


Figure 1.1 Features at different layers of trained CNN
Taken from Zeiler & Fergus (2014)

improvement of CNN in the last decade, such as better optimization algorithms, regularization mechanisms (such as dropout (Srivastava, Hinton, Krizhevsky, Sutskever & Salakhutdinov, 2014), and weight decay (Loshchilov & Hutter, 2017)), and novel network architectures. In this section, we first briefly introduce various components of a CNN, then we examine popular design choices and architectures of CNN for image classification and semantic segmentation tasks.

1.1.1 CNN Components

Convolution Layer: The convolutional layer is the foundational component for the CV task. The convolution operation is a linear operation defined by a kernel (or filter). It is used to learn the correlation of local pixels of the input by learning the kernel weights. Convolution layer is equipped with several channels in which each channel has its own filters that retrieve various patterns from the input or feature maps (Goodfellow, 2016). Generally, the greater the number of channels, the more powerful the representation a network can achieve. There are several types of convolutional operations, with the most commonly used being the standard convolution, depthwise separable convolution (Kaiser, Gomez & Chollet, 2017), and atrous (dilated) convolutions (Yu & Koltun, 2015). Figure 1.1 shows an example of features trained on the ImageNet dataset. Early layers of CNN capture low-level features such as edges and textures. As layers go deeper, more complex patterns such as corners and contours are detected.

Nonlinear Activation: The convolution layer also contains a normalization and a nonlinear activation function. The nonlinearity is crucial in learning the complex patterns in data. Without

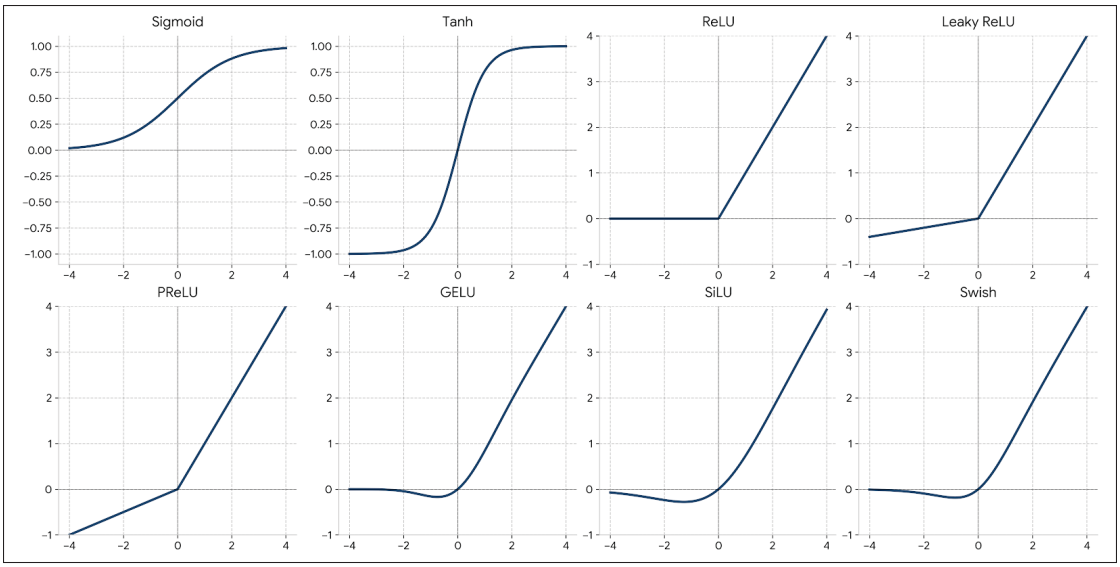


Figure 1.2 Various nonlinear activation functions

an activation function, multiple layers of a neural network would be equivalent to a single-layer linear model. Figure 1.2 shows common activation functions. Inspired by the activation of biological neurons, traditionally, tanh and sigmoid (logistic regression) were used as activation functions. The sigmoid function is defined by the following formula:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1.1}$$

has the significant drawback of vanishing gradients; for extreme input values, the function's derivative approaches zero. This small gradient hinders the learning ability of earlier layers of the network as the gradients are very small during backpropagation. The Tanh activation function was an alternative that provided stronger gradients. Rectified Linear Unit (ReLU) (Nair & Hinton, 2010) became the standard activation function that solved the vanishing gradient problem. ReLU is defined as:

$$ReLU(x) = max(0, x) \tag{1.2}$$

The operation for ReLU is computationally more efficient and promotes sparsity in the network. However, ReLU outputs zero for any negative input, resulting in dead neurons with zero derivative that are unable to update the parameters. Several alternative activation functions have been proposed, such as Leaky ReLU (Maas, Hannun, Ng *et al.*, 2013), Parametric ReLU (Zhang, Pan, Sun & Tang, 2018), Swish (Ramachandran, Zoph & Le, 2017), GeLU (Hendrycks, 2016), and SiLU (Elfwing, Uchibe & Doya, 2018).

Batch Normalization: The distribution of the layer's inputs changes during the training because of continuous updates of the weights of previous layers. To address the internal covariance shift, Batch Normalization (BN) (Ioffe, 2015) is used to normalize the activations within each minibatch of data during training. The normalization facilitates more stable and faster training of the network.

Fully-connected layer: A fully connected layer or dense layer has been used as the main building block of multilayer perceptrons (MLP). In CNN for classification, it is used at the end of the network, after the feature extraction is completed. It applies a linear transformation on the input vector x as:

$$y = Wx + b \tag{1.3}$$

Where W is a matrix of weights and b is the bias vector. W and b are learnable parameters.

Pooling layer: Pooling is a form of downsampling that is used to progressively reduce the spatial size of feature maps and increase the size of the receptive field. The most widely used pooling method is maxpooling, shown in figure 1.3. It outputs the maximum value in every filtered region. It is usually applied with the 2×2 window and a stride of 2. It provides more robustness to small translational variances and denoising by keeping only the strongest signal. Another commonly used pooling is average pooling, which, instead of taking the max value, returns the average value of the local region. Global average pooling, usually used before or in place of the fully connected layer for classification take the average of all values in the feature

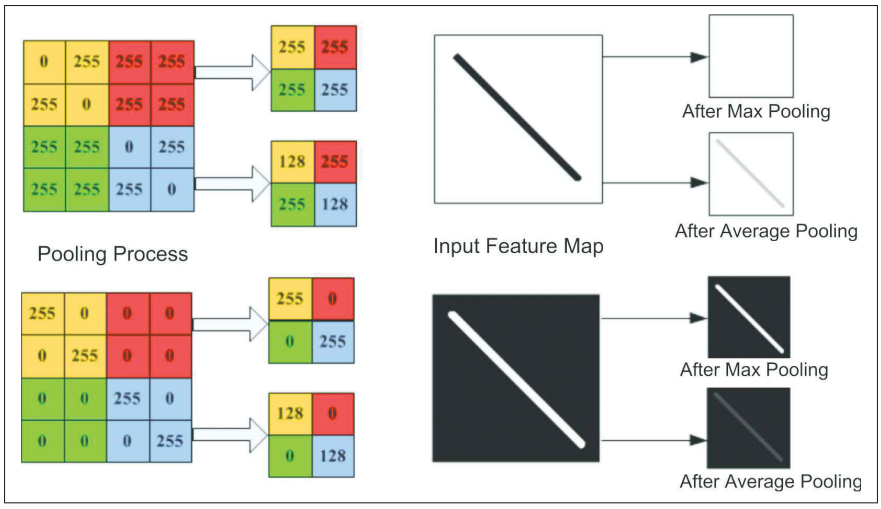


Figure 1.3 Max and average pooling
Taken from Jie & Wanda (2020)

map. Alternative pooling regimes such as mixed and gated max-average pooling have also been proposed (Lee, Gallagher & Tu, 2016). Equivalently, the max pooling layer can be replaced by using stride 2 convolutions, and several CNN architectures use this method (Springenberg, Dosovitskiy, Brox & Riedmiller, 2014).

1.1.2 Popular CNN Architectures

In this section, we provide an overview of popular architectures for image classification and semantic segmentation tasks and discuss design principles and components of these architectures.

1.1.2.1 Image Classification

In image classification, a specific label is assigned to an entire image from a predefined set of classes. Generally, for classification task, the receptive field CNN is continuously increased to cover the entire input image. The receptive field (or field of view) is one of the basic concepts of CNNs and is the specific region of the input image that a single neuron is viewing.

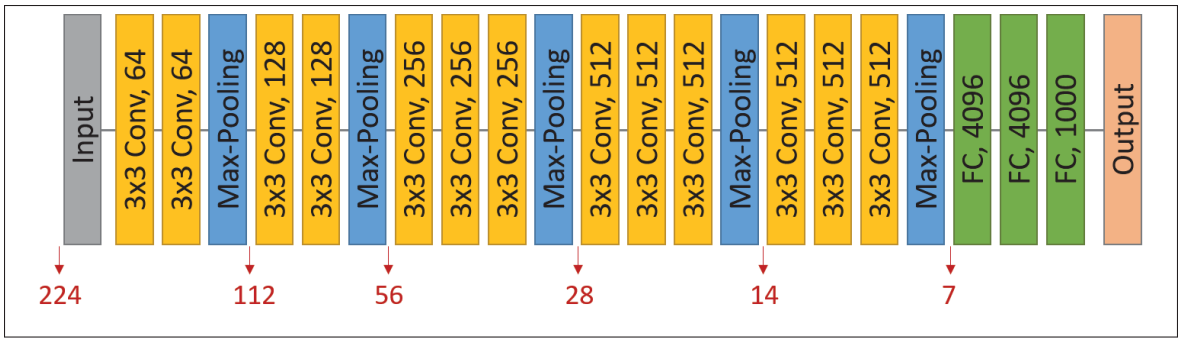


Figure 1.4 Architecture of VGG16. Multiple layers of convolutions with 3×3 kernels are followed by maxpooling layers. The numbers at the bottom show feature map sizes (height or width), which are halved by maxpooling layers

As mentioned earlier, one of the most important CNN architectures is AlexNet (Krizhevsky, Sutskever & Hinton, 2012), which consists of 5 convolutional and 3 fully connected layers (figure 5). AlexNet uses ReLU as the activation function and dropout to reduce overfitting. Inspired by AlexNet, VGG16 and VGG19 (Simonyan & Zisserman, 2014) utilizes very small spatial filters throughout the network, further reducing the impact of parameter explosion while making the network deeper. It replaces large kernel-sized filters (11 and 5 in the first and second convolutional layer of AlexNet) with multiple 3×3 kernel-sized filters, one after another. These convolutional layers are arranged in several blocks, each followed by a max-pooling layer. These pooling layers heavily contribute to the increase of the receptive field in CNN. The output channels are doubled after each block. Figure 1.4 shows the architecture of VGG16.

Inception-v1 (Szegedy *et al.*, 2015) increases the width of the architecture by introducing an inception module, which consists of several parallel branches (figure 1.5). To reduce computational cost, 1×1 convolutions before 3×3 convolutions reduce the number of channels. GoogleLeNet model consists of nine inception modules.

Skip connections have been a groundbreaking innovation in architectures and played a crucial role in helping the training of very deep networks. Previously, extremely deep CNNs suffered from vanishing gradients and struggled to outperform shallower models. Residual Network (ResNet (He *et al.*, 2016a)) introduced using skip connections, providing an alternative path for

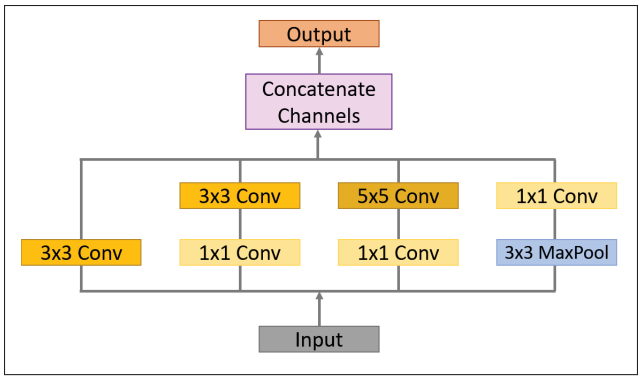


Figure 1.5 Inception module used in Inception models

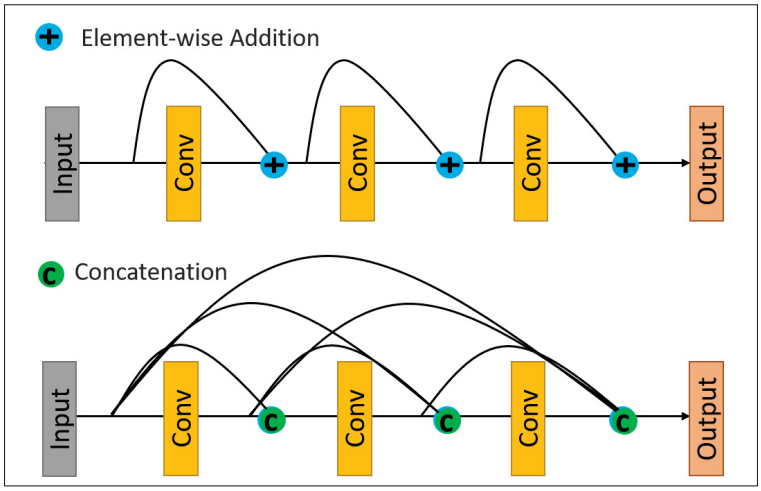


Figure 1.6 Resnet and DenseNet architectures

data and gradients flow by skipping over one or several layers. It allows the use of a combination of lower-level features (from earlier layers) with higher-level, more abstract features of later layers.

In figure 1.6, we show the overall architecture of ResNet and DenseNet (Densely Connected Networks (Huang, Liu, Van Der Maaten & Weinberger, 2017)) that utilize skip connections. In ResNet connectivity, each layer receives input from element-wise summation of the two previous layers. In DenseNet, each layer receives the concatenation of all previous layers as input. ResNet has become widely used not only for classification, but also as a default backbone architecture

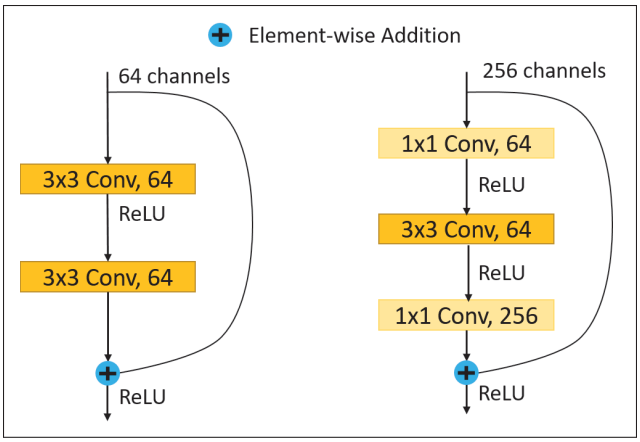


Figure 1.7 Basic (left) and bottleneck (right) blocks for ResNet-v1

for segmentation. ResNet-v1 utilizes two types of residual blocks: a basic block and a bottleneck block, shown in figure 1.7. Basic Blocks are used in previous-18 and 34. For deeper networks (50,101, 152 layers), 1×1 convolutions are used before and after 3×3 convolutions to reduce the number of parameters without degrading performance.

ResNet-v1 uses post-activation, i.e., the nonlinearity (ReLU) is applied after the first convolution and after the final addition. In ResNet-v2 (He, Zhang, Ren & Sun, 2016b), BN and ReLU are applied before each convolution. The pre-activation moves the activation to the main path, the final operation of the residual block, a simple addition, and makes the second nonlinearity and identity mapping, which helps the flow of the gradients. Other variations of ResNet include WideResNet (Zagoruyko & Komodakis, 2016) and ResNeXt (Xie, Girshick, Dollár, Tu & He, 2017).

MobileNet-v1 (Howard *et al.*, 2017) was designed for mobile vision applications where computational resources are limited. It replaced standard convolution with a more efficient operation: a depthwise separable convolution. In figure 1.8, we show depthwise and pointwise convolutions. A depthwise separable convolution consists of a depthwise convolution followed by a pointwise convolution. A standard convolution applies filters across all channels of the input, performing both spatial filtering and channel mixing at the same time. Depthwise separable

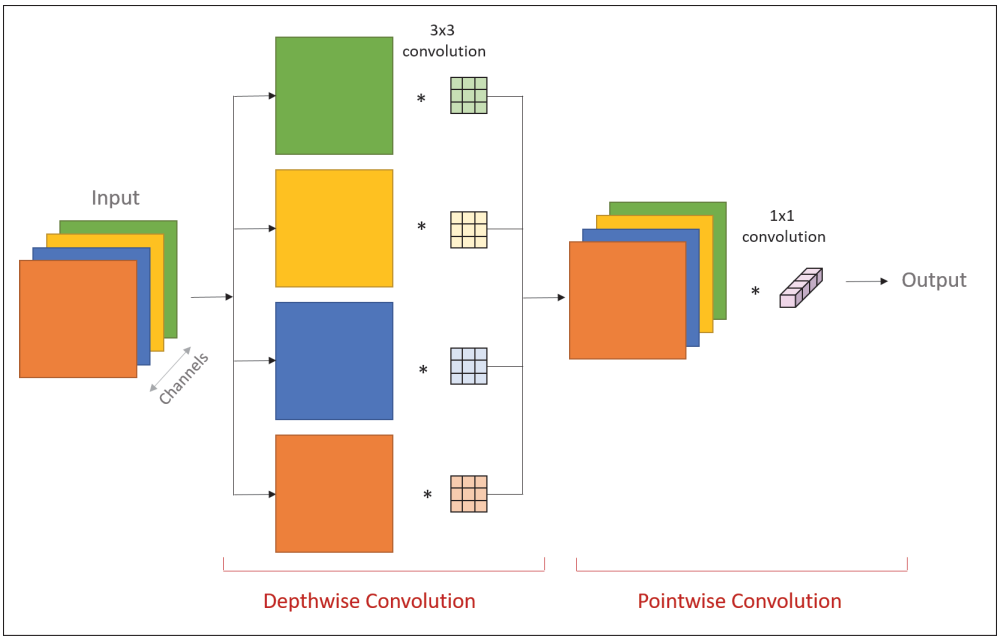


Figure 1.8 Depthwise and pointwise convolutions

convolutions separate these two: depthwise convolution performs spatial filtering by applying one single filter to each input channel independently; pointwise convolution then performs channel mixing by performing a 1×1 convolution. This separation reduces the number of parameters, FLOPs, and latency, making it suitable for mobile and edge devices.

MobileNet-v2 (Sandler, Howard, Zhu, Zhmoginov & Chen, 2018) utilizes residual connections by introducing the inverted residual block and linear bottlenecks. In ResNet bottleneck block, the structure in terms of channels is wide -> narrow -> wide. However, in MobileNet-v1 the depthwise convolution is computationally cheap while the cost of pointwise convolution dominates. Therefore, the narrow -> wide -> narrow channel structure called the inverted residual block is more suitable. Figure 1.9 shows blocks for both versions. In v1, the stride of depthwise convolution $s = \{1, 2\}$. The $s = 2$ replaces maxpooling layer for downsampling feature maps. V2 uses two types of blocks: inverted residuals for $s = 1$ and a block with no residual connection when $s = 2$, since the spatial dimensions of input and output do not match. The ReLU activation is destructive for cases with a low number of channels (narrow layer),

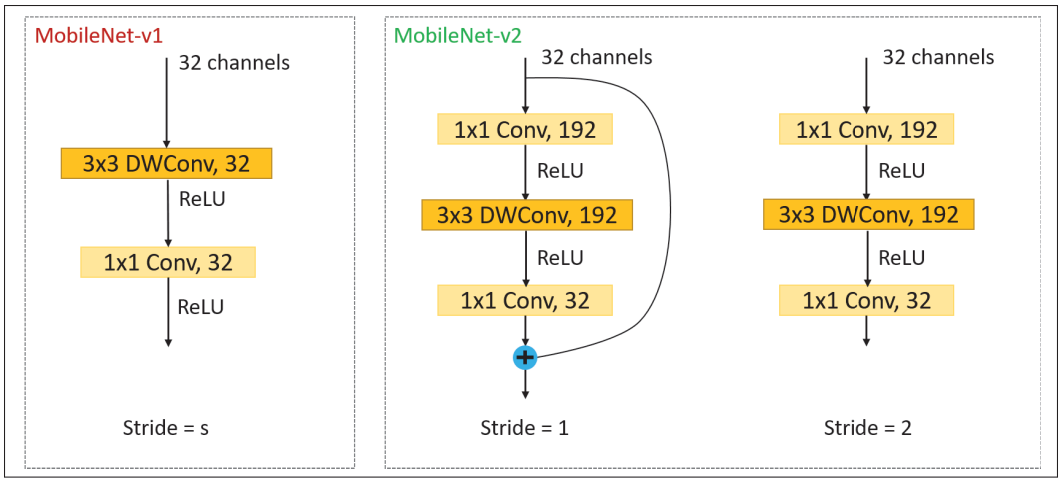


Figure 1.9 MobileNet-v1 and MobileNet-v2 both utilize depthwise convolutions

potentially resulting in significant loss of information. To avoid this in v2, the final projection layer (1×1 convolution) is linear.

MobileNet-v3 utilizes a combination of these layers and a modified swish nonlinearity (Ramachandran *et al.*, 2017; Elfwing *et al.*, 2018).

EfficientNet (Tan, 2019) is a family of CNNs that achieve great performance while being smaller and faster than previous models such as ResNet. NAS was used to search for the optimal kernel size and expansion ratio, and where to place squeeze and excitation (SE) (Hu *et al.*, 2018b) modules for MobileNet convolutional (MBConv) building blocks. In SE modules (figure 1.10), feature maps are shrunk in their spatial dimensions (squeeze). A gating mechanism is used to produce channel-wise weights, which then reweigh (excite) the feature maps. This allows content awareness to the network with a negligible increase in the number of parameters.

An optimum baseline architecture (figure 1.11) is found using NAS. Various models with different capacities and complexity are then generated by scaling this baseline in depth, width, and resolution with a fixed set of coefficients.

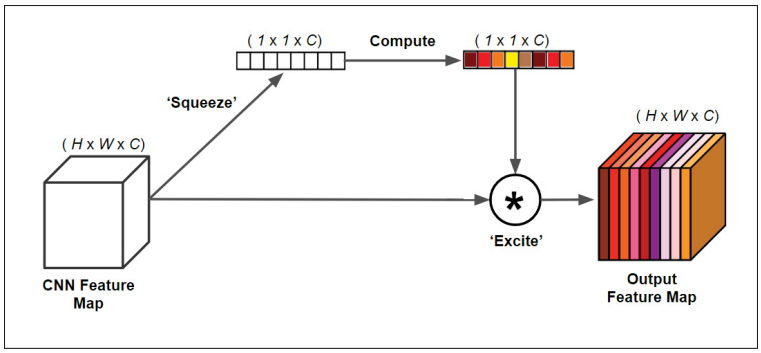


Figure 1.10 Squeeze-and-Excitation mechanism.
Adapted from Hu *et al.* (2018b)

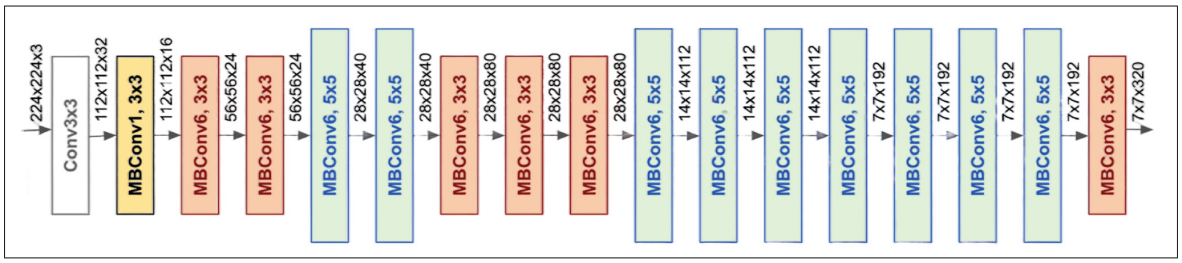


Figure 1.11 EfficientNet architecture found by NAS

1.1.2.2 Semantic Segmentation

In this section, we review architectures used for semantic segmentation. This task involves assigning a label to each pixel of an image. Most modern architectures for semantics segmentation are built upon the encoder-decoder concept:

- **Encoder:** This is the feature extractor that produces a salient code for the input image that shows what is in the image. It is often a pretrained classification network (such as VGG or ResNet) without the final classifier. It progressively downsamples the image using convolution and/or pooling layers while increasing the number of channels.
- **Decoder:** This part takes the encoding and upsamples it to rebuild spatial information to learn the location of features.

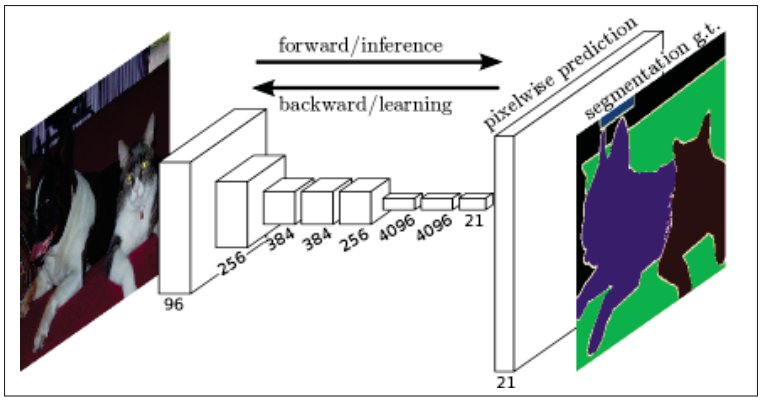


Figure 1.12 FCN generates a segmentation map with one forward pass
Taken from Long *et al.* (2015)

Fully Convolutional Network (FCN) (Long *et al.*, 2015) was the first major work that showed the application of CNN for this task. In this model (figure 1.12), the classifier, i.e., the fully-connected layer, is removed and a 1×1 convolution is instead added to produce the same number of channels as the classes. The transposed convolution then produces the final feature map. A novel skip architecture is also used to combine coarse and finer feature maps.

Figure 1.13 shows another model: DeconvNet (Noh, Hong & Han, 2015). Here, the encoder downsamples feature maps from high to low resolution, encoding information in channels, and loses fine-grained spatial information along this process. However, spatial information is crucial for semantic segmentation. U-Net (Ronneberger *et al.*, 2015), originally designed for medical image an important and widely used model that heavily uses skip connections between corresponding encoder and decoder layers (figure 1.14) to solve this problem. Several modifications and improvements have been proposed for U-Net (Çiçek, Abdulkadir, Lienkamp, Brox & Ronneberger, 2016; Zhou, Rahman Siddiquee, Tajbakhsh & Liang, 2018; Peng, Chen & Sonka, 2025).

DeepLab (Chen, Papandreou, Kokkinos, Murphy & Yuille, 2014) is a family of models that uses the core concept of atrous (dilated) convolutions to replace pooling layers. As shown in figure 1.15, dilated convolutions insert pixel gaps in the convolution kernel, which allows

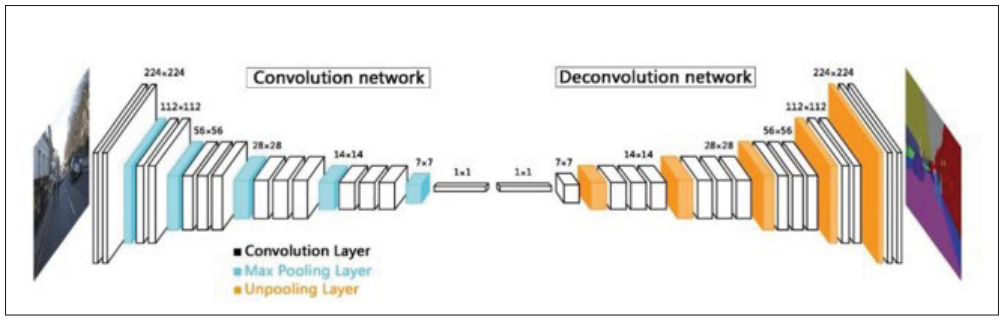


Figure 1.13 DeconvNet uses VGG16 as the encoder with symmetric design for decoder
Taken from Goodarzi (2020)

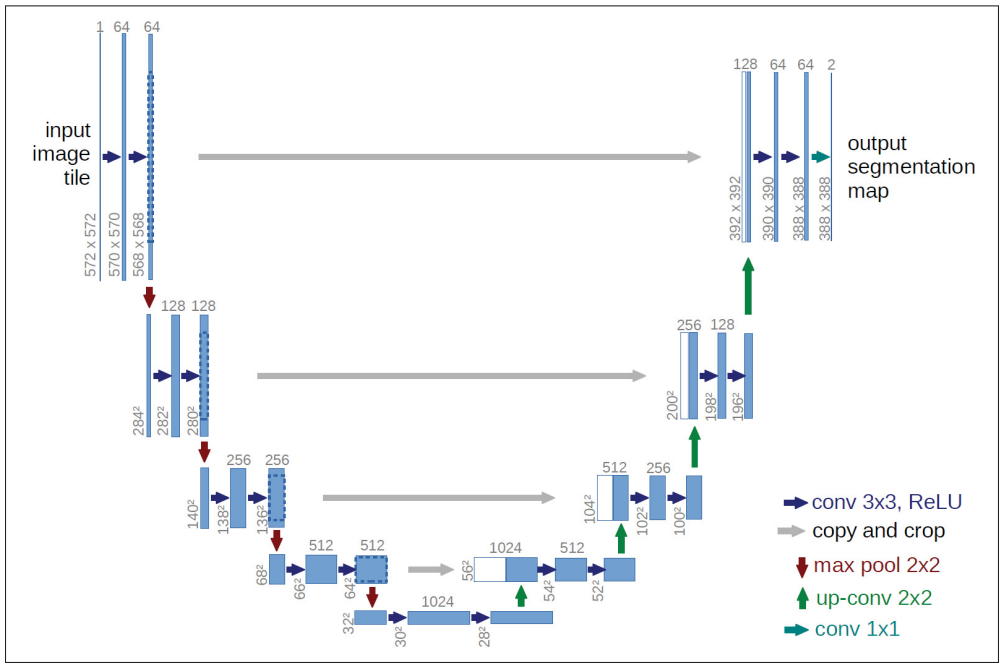


Figure 1.14 U-Net architecture uses skip connections to transfer feature maps between encoder and decoder
Taken from Ronneberger *et al.* (2015)

for an increased receptive field without increasing the number of parameters or aggressive downsampling. Figure 1.16 shows that dilation alongside stride-2 convolutions (subsample) are effective in increasing the effective receptive field. By utilizing dilated convolutions in the

encoder, the model can have a very large receptive field without downsampling the image and losing spatial information.

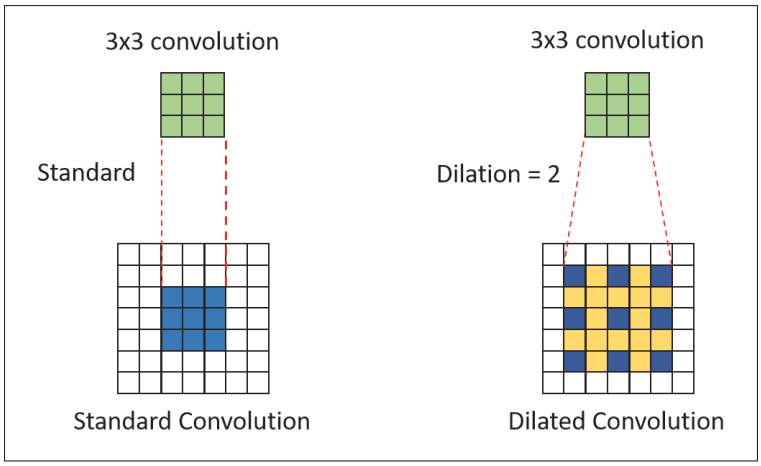


Figure 1.15 Illustration of dilated (atrous) convolution compared to normal (standard) convolution

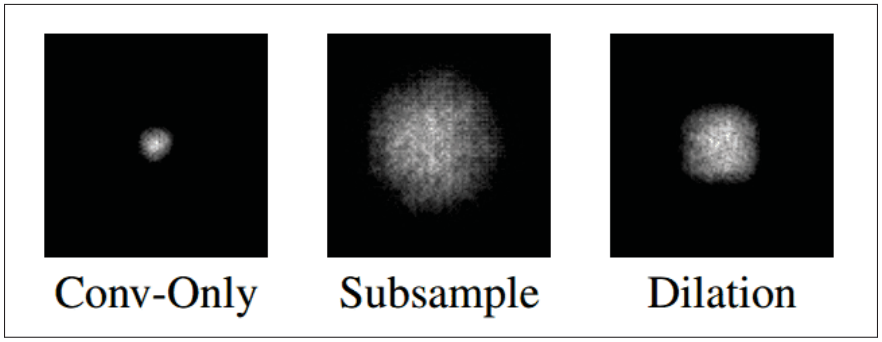


Figure 1.16 Effective receptive field of a 15-layer CNN. Replacing 3 layers with stride-2 convolutions (subsample) or dilated convolutions increases effective receptive field Taken from Luo *et al.* (2016)

DeepLab-v1 used VGG16 backbone and replaced the last few layers with dilated convolutions. To refine the coarse segmentation map, it uses a Conditional Random Field (CRF) as a post-processing step. DeepLab-v2 (Chen, Papandreou, Kokkinos, Murphy & Yuille, 2017a) introduces Atrous Spatial Pyramid Pooling (ASPP) to process objects at multiple scales. ASPP runs several parallel atrous convolutions with different dilation rates on the feature map and subsequently fuses them to form the final output.

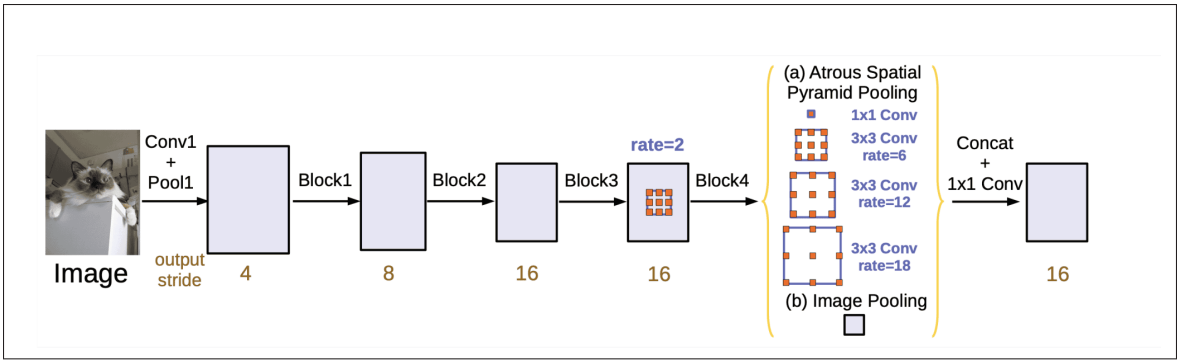


Figure 1.17 DeepLab-v3 architecture adds 1×1 convolution to ASPP and introduces image pooling
Taken from Chen *et al.* (2017b)

DeepLab-v3 (Chen *et al.*, 2017b) significantly improves ASPP by incorporating 1×1 convolutions, batch normalization, and most importantly image-level features global average-pooling (figure 1.17). Finally, DeepLab-v3+ (Chen *et al.*, 2018b) (figure 1.18) architecture is an encoder-decoder model with an encoder similar to DeepLab-v3 and a custom decoder. The decoder upsamples the encodings and concatenates them with high-resolution features from an early layer in the backbone.

Instead of following a downsample-upsample framework, HRNet (Wang *et al.*, 2020b) maintains a high-resolution feature stream in the entire depth of the network. It creates parallel streams of lower resolution as it goes through the layers. At multiple points in the network, different streams exchange information by performing fusion: high-resolution streams provide spatial information, while low-resolution streams provide semantic context.

1.2 Monte-Carlo Tree Search

In this section, we provide a brief overview and background on Monte-Carlo Tree Search (MCTS). MCTS is a fundamental and widely used method and is one of the core components of this thesis as the search method used in chapters 3 and 4 for architecture search. MCTS is a stochastic search algorithm that combines the tree search with Monte-Carlo simulations. Coulom (2006) first used the term MCTS by applying the algorithm in the board game Go. The

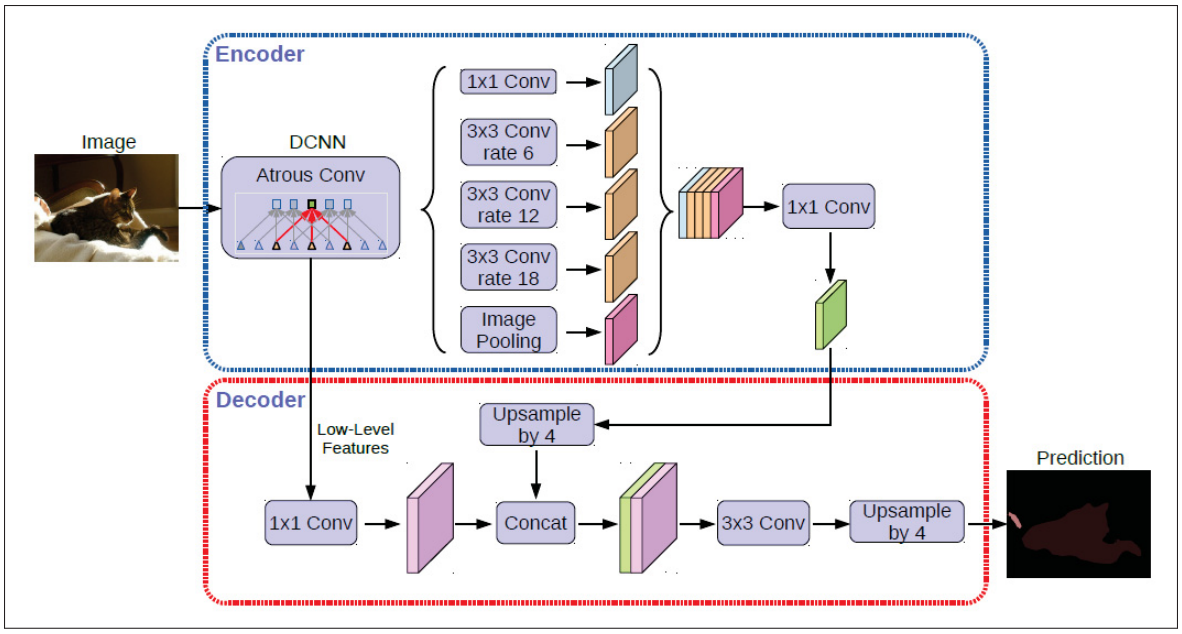


Figure 1.18 DeepLab-v3+ uses a encoder-decoder model and utilizes skip connections to transfer feature maps between encode and decoder
Taken from Chen *et al.* (2018b)

algorithm has roots in the Monte-Carlo method (Metropolis & Ulam, 1949), which uses repeated random sampling to solve deterministic problems.

AlphaGo, developed in 2015 by Google DeepMind (described by (Silver *et al.*, 2016)), was the first to win against the world champion in Go. The neural network was trained on a dataset of past games using reinforcement learning. The follow-up, AlphaGo Zero (Silver *et al.*, 2017), trained only using self-play and requiring no human knowledge. This result is considered a major breakthrough in AI, inspiring many applications of MCTS in deep learning.

MCTS has evolved far beyond its original application in board games to other areas, such as industrial and manufacturing optimization, robotics, supply chain management, and many other fields. In general, MCTS allows us to find the best move/action to take given a certain state of the environment based on the statistics obtained from simulations. This makes it more feasible to navigate large search spaces compared to applying bruteforce approaches.

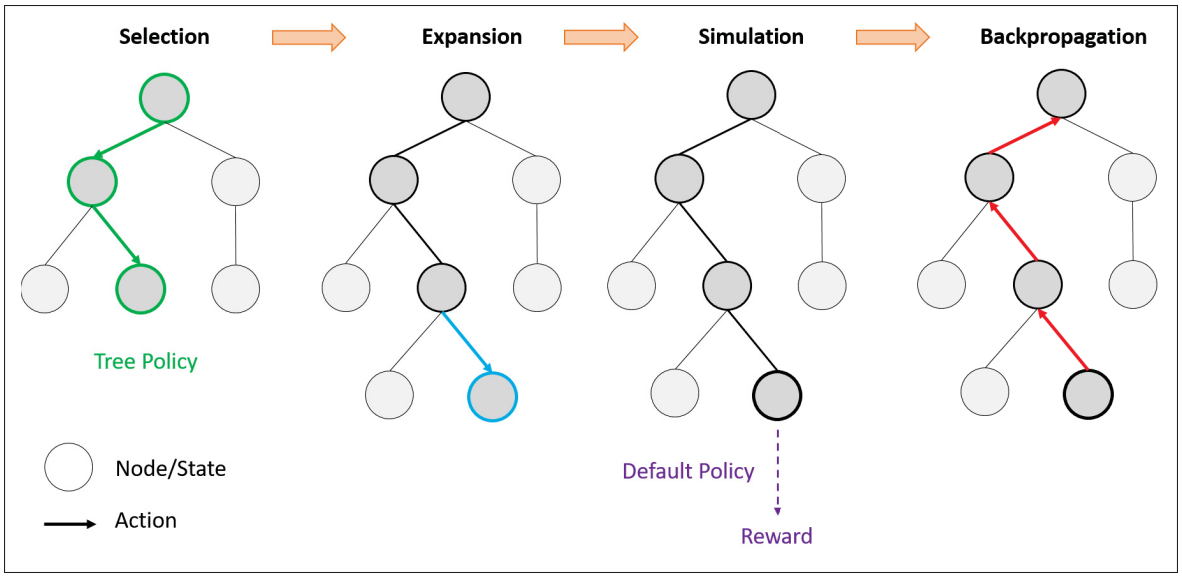


Figure 1.19 Four steps in one iteration of MCTS

1.2.1 MCTS Basics

One iteration of MCTS consists of four steps (Figure 1.19):

- **Selection:** Starting from the root of the tree, the algorithm recursively selects a single node (parent node) based on the *tree policy*. This is equivalent to an action in a decision tree. The selection is performed until the node contains one or more unvisited children (unexpanded nodes).
- **Expansion:** Expands the selected node by adding at least one new node (child) to the tree by applying one of the available actions to the parent node.
- **Simulation/Rollout:** Performs a random complete simulation to obtain the estimated reward (outcome) at a given node.
- **Backpropagation:** The reward is backpropagated to all the selected nodes along the path to the root. Node values and statistics are updated along this path.

These steps are repeated for a specified number of iterations or runtime based on the available budget. After the termination of MCTS, the best action is selected based on the reward statistics. MCTS is an anytime algorithm, meaning that it can be stopped at any time and provide the best

solution based on the current node selection. In general, as more iterations of the algorithm provide more information, it makes better node selection more likely. Another great feature of MCTS is that only the final value at the end of the simulation is required, and intermediate states do not require evaluation.

1.2.2 Upper Confidence Bound for Trees

The tree policy used in the selection step of MCTS aims to provide a balance between exploration of the environment (select nodes that are less visited and not well-tested) and exploitation of the available information (select nodes with the highest reward in the past). It dictates that given a state s (node in tree), how an action a^* (next node) is selected from the set of available actions $A(s)$ (sibling nodes). The most commonly used (and often regarded as the default policy) tree policy is Upper Confidence Bound for Trees (UCT). Introduced by Kocsis & Szepesvári (2006), it is based on UCB1 formula (Auer, Cesa-Bianchi & Fischer, 2002) that treats each node in the tree as a multi-arm bandit problem.

UCT checks each action once and selects one based on:

$$a^* = \arg\max_{a \in A(s)} Q(s,a) + C\sqrt{\frac{log(N(s))}{N(s,a)}} \tag{1.4}$$

where $N(s)$ is the number of times state s has been visited and $N(s,a)$ is the number of times action a is selected in state s. In fact, $N(s)$ is equivalent to the number of times the parent node has been selected. The first term $Q(s,a)$ is the exploitation term, the average reward of selection a from s based on the previous simulations. Therefore, this term favors selecting nodes with higher expected reward. The second term provides exploration by prioritizing selecting less-visited nodes. The hyperparameter C adjusts exploration vs. exploitation and should be finetuned, with generally the default value of $\sqrt{2}$ for normalized Q.

While UCT has been the most used and is often considered the default choice for MCTS, several alternative tree policies have been developed, such as UCB1-Tuned (Gelly & Wang, 2006), EXp3 (Auer *et al.*, 2002), and Thompson sampling (Bai, Wu & Chen, 2013).

1.3 Neural Architecture Search

AutoML aims to automate the entire ML pipeline to reduce the cost and expertise requirements. It allows users with varying levels of expertise to build and deploy ML models efficiently. AutoML covers several topics, such as hyperparameter optimization, data reprocessing, and NAS. NAS has shown great success in recent years in a variety of applications in Computer Vision (CV), Natural Language Processing (NLP), etc. The goal of the NAS is to automatically optimize the architectural hyperparameters of the network.

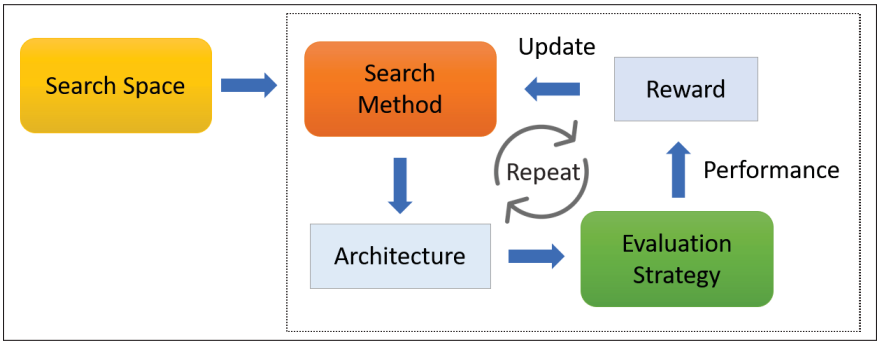


Figure 1.20 General NAS pipeline and its various components

Figure 1.20 shows the general NAS pipeline and interaction of its various components. In general, NAS has three major components:

- **Search space:** defines the set of operation/architecture choices and determines the type of network that is possible to design.
- **Search method:** the approach used to navigate the search space and sample/generate new architectures.
- **Evaluation strategy:** the performance estimation method that determines how candidate architectures are evaluated. This evaluation is used to update the search method.

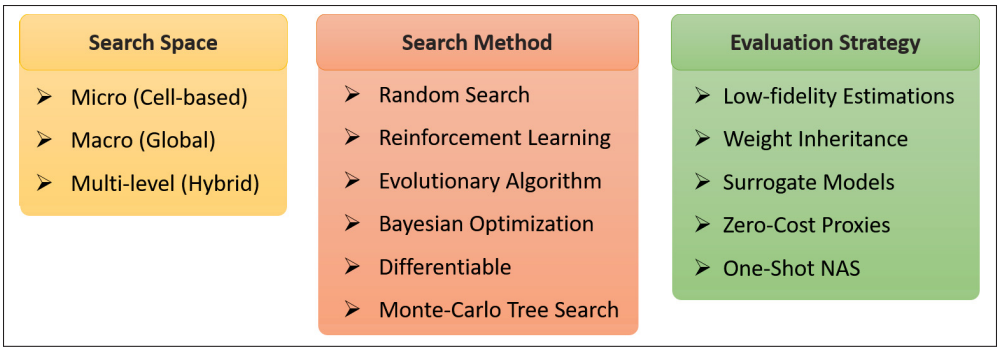


Figure 1.21 Main topics and methods of each NAS component

Figure 1.21 shows the main topics and methods for each component of NAS. In the following sections, we discuss these aspects of NAS pipeline, the search space (section 1.3.1), search methods (section 1.3.2), and evaluation strategy (section 1.3.3). We then focus on one-shot methods (section 1.3.4) as our main NAS framework.

1.3.1 Search Space

The search space is a fundamental component of NAS as it defines what architectures are possible as solutions to the search process and determines the difficulty of the problem. It is the first step in the setup of the NAS pipeline and defines all searchable architectural hyperparameters of the problem. Therefore, careful definition of the search space is crucial: too restrictive search space may exclude many optimal and novel architectures, too large search space will lead to inefficiency of the search algorithm to find optimal solutions in the vast space. In practice, prior knowledge is used to limit the size of the search space by applying constraints on valid architectures and guide the search towards near-optimal models.

Common NAS search spaces can be categorized as Micro (cell-based), Macro, and multi-level. Micro search spaces (Zoph, Vasudevan, Shlens & Le, 2018; Liu *et al.*, 2018b; Pham *et al.*, 2018) focus on yielding the optimal architecture by finding the operations that can produce the best model inside a cell or block. The cell is then stacked to form the entire network, while the outer skeleton of the network is often controlled manually by including reduction cells. This was

Table 1.1 Common CNN search spaces for NAS

Search Space	Characteristics	Examples
Micro/cell-based	Search repeated blocks (cells)	NASNet (Zoph *et al.*, 2018) DARTS (Liu *et al.*, 2018b) ENAS-micro (Pham *et al.*, 2018) NAS-Bench-101 (Ying *et al.*, 2019) NAS-Bench-201 (Dong & Yang, 2020)
Macro/global	Layer-by-layer search (+ channels) (+ spatial dimensions)	LCMNAS (Lopes & Alexandre, 2023) SPOS (Guo *et al.*, 2020b) Macro-Bench-NAS (Su *et al.*, 2021a) ENAS-macro (Pham *et al.*, 2018) OFA (Cai, Gan, Wang, Zhang & Han, 2019)
Hybrid/multi-level	Repeated cells + macro search	AutoDeepLab (Liu *et al.*, 2019a) FasterSeg (Chen *et al.*, 2019a) HNAS (Cheng *et al.*, 2020b)

inspired by observing that the state-of-the-art manual architectures were formed by repetition of a certain structure and helped to reduce the complexity of the search space to a manageable level (White *et al.*, 2023).

Macro search space (Su *et al.*, 2021a) instead searches for the outer skeleton of the network while fixing the operations at the micro level. This can include architecture parameters such as: type of layers, number of layers, or channels in layers, pooling positions, etc. Finally, a multi-level search space searches at two levels: cell and macro structure for a CNN (Liu *et al.*, 2019a) or convolution and self-attention layers for vision transformers (Chen *et al.*, 2021a).

We discuss micro search spaces in section 1.3.1.1 and macro search spaces in section 1.3.1.2. A multi-level or hybrid search space is discussed in section 1.3.1.3. In table 1.1 we summarize different categories of search space.

1.3.1.1 Micro Search Space

Micro or cell-based search spaces are characterized by repetitive architectural patterns called cells or blocks. They have been designed to reduce the search space size to a manageable

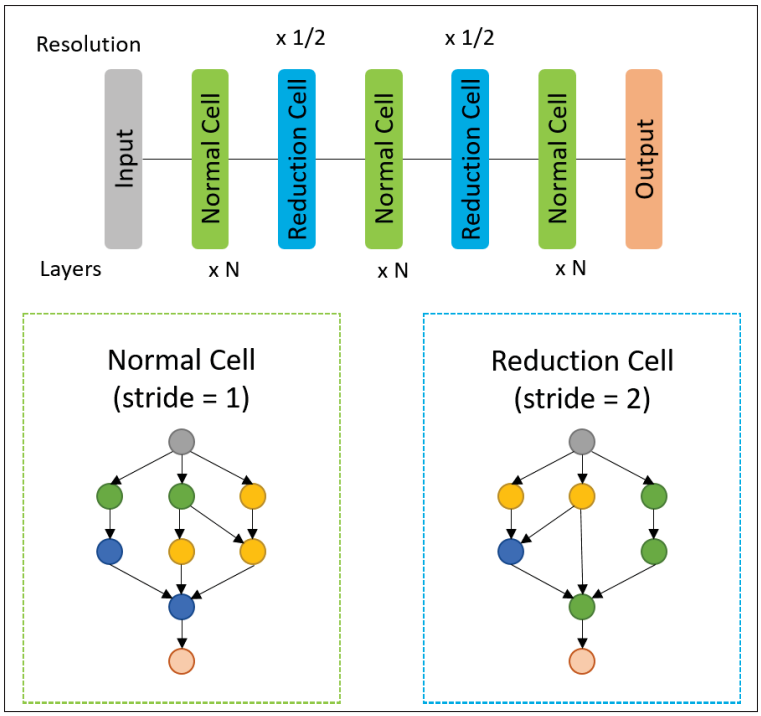


Figure 1.22 Example of micro search space with normal and reduction cells

level by taking advantage of the fact that many manually well-designed architectures (ResNet, MobileNet, etc.) have repetitive cell/block structure. Instead of searching for an optimal network layer-by-layer, an optimal cell is searched, which is then stacked to form the entire network. Figure 1.22 shows the general structure of the micro search space.

Zoph *et al.* (2018) introduced NASNet micro search space and proposed to search for two types of cells: a *normal* cell that maintains the spatial resolution of the feature map, and a *reduction* cell with a stride 2 that downsamples the feature map. The outer skeleton of the architecture (width and depth) is determined by the placement of normal and reduction cells. In this setting, they are manually fixed. Therefore, resolution (feature map size), number of channels, and number of layers (depth) are excluded from the search space.

Micro search spaces have been very popular in recent NAS research (Liu *et al.*, 2018b; Liu *et al.*, 2018a; Zhong, Yan, Wu, Shao & Liu, 2018; Cai, Zhu & Han, 2018b). Various search spaces

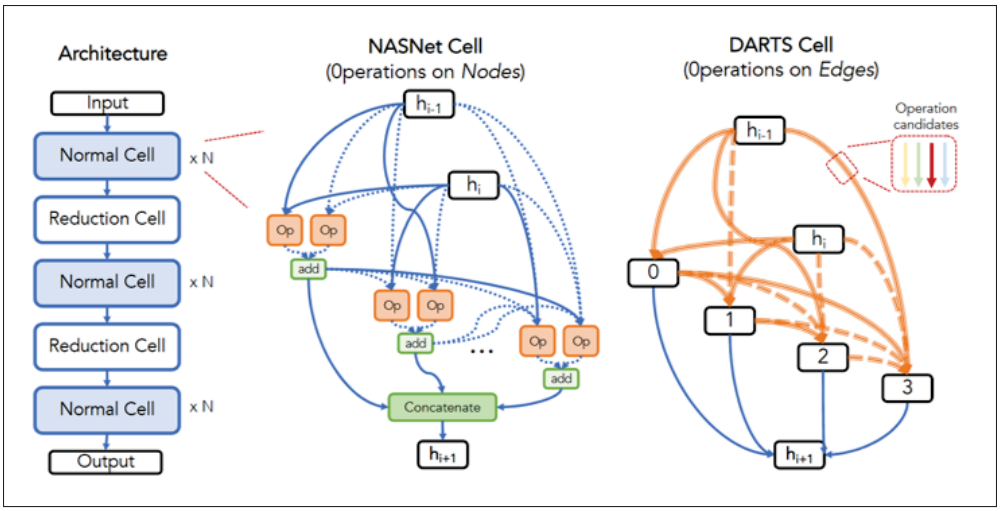


Figure 1.23 Two types of cell structures for micro search spaces
Taken from White *et al.* (2023)

have been built based on this principle, with variation in the outer skeleton structure, layout of the cells, and available operations inside cells.

NASNet (Zoph *et al.*, 2018) and DARTS (Liu *et al.*, 2018b) search spaces are two prominent search spaces and are shown in figure 1.23. A cell can be represented as a Directed Acyclic Graph (DAG). NASNet presents each operation on a node of DAG. With 13 operation choices and 5 available blocks, the number of architectures in the search space is 10^{35}. DARTS places operation choices on the edges of DAG, and nodes represent latent representations. This search space contains 8 edges with 8 operational choices, resulting in 10^{18} architectures. Another prominent micro search space is NAS-Bench-101 (Ying *et al.*, 2019) with 7 nodes, 3 operation choices, and the overall size of 423, 624 architectures. Other common micro search spaces are ENAS-micro search space (Pham *et al.*, 2018) and search spaces of NAS-Bench-201 (Dong & Yang, 2020).

In summary, micro search spaces reduce the complexity and size of the search space and enable the transfer of architectures to various depths by stacking a variable number of optimal cells. The transferability is often used to perform NAS on a smaller dataset and transfer to larger datasets by stacking more cells. These features have made them the most popular search space design with a significant amount of NAS focused on these search spaces (White *et al.*, 2023). However,

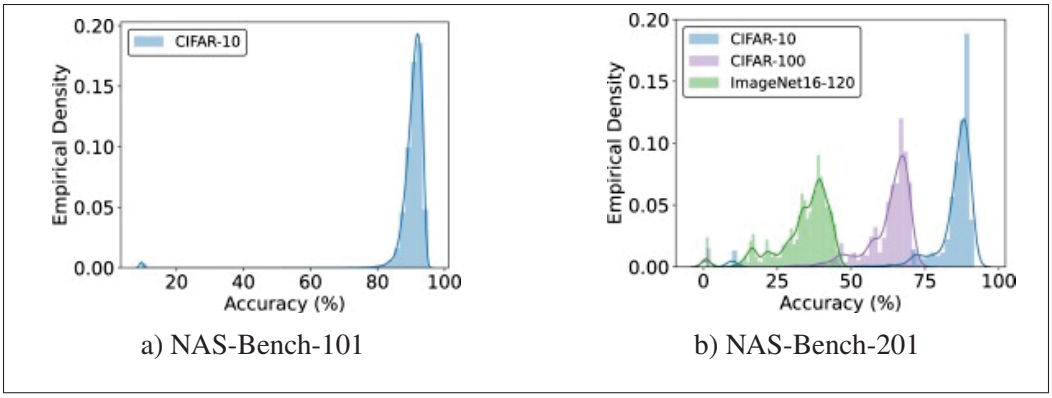


Figure 1.24 Distribution of accuracy and density of architectures in NAS-Bench-101 (Ying *et al.*, 2019) and NAS-Bench-201 (Dong & Yang, 2020) Taken from Lopes *et al.* (2023)

there are several limitations to this design. In essence, the objective of this approach to search space design is to find a cell that performs well in all parts of the network at various depths.

However, it excludes exploration of non-homogeneous architectures, where various depths of networks utilize different operations. Furthermore, several architectural components, such as placement of normal and reduction cells, number of channels, etc., which can contribute to the performance, are excluded from the search space. This limits the expressivity of the search space and often leads to low variance in the entire search space (An example is NAS-Bench 101 shown in figure 1.24), and therefore, the capacity of finding very novel architectures is diminished (Lopes *et al.*, 2023).

1.3.1.2 Macro Search Space

Macro search space focuses on finding the optimal structure of the network (the outer skeleton) without optimizing a repetitive cell structure. In this case, each layer of CNN may use a different operation. Architectures are categorized based on whether the network is chain-structured or allows for complex parallel operations, branches, and skip connections (Figure 1.25). Architectures such as ResNet and DenseNet that allow for skip connections can be considered a special case where the pattern of skip connections is fixed.

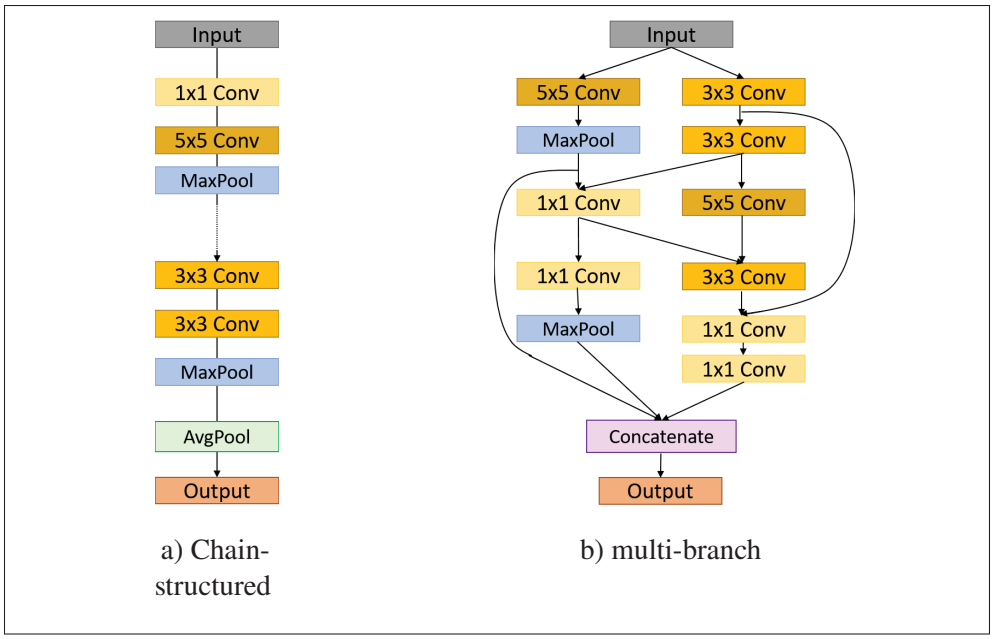


Figure 1.25 Illustration of an architecture from (a) chain-structured and (b) multi-branch macro search space

In several macro search spaces, the choice of operation at each layer of the network, as well as the network topology, is searched (Baker, Gupta, Naik & Raskar, 2016; Real *et al.*, 2017; Zoph & Le, 2016). The choices of operation are pooling, different convolutions, and fully connected layers. Due to the size of the search space, macro search spaces are often not entirely unconstrained, and there are components of the architecture design that are excluded from the space. MobileNet-based Search space is a very popular macro search space with fixed feature map sizes and number of channels per layer. Macro-Bench-NAS (Su *et al.*, 2021a) also manually fixes these components.

OFA (Cai *et al.*, 2019) introduces an elastic search space, where the weight sharing is performed in-place. In this search space, the supernet is the largest network, i.e., the network with the most layers, number of channels, and largest kernel sizes. The smaller architectures are derived by slicing this model. For example, an architecture with 3×3 convolution kernel uses the central part of 5×5 kernel, an architecture with 16 channels uses the first 16 channels of the total 64 channels, etc. This design is especially useful for hardware-aware NAS (see section 1.3.5).

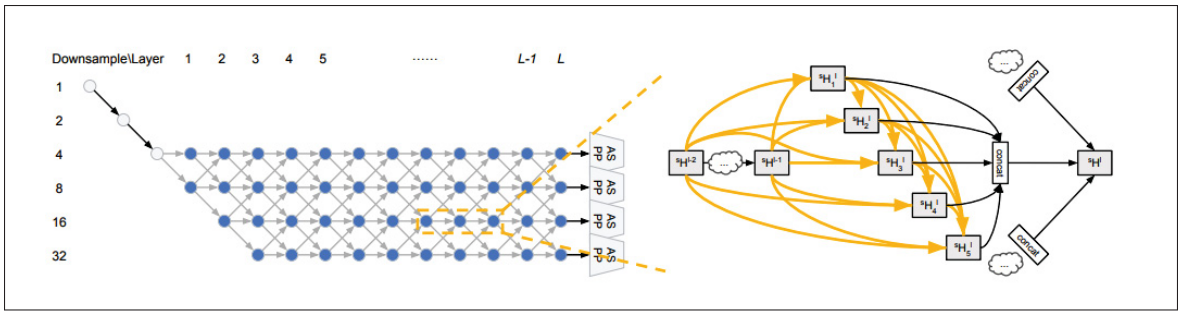


Figure 1.26 Illustration of multi-level search space of Auto-DeepLab. It consists of macro level search space (left) and cell (right) level search space
Taken from Liu *et al.* (2019a)

BigNAS (Yu *et al.*, 2020a) uses similar search space and Optimize the width by learning the expansion ratio, and the resolution by adjusting the input resolution.

1.3.1.3 Multi-Level Search Space

Multi-level (sometimes referred to as bi-level or hierarchical) defines the search space in two levels: the micro level that searches for optimal cell structure, and the macro level that searches for optimal resolution and channels when stacking the cells. Auto-DeepLab (Liu *et al.*, 2019a) is a prominent work that separates the search of macro and micro structure (figure 1.26). At the macro level, the choices are for the spatial resolution to double, halve, or remain the same. The number of channels is then chosen based on the resolution choice. The micro search space is inspired by DARTS search space.

HNAS (Cheng *et al.*, 2020b) applies a similar design for deep stereo matching and HNAS-Reg (Wu & Fan, 2023) similarly separates topology and cell search space for medical image registration. C2FNAS (Yu *et al.*, 2020c) applies Auto-DeepLab principles to 3D images and FasterSeg (Chen *et al.*, 2019a) allows for multiple branches in Auto-DeepLab macro search space.

1.3.2 Search Method

Search method or strategy is the optimization process performed by the NAS algorithm to find the final optimal architecture. It is the methodology that determines how new architectures are sampled or generated and how promising architectures are identified. Therefore, it determines the efficiency of exploring the search space. Classic approaches, such as Reinforcement Learning (RL), Evolutionary Algorithm (EA), as well as random search and MCTS, determine how new architectures are sampled at each iteration of NAS. On the other hand, differentiable (gradient-based) methods determine the mixing weights of each operation at each iteration of NAS. The appropriate search method depends on the specific search space and task, as well as available computational resources. In this section, we discuss existing search strategies, including random search (section 1.3.2.1), RL (section 1.3.2.2), and EA (section 1.3.2.3). We discuss differentiable method later (section 1.3.4.1) after introducing one-shot NAS in section 1.3.4. We will discuss MCTS for NAS in section 3.2.4.

1.3.2.1 Random Search

The simplest NAS search method is random search, where architectures are sampled randomly from the search space. Therefore, it has been used as a baseline for comparison with other NAS methods. Depending on the search spaces, random search can performs comparable to more complex NAS methods: for example Liu, Simonyan, Vinyals, Fernando & Kavukcuoglu (2017) show only around 1% improvement on ImageNet and CIFAR10 when random sampling is replaced by evolutionary search.

1.3.2.2 Reinforcement Learning

The prominent work of Zoph & Le (2016) used RL for architecture search, arguably pioneering the field of NAS. Their work was followed by important early NAS work such as that of Pham *et al.* (2018), Baker *et al.* (2016), Zhong *et al.* (2018), and Zoph *et al.* (2018).

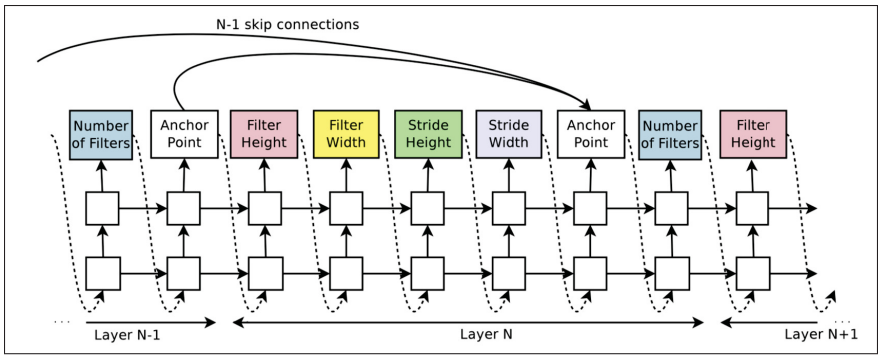


Figure 1.27 RNN controller samples a CNN architecture by predicting various architectural hyperparameters
Taken from Zoph & Le (2016)

The NAS problem is framed as a RL problem, and an agent (controller) is used to build new architectures. The controller outputs actions from the action space, and the complete sequence of actions defines an entire architecture, which is then constructed and evaluated. Figure 1.27 shows the RNN controller sampling a CNN used by Zoph & Le (2016). It makes sequential decisions about various components of a convolution layer, such as filter sizes and strides. The process is repeated for other layers to eventually output an entire architecture.

The validation performance (accuracy) of the sampled architecture is then used as the reward signal for training the controller. The controller learns to get better at sampling architectures with higher expected accuracy as the search progresses. Zoph & Le (2016) trained the RNN controller using REINFORCE policy gradient algorithm (Williams, 1992). Other early NAS works focus on improving RL. Baker *et al.* (2016) used Q-learning algorithm to train the agent and Zoph *et al.* (2018) used the proximal policy optimization algorithm (Schulman, Wolski, Dhariwal, Radford & Klimov, 2017). InstaNAS (Cheng, Lin, Juan, Wei & Sun, 2020a) proposes to sample multiple architectures at each iteration to speed up the training. One of the challenges of these early works was the need to fully train each candidate architecture before evaluation. We will discuss this further in section 1.3.3.

1.3.2.3 Evolutionary Algorithm

EA has historically been used to search for network architectures alongside the weights of the networks (Floreano, Dürr & Mattiussi, 2008) long prior to the emergence of NAS. EA uses the biological principles of genetics and evolution to evolve a population over generations and is a widely used NAS strategy due to its flexibility.

ENAS (Elsken, Metzen & Hutter, 2018) was the first work to use EA for NAS. To generate the initial population, random sampling is often used (Real, Aggarwal, Huang & Le, 2019; Sun, Xue, Zhang & Yen, 2019). EA then selects one or more *parent* architectures from the population and performs evolutionary operations to generate *offspring* (new architectures). These new architectures are then added into the population. The evolutionary NAS methods vary in how they update the population: i) selection strategy that determines which architectures are retained for the next generation or how the parents are selected, ii) evolutionary operations to generate new architectures.

The selection strategy to evaluate the fitness of individual architectures is often based on their validation performance, such as accuracy. Common strategies include: elitism (Golberg, 1989), regularized (age-based) (Real *et al.*, 2019), and tournament selection (Real *et al.*, 2017; Sun *et al.*, 2019; Sun, Xue, Zhang, Yen & Lv, 2020b).

Elitism is the simplest strategy in which the population retains the architectures with the highest fitness value. It can be used to keep a fraction of the fittest individual in the population (Kang & Ahn, 2019) or the fittest as the parent (Elsken, Metzen & Hutter, 2017; Kwasigroch, Grochowski & Mikołajczyk, 2020). Similarly, the worst architectures can be discarded from the population. However, both these strategies can lead to the population being trapped in a local optimum due to the diversity loss in the population. To explore the search space further, an age-based selection can be used to remove the oldest samples from the population (Real *et al.*, 2019; Zhang *et al.*, 2019b), while both strategies can be combined (Zhu *et al.*, 2019a). Tournament selection is a popular method that samples a random set of a few architectures from the population and chooses the fittest in that set as the parents. For example, Real *et al.* (2017)

randomly selected two architectures and discarded the less fit one, and selected the other as the parent for the next generation.

Another aspect of EA is to generate new offspring from the existing architectures. The two most widely used evolutionary operations to achieve this are mutation and crossover. A mutation modifies a single architecture from the population to create an offspring. The mutation choices in the context of NAS can include changing operations (convolution, maxpooling, etc.), changing operation hyperparameters (number of channels, convolution kernel size, stride, etc.), adding or removing layers, or adding skip connections between two layers. The mutations can be chosen randomly (Real *et al.*, 2017, 2019), or more complex methods, such as using an RNN (Chu, Zhang, Ma, Xu & Li, 2021b; Maziarz, Tan, Khorlin, Georgiev & Gesmundo, 2018) and Gaussian mutations (Lorenzo & Nalepa, 2018) are used to guide the mutations.

Cross-over (or recombination) generates an offspring based on the combination of two architectures. The simplest and most commonly used for NAS (Gibb, La & Louis, 2018; Hu, Sun, Li, Wang & Gu, 2018c) is single-point cross-over (Mitchell, 1998).

EA-based NAS is versatile and applicable to complex discrete search spaces with various constraints; however, it can be computationally inefficient as they require the evaluation of a large number of architectures.

1.3.2.4 Bayesian Optimization

Bayesian optimization is a powerful method for hyperparameter optimization. It is an iterative algorithm based on a probabilistic surrogate model that is constructed based on previous observations. It uses an acquisition function to generate new architectures based on their predicted performance from the surrogate model. The candidate architecture is trained, evaluated, and its performance is used to update the surrogate model.

Many frameworks are based on Gaussian processes that operate in low dimensional continuous space, making the application to NAS not straightforward. Therefore, several works project

the architecture into a continuous latent space (Ru, Esperanca & Carlucci, 2020) or use graph neural networks (Wan, Ru, Esparança & Carlucci, 2022a). NASBOT (Kandasamy, Neiswanger, Schneider, Poczos & Xing, 2018) uses Gaussian process and develops a distance metric for graphs using optimal transport. BANANAS (White, Neiswanger & Savani, 2021) uses a neural network as a surrogate model, taking an architecture as an input and predicting its accuracy. It uses graph-based representations to capture the architecture information.

Bayesian methods are very promising with lower computational cost compared to RL and EA methods, however their effectiveness relies highly on the choice of acquisition function and surrogate model.

1.3.3 Evaluation Strategy

In this section, we discuss strategies that are used to evaluate the performance of candidate architectures. This has been the main bottleneck of NAS (Ren *et al.*, 2021), especially in early NAS works (Baker *et al.*, 2016; Real *et al.*, 2017; Zoph & Le, 2016; Xie & Yuille, 2017). These works fully trained each architecture, i.e. each candidate architecture is trained from scratch until convergence and then evaluated. This is the simplest way that provides the true performance of the architecture, however training many candidates during the search becomes very costly. For example, NASNet (Zoph *et al.*, 2018) used 500 GPUs and 4 days, while Real *et al.* (2019) used 450 GPUs for 7 days for NAS.

Since these early works, several approaches have been used to reduce the evaluation cost: low-fidelity evaluations, weight inheritance, surrogate models, zero-shot methods, and one-shot methods. In this section, we briefly discuss the first few methods as well as an overview of NAS benchmarks. We leave the detailed discussion of one-shot methods for section 1.3.4.

Low-fidelity estimations: These techniques aim to approximate the actual performance of architecture by training with a reduced computational setting. Training for fewer epochs (early stopping) (Klein, Tiao, Lienart, Archambeau & Seeger, 2020) has been used as a proxy for full training. This assumes that the performance of architectures after a few epochs of training

correlates with their performance when fully trained. This assumption ignores the various rates of convergence of different architectures and generally punishes architectures that are slow learners.

Another approach is using performance on a proxy task. This can include training and evaluation on a small subset of training data (Klein, Falkner, Bartels, Hennig & Hutter, 2017; Zoph *et al.*, 2018) or lower resolution images (Chrabaszcz, Loshchilov & Hutter, 2017). Another option is reducing the model complexity (fewer filters in layers or fewer layers) (Real *et al.*, 2019). While efficient, generally the resulting architecture performance approximations from these methods may have low correlation with the ground truth obtained from full training (Zela, Klein, Falkner & Hutter, 2018) and are inconsistent for various search spaces and computational reduction settings (Zhou *et al.*, 2020).

Weight inheritance: The weights from a trained architecture can be transferred to a new architecture. This allows a better initialization for new architectures and reduces the training iterations required for them. For example, Real *et al.* (2019) uses the weights of parent architectures in EA to initialize the weights of the offspring.

Surrogate models: An external surrogate model can be used to efficiently predict the performance of candidate architectures. Liu *et al.* (2018a) trained a predictor model based on its architectural properties and used this model to select candidate architectures for training. Another option is to predict the final performance by extrapolating the learning curve after partial training. If a candidate is predicted to perform poorly, its training will be stopped (Baker, Gupta, Raskar & Naik, 2017). The main challenge of these methods is that they require reliable prediction based on very few evaluations in a very large space of NAS (Elsken, Metzen & Hutter, 2019).

1.3.3.1 Zero-Shot NAS

The goal of the zero-shot NAS is to bypass the expensive training part of traditional NAS by using *zero-cost proxies* that can estimate the performance of the architecture at initialization.

These approaches are training-free, and bypassing the training of architectures makes them lightweight and very computationally efficient. These proxies are often developed by theoretical analysis of neural network mechanisms and empirical evidence. They often use a single forward and backward pass on a minibatch of data with negligible computational cost (hence zero-cost). Some proxies do not even require any data. Their low cost (an architecture can be evaluated in milliseconds) and the fact that high-end GPU clusters are not needed have gained significant popularity in NAS research in the past few years.

Mellor, Turner, Storkey & Crowley (2021) is widely credited with first introducing the concept by estimating the performance of an architecture without training for NAS. Another influential work by Abdelfattah, Mehrotra, Dudziak & Lane (2021) systemically evaluated several pruning methods at initialization as a proxy for architecture performance.

Several zero-cost methods measure the importance of layers or operations based on the gradients at initialization. GradNorm sums the norm of the gradient vector for each layer. A minibatch of data is used as the input of the network, and l_2-norm of convolutional and linear layers is calculated after back propagation. This norm is used to assess the learning potential of an architecture, with a small norm indicating difficulty in learning.

SNIP (Lee, Ajanthan & Torr, 2018) approaches this problem from the perspective of pruning the network at initialization. It uses a saliency criterion to measure the sensitivity of performance to certain operations and connections. Larger changes in loss correspond to more importance of the weight. GraSP (Wang, Zhang & Grosse, 2020a) instead focuses on changes in gradient norm instead of loss by evaluating how perturbations in parameters affect the gradient. Architectures that maintain or increase their gradient signal are predicted to be more probable to learn effectively. SynFlow (Tanaka, Kunin, Yamins & Ganguli, 2020) is a data-agnostic approach that avoids the problem of layer collapse in previous methods. It instead focuses on gradient flow through the network, looks at the product of weights in an architecture, and identifies bottlenecks in the gradient flow.

Fisher score (Turner, Crowley, O'Boyle, Storkey & Gray, 2019) is another pruning technique that predicts the importance of parameters based on their contribution to the objective function using the summation of the fisher score in the network. Jacobian covariance (Lopes, Alirezazadeh & Alexandre, 2021) measures the correlation of activations across different inputs in a minibatch. Lower correlation indicates better generalization ability of the network, as it is able to adequately differentiate between various inputs.

NASWOT (Mellor *et al.*, 2021) measures the capability of architecture using ReLU activation patterns. It assigns a binary code to indicate whether a non-linear unit is active (non-negative) or inactive (negative). The distance of these binary codes for different inputs shows the ability of the architecture to learn to separate different inputs, and therefore its potential.

Zen-Score (Lin *et al.*, 2021) estimates the gradients of the output with respect to the input. The input is random Gaussian noise, making the method data independent. ZiCo (Li, Yang, Bhardwaj & Marculescu, 2023b) proposes that the relative change of gradients early in training is a strong indicator of the training loss at convergence. It calculates the ratio of mean to the standard deviation of the gradients across different layers in the network, with a higher score indicating a more promising architecture.

The trainability (how effectively a network can be trained) of a neural network can be estimated using Neural Tangent Kernel (NTK) (Jacot, Gabriel & Hongler, 2018). TE-NAS (Chen, Gong & Wang, 2021e) analyses the spectrum of NTK and the number of linear regions in the input to assess both trainability and expressivity (how complex is the function that the network can represent) of architectures and ranks them. ETE-NAS (Rumiantsev & Coates, 2023) proposes a training-free framework that uses a set of ranking functions based on neural network Gaussian process kernel (Lee *et al.*, 2017). They use a combination of ranking functions to accelerate NAS evaluations without requiring backpropagation.

Finally, another zero-cost proxy is simply using the number of parameters or FLOPs. Some work has shown that in some cases they perform surprisingly well (Ning *et al.*, 2021; White *et al.*, 2022) and can be used as the baseline.

1.3.3.2 NAS Benchmarks

While not an evaluation strategy, NAS benchmarks are an important part of NAS. They are used to evaluate for a given method the quality and the amount of computation required to yield a solution. They have played a crucial role in the NAS community as they provide evaluation of all architectures in a brute-force way to find the optimal solution and eliminate the need to run this expensive process independently (Chitty-Venkata, Emani, Vishwanath & Somani, 2023). They facilitate comparison of various methods on a standard benchmark and help the reproducibility of NAS research. The development of NAS benchmarks has also improved the reproducibility and efficiency of NAS research. Tabular benchmarks (Ying *et al.*, 2019; Su *et al.*, 2021a) are constructed by exhaustively training and evaluating (metrics such as accuracy, FLOPS, number of parameters, etc.) all possible architectures.

On the other hand, surrogate benchmarks (Siems *et al.*, 2020) estimate the architecture performance using a model that is trained on data from several trained architectures. Benchmarks have been developed for both micro (Ying *et al.*, 2019; Dong & Yang, 2020; Siems *et al.*, 2020) (with addition of channels (Dong, Liu, Musial & Gabrys, 2021)) and macro (Su *et al.*, 2021a; Roshtkhari, Toews & Pedersoli, 2023) search spaces.

1.3.4 One-Shot Methods

One of the most important and widely used NAS methods is one-shot method based on weight-sharing. Its goal is to train a set of shared weights among architectures in the search space and thus avoiding training each candidate individually and its cost. The search space is defined as an overparameterized supernetwork or *"supernet"*, from which every possible architecture (or *subnet*) can be derived. A depiction of supernet for macro search spaces is shown in figure 1.28, where each architecture is a path in the supernet.

After training the supernet, the candidate architectures are evaluated without any additional training by directly inheriting the weights from the supernet. This eliminates any additional training cost and provides much more efficient evaluation estimation. The concept of supernet

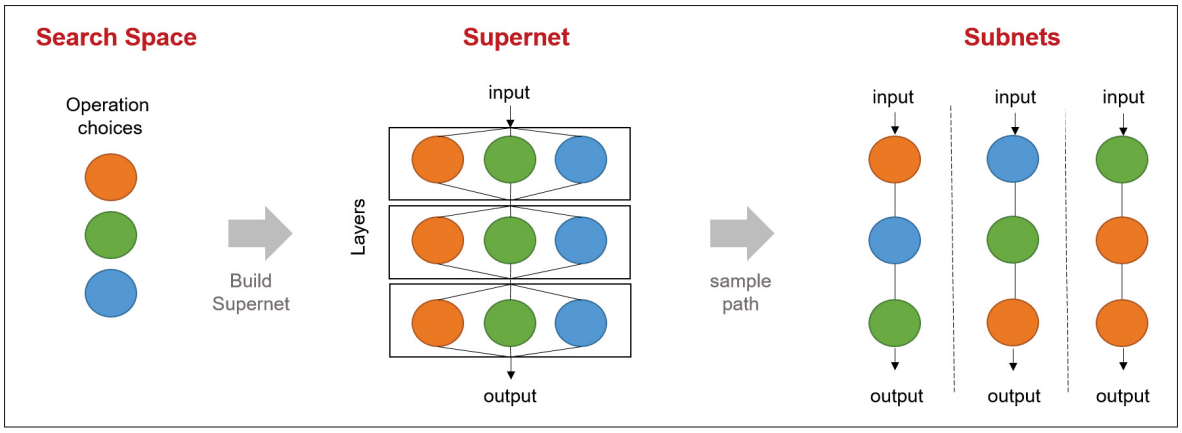


Figure 1.28 Depiction of supernet for a macro search space and sampled subenets (architectures)

has its foundation on Convolutional Neural Fabrics (Saxena & Verbeek, 2016) a predecessor work. It was one of the pre-NAS works that attempted at automating network design by finding optimal path in a 3D trellis structure of layers, kernels, and scales. The idea of supernentwork was first adapted for NAS by Pham *et al.* (2018). The supernet provides a cost effective evaluation method method and can be used with various search strategies. In the next sections we discuss differentiable and non-differentiable NAS methods that utilize one-shot framework.

1.3.4.1 Differentiable Methods

Differentiable (or gradient-based) NAS method was first proposed as Differentiable Architecture Search (DARTS) by Liu *et al.* (2018b). It has been one of the most popular and impactful NAS methods that relaxes the discrete architecture search space into an equivalent, but continuous one. In the discrete search space, the gradient cannot backpropagate to the operation (or architecture choice) and therefore fast and optimal gradient descent algorithms cannot be used to solve the problem. DARTS proposes to instead of selecting one operation/architecture, a mixture of all candidates at every edge of the supernet is used. Figure 1.29 shows DARTS in original form for a cell in micro search space.

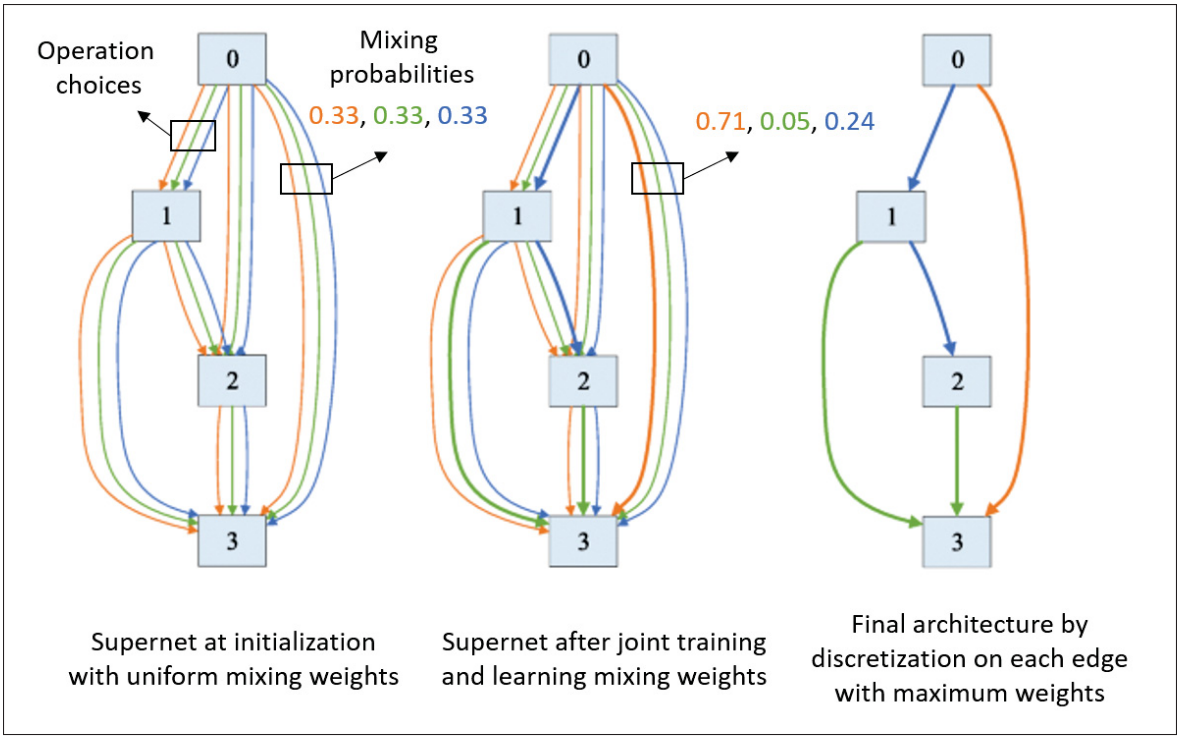


Figure 1.29 Differentiable architecture search (DARTS)
Adapted from Liu *et al.* (2018b)

In this case, mixture weight (architecture parameter) therefore can be optimized using gradient descent. DARTS proposes a bi-level optimization algorithm that optimizes both the supernet weights and the mixture weight. Due to the simplicity and efficiency of DARTS, it has been one the mostly used NAS methods with many follow-up works aiming to improve the original DARTS approach.

One of the problems for DARTS is that it heavily favors skip connections among operations during training (Liang *et al.*, 2019; Wang, Cheng, Chen, Tang & Hsieh, 2021c; Zela *et al.*, 2019). This results in the final architectures which heavily feature skip connections. Chu, Zhou, Zhang & Li (2020c) suggest that this is due to skip connections compensating for vanishing gradients (similar to residual connections in Resnet), while the increase in its mixing weights harming other operations due to softmax normalization. They propose to replace softmax with sigmoid and render operation weights independent from each other. Chu *et al.* (2020b) propose

to separate skip connections by manually adding them to the supernet. Wang *et al.* (2021c) address this issue implicitly by changing how final architecture is selected. They propose to instead of selecting the operation with maximum parameter weight for each edge, to select the operation which its removal harms the performance the most.

Another challenge is large memory requirement for training the supernet compared to an individual architecture as it requires storing a large number of activations and gradients. One solution is that at each iteration of training parts of the supernet are masked and not trained. ProxylessNAS (Cai *et al.*, 2018b) masks all operation on the edges except one by modification of BinnaryConnect discretizations method (Courbariaux, Bengio & David, 2015). The probability of keeping an operation is determined by its architecture weight. PC-DARTS (Xu *et al.*, 2019b) samples only a portion of channels to reduce the memory cost. SNAS (Xie, Zheng, Liu & Lin, 2018) uses Gumbel-Max trick (Maddison, Tarlow & Minka, 2014) to allow backpropagation while using discrete samples to reduce both memory and entanglement of architecture parameter during training.

Another issue is the architecture performance gap upon discretization at the end of the training. Similar to SNAS, GDAS (Dong & Yang, 2019b) uses Gumbel-softmax sampling to samples only one operation per edge to reduce the performance gap as well as memory requirement. Li *et al.* (2020d) propose Geometry-Aware Gradient Algorithm (GAEA) to encourage sparse architecture parameters and to reduce the gap.

1.3.4.2 Sampling-Based One-shot Methods

The other group of one-shot NAS methods operate in the discrete search space and are based on sampling one or multiple architectures at each iteration of supernet training. This alleviates the memory and computational cost of differentiable approaches. Due to their competitive performance and lower cost, they have become one of the most widely used NAS methods. Specially Single-Path One-Shot (SPOS) that samples a single architecture at each iteration of training has become a standard NAS method.

The weight sharing among architectures and joint training of these weights in a supernet at the core of one-shot NAS. As mentioned before, the main benefit of this mechanism is the direct inheritance of weights for evaluation of each architecture and greatly increasing efficiency. However, in this setting a set of weights (e.g. convolution kernel) is sampled in very different paths. The gradient updates of these different path can be in different directions causing co-adaptation and interference (Xu *et al.*, 2022; Zhang, Li, Pan, Chang & Su, 2020a; Zhao *et al.*, 2021a). Therefore, the shared weights struggle to adequately estimate the independent training of architecture. Furthermore, gradient updates from next sample may cause the supernet to override (or forget) previous sampled (termed multi-model forgetting in NAS). This degrades the reliability of performance estimation of supernet and cause the performance estimated from the supernet to have low rank correlation with ground truth. This is a fundamental challenge as the effectiveness of one-shot NAS is dependent on the ranking ability of the supernet.

Many research in recent years has been dedicated to mitigate this problem. In general these research can be categorized to three categories: 1) directly reducing weight sharing by using multiple supernets. 2) directly reducing gradient conflict 3) modifying sampling to focus on parts of search space.

One solution is few-shot methods that partition the search space and use separate supernets for each partition, therefore reducing the weight sharing among architectures. Few-shot NAS (Zhao *et al.*, 2021a) proposes to partition the supernet along the operation edges, making the architectures that have different operations on these edges do not share weights. They select split edges randomly show improved results with only few split edges. Hu *et al.* (2022) argue that weight sharing among architectures are not equally harmful. They proposes that similar operations on the split edge can be grouped together, while very different operations should be assigned to different supernets. They improve simple edge partitioning of few-shot NAS by proposing partitioning based on gradient similarity. K-shot NAS (Su *et al.*, 2021b) uses a dictionary of different weight matrices per operation instead of one shared weight. The actual weight is then a weighted combination of the dictionary.

Some methods attempt to directly reduce the gradient conflict among architectures. Benyahia *et al.* (2019) and Zhang *et al.* (2020a) use regularizer to limit the weights to the vicinity of the learned distribution of previously trained architectures. Zhang *et al.* (2020a) specifically prevent the training on the new architecture to degrade the performance of a representative set of previously sampled architecture. Additionally they implement a diversity maximization scheme to ensure a varied set of representative architectures.

SUMNAS (Ha, Kim, Park & Chun, 2021) trains the supernet to learn met-features that work well for entire search space rather than individual sampled architecture. Subnet-aware sampling (Jeon *et al.*, 2025) show that the single momentum of supernent optimizer is noisy and inadequate in representing all architectures. They propose using separate momentum for different cluster of architectures using structural properties. PA&DA (Lu *et al.*, 2023) attribute the ranking inconsistency of supernet to high gradient variance during training caused by both the architecture and the data minibatch. They propose adjusting both path and sampling distribution to decrease the variance improve generalization. In general, the methods that modify the sampling distribution of architecture and increasingly focus on narrower parts of search space (You *et al.*, 2020; Su *et al.*, 2021a) implicitly reduce weight sharing.

Sampling multiple path and aggregating their gradients before optimization step can help stabilize supernet training (Chu *et al.*, 2021c; Yu *et al.*, 2020a), however they generally incur more cost as they require multiple forward/backward passes per optimization step.

Another approach is to ensure fairness in training of supernet, i.e. all architectures/operations are trained equally. Uniformly sampling architectures used in SPOS (Guo *et al.*, 2020b) is often treated as a baseline that ensures basic fairness. FairNAS (Chu *et al.*, 2021c) applies a more strict fairness policy to ensure different weights are updated for same number of times. Other approaches tackle the fairness in sampling from the architecture complexity perspective and ensuring different architectures receives adequate training. Generally, models with larger number of parameters or higher FLOPs require more training steps than less complex models. Subnet-aware sampling (Jeon *et al.*, 2025) proposes a complexity-aware learning rate scheduler

that adapts the learning rate by decaying the learning rate slower for more complex architectures. ShiftNAS (Zhang *et al.*, 2023) shows that uniform sampling focuses on regions in search space that have intermediate computational cost and proposes to adjust sampling probability based on resource consumption.

1.3.5 Advanced NAS Methods

In this section we briefly discuss some of more advanced NAS methods that have gained popularity in recent years. Hardware-aware NAS extends traditional NAS to incorporate hardware-specific constraints into the search process. Transferable NAS aims at transferring NAS across tasks and domains to reduce computational cost. NAS for Multi-task Learning (MTL) extends NAS to find architectures specifically for MTL.

Hardware-aware NAS: One of the challenges of manually designing a neural network architecture is to respect the constraints of the target devices as well as the performance of the architecture. In single objective NAS, the only optimization objective is the performance. However, for many real world applications, a multi-objective NAS approach is needed to optimize other objectives for deployment on resource constraint devices.

The simplest form is pruning the search space with a hard constraint (e.g. FLOPs budget) by rejection sampling (Guo *et al.*, 2020b; Su *et al.*, 2021a). Another approach is to train a single supernet once and derive smaller architectures that satisfy hardware constraints during deployment (Cai *et al.*, 2019; Yu *et al.*, 2020a). OFA (Cai *et al.*, 2019) trained a large supernet and progressively shrinks it to smaller subnets. For a new hardware, a quick evolutionary search is performed to find the best performing slice. Similarly BigNAS (Yu *et al.*, 2020a) uses the sandwich rule (Yu & Huang, 2019) for supernet training to train both the lowest complexity and highest complexity architectures and improve lower and upper bound of performance.

Another approach is to incorporate the constraint as a regularizer to the objective function (Wu *et al.*, 2019a; Cai *et al.*, 2018b). The cost of an architecture can be estimated based on a look-up

table or predicted using a surrogate model. The hardware penalty $\mathcal{L}_{hw}$ can then be simply added (with the scaling factor λ controlling the trade-off) to the task loss (e.g. cross-entropy):

$$\mathcal{L}_{total} = \mathcal{L}_{task} + \lambda \cdot \mathcal{L}_{hw} \tag{1.5}$$

The search is sensitive to λ and can become easily biased towards skip connections as have no hardware cost making it easy to satisfy the regularizer without necessarily contributing to the accuracy. HW-EvRSNAS (Sinha *et al.*, 2024a) re-frames the problem by considering a known well-performing architecture as a reference and searching for architectures with closest similarity that satisfy hardware constraints. MO-HDNAS (Sinha, Rostami, El Rahman Shabayek, Kacem & Aouada, 2024b) propose to add a hardware cost diversity as the third objective to facilitates better exploration of the architecture search space.

Transferable NAS: In traditional NAS, the search process for a given task often starts from the scratch. Transfer learning has been used for NAS to reduce the computational cost and speed up NAS process, by transferring the knowledge accumulated during architecture search from one task/dataset to others (Lu *et al.*, 2021; Elsken, Staffler, Metzen & Hutter, 2020). The transfer can be applied to a different dataset (e.g. CIFAR to ImageNet), different task (e.g. classification to segmentation/detection), or different domain (e.g. natural to medical images).

A commonly used method is to transfer final architectures searched on smaller datasets (e.g. CIFAR) to larger datasets such as ImageNet (Lu *et al.*, 2020). Several work utilize searching on a smaller proxy datasets to save computational cost specially in cell-based search spaces.

Another approach is transferring architectures across different tasks. NAS for segmentation or detection is often challenging due to larger and more complex search spaces. An architecture searched for classification task can be used as the backbone such as YOLO (Redmon, Divvala, Girshick & Farhadi, 2016). The classification head is replaced with the task-specific head (e.g. feature pyramid network). Many of the works that transfer architectures across datasets or tasks focus on micro or cell-based search spaces. To accommodate larger datasets, often number of

layers are increased. Zoph *et al.* (2018) transferred the learned cells on CIFAR10 dataset to ImageNet for classification and to COCO dataset (Lin *et al.*, 2014) for object detection.

TransNAS-Bench-101 (Duan *et al.*, 2021) is a benchmark designed specifically for transfer NAS methods and assess performance of architectures across seven tasks. They found that architectures perform very differently across different tasks and macro structure of the architecture plays a larger role in the performance for some tasks such as semantic segmentation.

Other approach is to utilize the NAS model from the original task to guide the search for target task. Lu *et al.* (2021) initialize the supernet on ImageNet that is adapted to search for task-specific subnets. The show that small scale datasets benefit greatly from this approach. Elsken *et al.* (2020) integrates NAS with gradient-based meta learning to adapt architectures to new tasks in a few steps. Another approach is to transfer the information learned from the search process rather than the architecture itself. (Zhou, Wu, Feng, Lu & Tan, 2025) use LLMs to extract design rules from the source task and use them to reduce the search space for the target task.

NAS for Multi-task Learning: Multi task learning (MTL) involves training a single model to tackle multiple tasks simultaneously. The core idea is that instead of training separate models specialized for each task, one model can use the shared information among related task. This helps improve the generalization by using shared information, improve data efficiency, faster convergence and reducing model size, and improving overfitting. MTL is useful to reduce hardware requirements for edge devices and real -time inference, or when there are limited labeled data for the task. Architectures used for MTL commonly follow two structures:

- **Hard parameter sharing:** The most common architecture is to share all hidden layer of the network between across all tasks (Figure 1.30). This extracts general features that are the input of task-specific last layers (head). This is very efficient as a significant portion of parameters are shared and overfitting is significantly reduced. They are best for cases when tasks are highly related.

- **Soft parameter sharing:** Each task has it's own model and parameters and the parameters are regularized by enforcing similarity to promote information sharing among tasks. They are often used when hard parameter sharing is too restrictive.

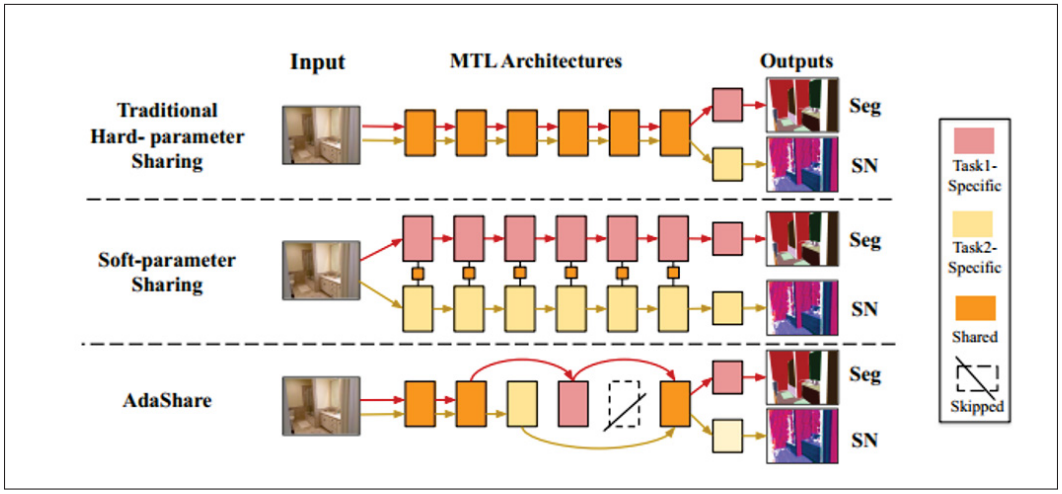


Figure 1.30 Hard and soft parameters sharing for MTL (semantic segmentation (Seg) and surface normal prediction (SN)). Bottom shows Adashare that learns the sharing pattern
Taken from Sun *et al.* (2020a)

Majority of NAS work focus on the single task learning with limited wok on MTL. Liang, Meyerson & Miikkulainen (2018) treats MTL as a routing problem and uses EA to determine task-specific routing. AdaShare (Sun *et al.*, 2020a) proposed to learn the pattern of parameter sharing (which layers share which parameters) across multiple tasks (Figure 1.30). They introduced a differentiable approach to jointly learn this pattern alongside model weights. ILASH (Rahman, Rizvee, Shomaji & Chakraborty, 2024) proposes a hybrid layer sharing in which for a new task, specialized layers are added to a branching point on the existing network. They propose a NAS method to efficiently learn the branching points.

CHAPTER 2

BALANCED MIXTURE OF SUPERNETS FOR LEARNING THE CNN POLLING ARCHITECTURE

Downsampling layers, including pooling and strided convolutions, are crucial components of the convolutional neural network architecture that determine both the granularity/scale of image feature analysis as well as the receptive field size of a given layer. To fully understand this problem, in this chapter we analyze the performance of models independently trained with each pooling configurations on CIFAR10, using a ResNet20 network to build a small "Pooling" benchmark. We show that the position of the downsampling layers can highly influence the performance of a network and predefined downsampling configurations are not optimal.

Optimizing downsampling configurations can be cast as a NAS problem. However, we show that common one-shot NAS based on a single supernet is not suitable for this problem. We demonstrate that this is because a supernet trained for finding the optimal pooling configuration fully shares its parameters among all pooling configurations. This *full* weight sharing exacerbates supernet training difficulty compared to partial weight sharing present in many search spaces.

Therefore, we propose a balanced mixture of supernets that automatically associates pooling configurations to different weight models and helps to reduce the weight-sharing and inter-influence of pooling configurations on the supernet parameters. We evaluate our proposed approach on CIFAR10, CIFAR100, Food101, and ImageNet with variations of Resnet architectures. We show that in all cases, our model outperforms other approaches and finds a similar or improved pooling configuration to the default.

2.1 Introduction

In this section, we first discuss the crucial role of downsampling in a CNN and how it is closely related to effective filter size. We then frame finding the optimal downsampling configuration as a NAS problem.

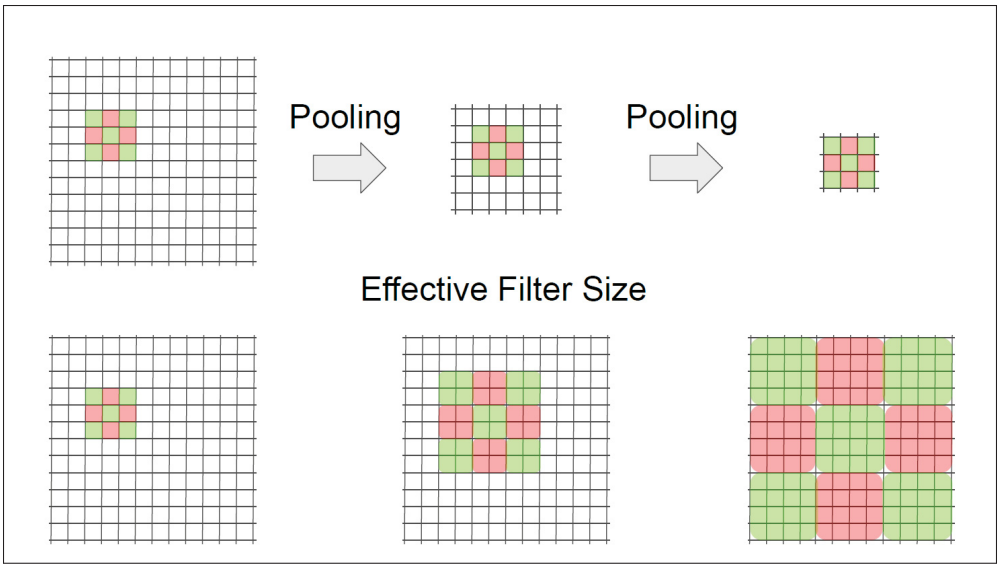


Figure 2.1 Effective Filter Size. CNNs typically interleave convolution layers using fixed-size filters (e.g. 3×3) with pooling layers (upper row). The filter size effectively increases with respect to the original image

2.1.1 Downsampling in CNN

Downsampling layers in CNNs are crucial, as they provide robustness to shift and scale variations (Zhao, Wang, Zhang *et al.*, 2017), reduce the computational cost of models (Jin, Tanno, Mertzanidou, Panagiotaki & Alexander, 2021; Riad, Teboul, Grangier & Zeghidour, 2022), and control the access of subsequent convolution kernels to spatial information, determining their receptive field (Le & Borji, 2017; Luo *et al.*, 2016). Spatial resolution is related to the receptive field, which determines the aggregation of local features and affects the performance of the CNN (Jang, Chu, Kim & Han, 2022; Richter & Pal, 2022). The receptive field in turn is controlled indirectly by the hyperparameters of the network such as depth, filter sizes, and downsampling/pooling layers. Figure 2.1 shows how the downsampling essentially increases the effective filter sizes, even if the kernel size remains the same. The spatial density of the content in a dataset highly affects the optimal receptive field and therefore the optimal spatial pooling configuration.

For instance, for good recognition on textures, smaller and more detailed local patterns are more important (Jang *et al.*, 2022), while for shapes, considering larger regions of the image should provide a better representation (Luo *et al.*, 2016). Thus, being able to select how to downsample the image representation in CNNs can help to better adapt the representation to the specific characteristics of a given dataset and help to better understand the way that CNNs find meaningful patterns in images, and therefore determine what the relevant features for a given task are (Riad *et al.*, 2022).

In CNN design, feature map downsampling is commonly performed by applying a strided convolution (Howard *et al.*, 2017), a convolution followed by a pooling (Simonyan & Zisserman, 2014; Szegedy *et al.*, 2015), or a combination of the two (He *et al.*, 2016a). As mentioned earlier, for a downstream task such as classification, the position of the downsampling in a network architecture is pre-defined and based on the assumption that the receptive field should increase over layers until covering all or most of the image (Richter & Pal, 2022). While this assumption can be removed by the use of self-attention (Chen *et al.*, 2021g), its usage seems still very important for a good trade-off of computation and accuracy (Liu *et al.*, 2021).

In this chapter, we show that there is no guarantee that commonly used pooling configurations are optimal. A possible solution is to learn the best pooling configuration for the dataset at hand. However, pooling configurations are discrete parameters and the number of candidate architectures grows exponentially with depth, making bruteforcefully searching for the best pooling configuration for modern CNNs computationally infeasible.

Previous works that attempt to find optimal feature map sizes in a predefined architecture avoid the discrete nature of sub-sampling layers by relaxing the problem by learning resizing modules (Liu *et al.*, 2020; Riad *et al.*, 2022; Jang *et al.*, 2022), or indirectly do so by learning continuous filter sizes (Romero *et al.*, 2021; Pintea, Tömen, Goes, Loog & van Gemert, 2021) at the same time as training the CNN. Other works, such as DiffStride (Riad *et al.*, 2022), cast learning fractional strides as learning cropping size in the frequency domain, the pooling is performed in the spectral domain resulting in higher cost and involving complex value operations.

2.1.2 Downsampling as a NAS Problem

Differently than previous work, we cast the problem of finding the optimal scales of analyzing the CNN features as a NAS problem. As discussed in section 1.3.4.1, a popular research direction for solving NAS problem is to utilize differentiable methods (DARTS) by first relaxing the optimization problem into an equivalent, but differentiable one and then finding the optimal hyperparameters through bi-level optimization (Liu *et al.*, 2018b). However, differentiable models can be computationally and memory-wise demanding, because they evaluate all model configurations at each training iteration. Additionally, they do not always provide optimal solutions as the bi-level optimization is heavily approximated (Xue *et al.*, 2021) and it is difficult to impose constraints on configurations (as some configurations might not be feasible). Alternatively, sample-based (SPOS) approaches can be adapted to solve the memory issue, however the noise can easily mislead the search.

In the following sections, we tested both differentiable and sample-based approaches, showing that both failed to provide good results for finding the optimal pooling configuration of a network. We hypothesize that the underlying reason is two-fold: 1) inappropriate search space design, 2) strong (full) weight sharing in supernet. We show that defining the search space naively and by treating each resolution similar to independent operations does not necessarily return better results than fully sharing weights among all resolutions. This is despite a lower degree of weight sharing. In fact, this search space design results in greedily reducing the weight sharing, i.e., all configurations with the same resolution at a layer share the same weights, regardless of the path as a whole. Therefore, even though complete weight sharing poses a problem, its reduction should be performed in a more appropriate and nuanced way.

To investigate this problem, we perform extensive experiments on CIFAR10 dataset to find the optimal pooling configuration on ResNet20 architecture. As the resolution of CIFAR10 images is low, ResNet20 uses only 2 pooling layers, which amounts (after reasonable constraints discussed in section 2.2.1) to 36 configurations. Thus, we have independently trained all configurations to convergence and consider the obtained accuracy as our ground truth performance (section

2.3.1). A revealing result is that, even with only 36 possible configurations and the extreme use of CIFAR10 for image classification, the standard pooling strategy is not optimal, and there is a gap of more than one point.

In order to find the optimal pooling configuration, we propose a new model based on single-path supernet sampling that reduces the problem of weight sharing by using multiple balanced supernets. As the standard practice, the sampling distribution of the pooling configurations is kept uniform as in (Guo *et al.*, 2020b), to avoid introducing bias. However, to avoid interference among the different pooling configurations, we train multiple models (supernets) at the same time. Each configuration favors sampling the model that leads to higher accuracy with it, while making sure that all models on average, receive equal amounts of training, so that they are balanced. This training strategy allows each model to specialize to different pooling configurations.

Our main contributions are summarized as follows:

- We present the task of finding the optimal CNN downsampling or pooling layers as a NAS problem, and perform extensive experiments to evaluate search space design and NAS methods on the CIFAR10 classification task with ResNet20 architecture. We show that designing the search space for this problem requires more insight and a naive design can lead to finding sub-optimal pooling configurations.
- We show that, while optimal pooling configurations can improve upon the performance of standard configurations on the widely used CIFAR10 dataset, they are not identified by common NAS methods. We argue that this is due to weight sharing in the supernet and, more specifically, to the full weight sharing of our problem.
- We propose a balanced mixture of supernets that reduces the weight sharing problem by learning the correct association between each pooling configuration and one of the weight models.
- We validate our approach on several datasets and CNN configurations and show that by only learning the optimal pooling configuration with our method, we can improve the classification performance of ResNet architectures without altering any other hyperparameters.

2.2 Method

In this section, we first define the search space used to find the optimal pooling configurations (section 2.2.1). We then propose a balanced mixture of supernets (section 2.2.2) to tackle the full weight sharing problem.

2.2.1 Search Space

In this section, we focus on defining a search space and constraints for finding the optimal spatial resolution of the feature maps in a CNN, which are controlled by downsampling operations. First, we consider a general search space that contains r resolutions per layer. The downsampling is performed by applying the most commonly used downsampling operations, such as max pooling, reducing feature map size by a factor of two. Similar to typical layer-wise operations, we can assign unique convolutional operations to each resolution at each layer. Considering L layers, this will result in a search space of size r^L.

We then apply several constraints on the search space as follows:

- We note that for classification, it is well known that the resolution of the feature map is reduced across layers. Therefore, paths in the search space that contain upsampling operations are not appropriate for this task and are excluded.
- We exclude from the search space the first pooling layer as it corresponds to a manipulation of the input data.
- As downsampling operations reduce the feature map size, the boundary of this search space is determined by the input size and the minimum feature map size expected before the classification layer. Therefore, we consider the same number of downsampling operations as a default network (i.e., the predefined network configuration).
- We exclude the application of more than one downsampling operation in each layer.

Figure 2.2 shows an architecture in our search space. With these restrictions. With p pooling operations available, the search space size is the combination $\binom{L-1}{p}$, exponentially growing with the depth of the network. With $p + 1$ resolutions present at the search space, each architecture

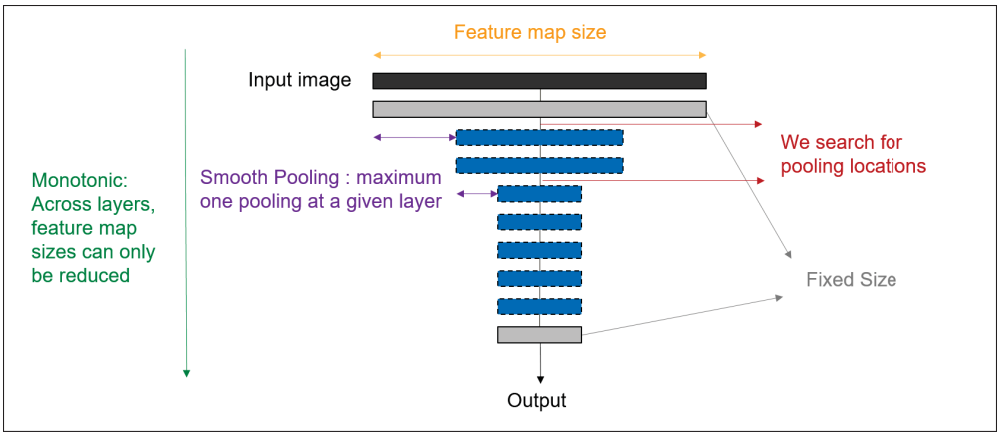


Figure 2.2 Illustration of pooling search space with various constraints

in this search space can be uniquely identified by the number of blocks in each resolution as $\alpha = [n_0, n_1, ...n_{p+1}]$, where n_i is the number of blocks in resolution i and $\sum_i n_i = L$. For example, the architecture shown in the figure can be identified as [1,2,6]. The search space details for each experiment in this chapter are detailed in table 2.7.

For simplicity, we use a pre-defined number of channels for all architectures, ensuring the same number of parameters in all architectures. We choose ResNet (He *et al.*, 2016a) as the building block of our search space, as it is one of the most widely used and well-studied architectures and ensures the inclusion of skip connections in our search space. Each basic block in Resnet contains two convolutional layers and the skip connection. We consider one block as the basic unit of our search space instead of considering a single convolutional layer.

2.2.2 Balanced Mixture of supernets

In order to find the optimal pooling configuration, we first sample all configurations during training and then evaluate the best ones at evaluation time. We follow SPOS strategy (Guo *et al.*, 2020b), by sampling a pooling configuration $c \in C$ (from the search space described above) with uniform probability at each iteration. For a mini-batch of training samples and the corresponding annotations $(x, y) \in \mathcal{X}_{tr}$, we update the network weights w by minimizing the

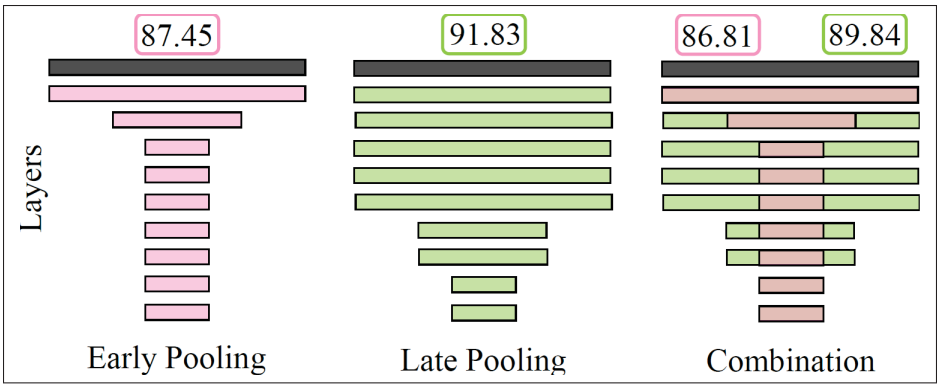


Figure 2.3 Interference of representations for ResNet20 on CIFAR10. *Early pooling* and *late pooling* produce different feature representations on their layers, which lead to different performance (on top). When training a model by sampling either one or the other pooling configuration (*combination*), the two representations interfere, which leads to lower performance of both models

following loss:

$$\sum_{(x,y)\sim\mathcal{X}_{tr},c\sim C} \mathcal{L}(f_c(x,w),y), \tag{2.1}$$

where f_c is the output of the network for a given pooling configuration c and $\mathcal{L}$ is a classification loss such as cross-entropy. The choice of uniform sampling (Xie *et al.*, 2018; Guo *et al.*, 2020b) is hyperparameter free and ensures more fairness in training among architectures. As the configurations are chosen uniformly, this training does not favor any specific configuration and provides a meaningful estimation of the performance of each configuration.

At the end of training, the network f is evaluated on a validation set $\mathcal{X}_{val}$ for all configurations C, and top-k configurations with higher accuracy are selected as best configurations. Previous work has shown that this approach works when used to select network parts that do not share weights; however, in our setting, as all configurations share the same parameters, they produce very high interference, and the supernet is no longer a good proxy to find the best performing configurations, which is the aim of this approach. This is illustrated in Figure 2.3, where jointly training two pooling configurations produces worse results than training either one or the other independently.

In fact, the features and structures seen by the convolutional filters when working with different pooling configurations are drastically different, and learning them together hinders the performance. For this reason, to reduce the weight sharing and to avoid the interference of different configurations on the same model, we propose to use a mixture of models (Figure 2.4).

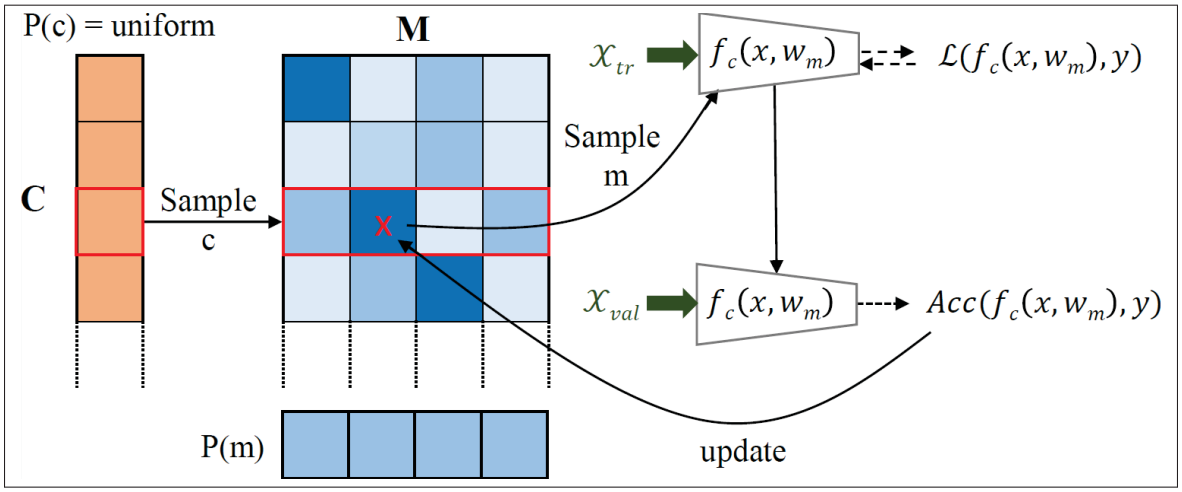


Figure 2.4 Balanced Mixture of supernets. At each training iteration, we uniformly sample a pooling configuration c, then a model with a probability proportional to $p(m|c)$. The model weights w_m are updated on a mini-batch of training data from model accuracy Acc on validation data. A moving average of the accuracy is used to update $p(c, m)$ such that $p(m)$ remains a uniform, balanced mixture of models, ensuring that each model is trained for the same number of iterations

Instead of using a single set of weights or supernet, we propose to use M independent supernet models or weight sets w_m associated with a network f_c. In this way, each set of weights may specialize to represent unique subsets of specific pooling configurations, leading to improved performance. After each mini-batch training iteration, we compute the moving average of the accuracy $a_{c,m}$ on a validation minibatch $\mathcal{X}_{val}$ for a given network $f_c(\cdot, w_m)$ with pooling configuration c and weight set w_m as follows:

$$a_{c,m} = \beta\, a_{c,m} + (1 - \beta)\, Acc(f_c(x, w_m), y),\ (x, y) \sim \mathcal{X}_{val},\ c \sim C,\ m \sim p(m|c) \qquad (2.2)$$

where β is a hyper-parameter controlling the smoothness of the moving average. At each iteration, the pooling configuration c is sampled uniformly, while the model m is sampled based on the conditional probabilities $p(m|c) = \frac{p(c,m)}{\sum_c p(c,m)}$.

The probability $p(c, m)$ is computed by normalizing the accuracies $a_{c,m}$ with a τ-softmax function:

$$p(c, m) = \frac{\exp(a_{c,m}/\tau)}{\sum_{j,k} \exp(a_{j,k}/\tau)}, \tag{2.3}$$

where in Equation (2.3), τ is a temperature hyperparameter of the probability distribution where $\tau \to 0$ implies a maximally concentrated distribution. These probabilities are thus proportional to the accuracy of the chosen joint configuration of pooling c and model m. We could use directly these probabilities to sample with a multinomial distribution a joint configuration (c, m) to train a mini-batch. However, this would make the model focus on some specific joint configuration/model during training, and will lead to coupling of pooling configurations and models due to unbalanced sampling. Instead, we want the training to give equal importance to each pooling configuration c while selecting the most promising model. The best pooling strategy is then selected at the end of the training, making sure that each configuration and each model has received an equal amount of training.

We thus achieve balanced supernet mixtures by imposing the constraint that the joint probability distribution $p(c, m)$ has uniform marginals, i.e., $\sum_i p(c_i) = 1/C$ and $\sum_j p(m_j) = 1/M$. We use the Iterative Proportional Fitting algorithm to achieve this, where $p(c, m)$ is alternatingly normalized along c and m dimensions until uniformity is achieved. The KL-distance is used to estimate the deviation of $p(m)$ from uniformity, and IPF terminates when the KL-distance falls below the threshold of $\delta = 0.0001$. At this point, the pooling configuration c is sampled uniformly while the model m is sampled from the conditional distribution $p(m|c)$.

Balancing allows each model to focus on different configurations while ensuring equal importance of all models during training iterations. Scalar τ is a concentration parameter and is decreased linearly over the course of training.

After training, the mixture of supernets is used to select the top-k performing configurations, by evaluating the model m with the highest $p(m|c)$ for each configuration c on the validation data. In this way, the number of evaluations required depends only on the number of configurations, even if many models are considered. The number of configurations to evaluate may still become prohibitive when using very deep models (such as ResNet50). In this case, instead of evaluating all configurations, we can use $p(c,m)$ as a proxy to select the correct model and $a_{c,m}$ to rank configurations and evaluate only the top ranking on the entire validation set, and therefore reduce the computation required to select the best model after training.

2.3 Experiments

In this section, we perform several experiments and ablations in order to evaluate the performance of our proposed approach. We first individually train and evaluate the performance of a small ResNet with 36 different pooling configurations on CIFAR10 to develop a Pooling benchmark (section 2.3.1). It also establishes that optimal pooling can improve the performance of the model.

We then compare the correlation between different configurations of a Mixture of supernets for various number of mixtures (M) with the individually trained configurations and demonstrate that more models help in obtaining a better correlation (section 2.3.2). Next, we present an ablation, considering different variants of weight sharing for DARTS and SPOS approaches (section 2.3.3). Additionally, we compare our model with other NAS and non-NAS approaches in sections 2.3.4 and 2.3.5. Finally, we evaluate our model on a higher resolution datasets (Food101, ImageNet), with a larger model (ResNet50) in section 2.3.6. In all experiments, we separate the default training set of each dataset in 50% for training and 50% for validation, used for estimating the quality of the configurations.

2.3.1 Pooling Benchmark

As shown by Riad *et al.* (2022), the pooling configuration of a CNN has a large impact on the performance of the model. To establish a benchmark, we consider all possible pooling configurations that satisfy conditions set in section 2.2.1. With 2 pooling operations and 9 available pooling locations, the search space size is a combination $\binom{9}{2} = 36$. This limited search space facilitates the exhaustive search of the entire space and allows us to find the true ranking by fully training all baseline architectures independently. In other words, unlike benchmarks such as Su *et al.* (2021a) and Chau, Dudziak, Wen, Lane & Abdelfattah (2022), we fully train architectures until convergence.

For this evaluation, we choose a lightweight ResNet configuration (He *et al.*, 2016a) and exhaustively train each architecture 3 times with different seeds and report the average results. The complete results on the entire space are included in table 2.1. The standard pooling configuration is configuration [4,3,3], which has the first pooling layer after 4 ResNet blocks and the second after another 3 blocks, and its classification accuracy is 90.52% ± 0.6. In contrast, the best configuration is [6,1,2], with an accuracy of 92.01% ± 0.12. This shows that even for one of the most common datasets, the default pooling structure is not optimal, and therefore, it is reasonable to propose models that can optimize the CNN pooling configuration. We also hope that this benchmark will further motivate research in the field to not overlook the importance of an optimal pooling configuration by focusing on micro search spaces. While this benchmark is relatively small, we show in the next experiments that it is quite challenging search space and most of the commonly used NAS methods fail to find a good pooling configuration.

2.3.2 Balanced Mixture of supernets

We evaluate our proposed balanced mixture of supernets on our benchmark with different numbers of models $M = \{1, 2, 4, 8\}$. The case of $M = 1$ is equivalent to SPOS method with uniform sampling as in Guo *et al.* (2020b) with full weight sharing among architectures. In figure 2.5, we show the correlation of the performance obtained by our supernet trained with

Table 2.1 Pooling benchmark. CIFAR10 accuracies and standard deviations for all configurations with ResNet20 Backbone. Architectures are displayed in terms of the number of blocks associated with feature map sizes of [32, 16, 8]. Architecture 24 is the original ResNet20 architecture pooling configuration

no.	Architecture	Accuracy	no.	Architecture	Accuracy
1	[1 , 1 , 8]	87.45 ± 0.06	19	[3 , 4 , 3]	90.92 ± 0.10
2	[1 , 2 , 7]	87.69 ± 0.08	20	[3 , 5 , 2]	90.88 ± 0.08
3	[1 , 3 , 6]	87.89 ± 0.17	21	[3 , 6 , 1]	90.14 ± 0.16
4	[1 , 4 , 5]	88.60 ± 0.15	22	[4 , 1 , 5]	89.68 ± 0.14
5	[1 , 5 , 4]	89.38 ± 0.07	23	[4 , 2 , 4]	90.34 ± 0.13
6	[1 , 6 , 3]	90.13 ± 0.14	24	[4 , 3 , 3]	90.52 ± 0.15
7	[1 , 7 , 2]	90.16 ± 0.10	25	[4 , 4 , 2]	90.85 ± 0.12
8	[1 , 8 , 1]	89.41 ± 0.15	26	[4 , 5 , 1]	89.71 ± 0.16
9	[2 , 1 , 7]	88.78 ± 0.10	27	[5 , 1 , 4]	91.05 ± 0.15
10	[2 , 2 , 6]	89.03 ± 0.12	28	[5 , 2 , 3]	90.96 ± 0.15
11	[2 , 3 , 5]	90.42 ± 0.10	29	[5 , 3 , 2]	91.55 ± 0.09
12	[2 , 4 , 4]	90.57 ± 0.11	30	[5 , 4 , 1]	89.84 ± 0.08
13	[2 , 5 , 3]	90.89 ± 0.07	31	[6 , 1 , 3]	91.78 ± 0.11
14	[2 , 6 , 2]	91.01 ± 0.13	32	[6 , 2 , 2]	91.83 ± 0.13
15	[2 , 7 , 1]	90.22 ± 0.18	33	[6 , 3 , 1]	90.96 ± 0.12
16	[3 , 1 , 6]	89.10 ± 0.06	34	[7 , 1 , 2]	92.01 ± 0.18
17	[3 , 2 , 5]	89.70 ± 0.10	35	[7 , 2 , 1]	90.47 ± 0.11
18	[3 , 3 , 4]	90.61 ± 0.17	36	[8 , 1 , 1]	89.99 ± 0.12

different number of mixture models, with true accuracies of given pooling configuration models trained independently on the test set, and we calculate Kendall's tau rank correlation coefficient.

As expected, using multiple mixtures shows an overall stronger correlation compared to SPOS uniform sampling ($M = 1$). This model has poor correlation with ground-truth and is unable to find optimal configurations even in our small search space. Critically, the correlation for higher performing architectures is very poor. The improvement with balanced mixture is more prominent with these higher ranking models, resulting in finding better final configurations. For this experiment, for $M > 4$ mixture models the gain in performance seems to saturate, providing no substantial benefit.

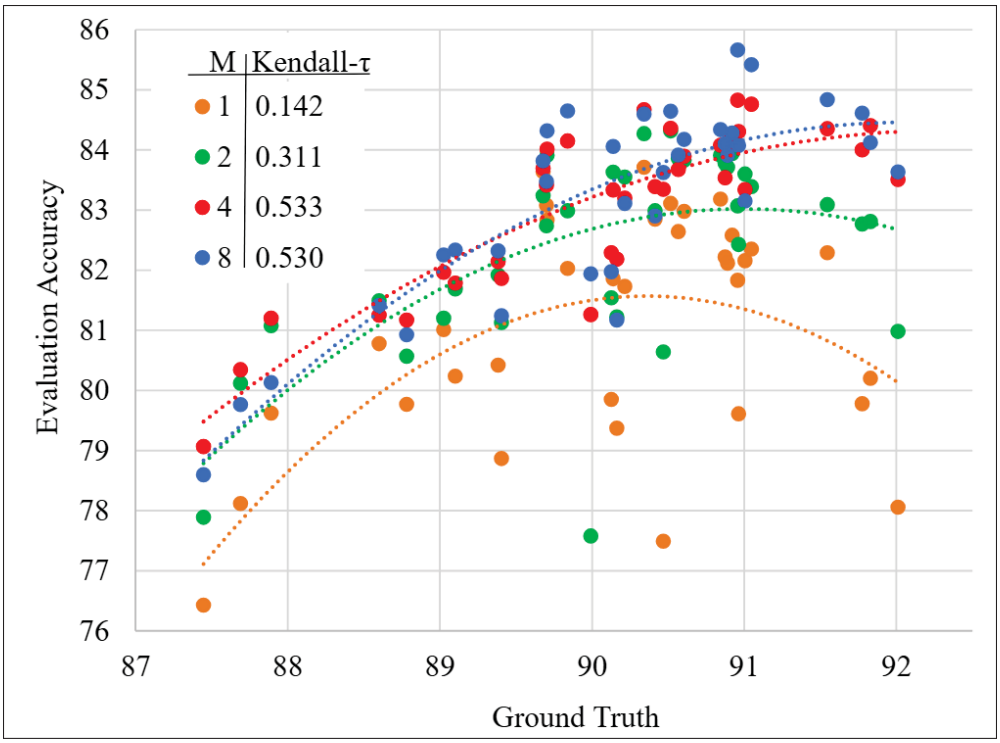


Figure 2.5 Evaluation accuracy vs. ground truth accuracy for CIFAR10 test dataset for different number of mixture models (M). Each point represents the performance of a given configuration with a model trained independently (Ground truth) and the same configuration evaluated with the supernet (Evaluation Accuracy) with different numbers of mixtures M (colors). Rank correlation measured by Kendall's tau increases with the number of models

2.3.3 Relaxing the Full Weight Sharing

We argued that one reason that makes the learning of optimal pooling difficult is the fact that the model weights are fully shared, i.e. the same weights are used for all feature scales/resolutions. In this section, we consider the case of relaxing the weight sharing and using independent weights for each resolution. For this experiment, we evaluate SPOS and DARTS, in case of using the same parameters for each feature map resolution (*Fully shared*) or different (*Not shared*).

For SPOS, we consider two different search spaces. The first uses only 36 paths in the benchmark. However, we note that by using only those 36 paths and different filters per resolution would

Table 2.2 CIFAR10 results for different search methods, number of model weights and paths. For DARTS, we consider a model with shared weights for different feature map resolutions (*Fully Shared*) and not shared (*Not Shared*) with different weights per resolution, with either the same or different initialization. In all cases, accuracy is lower than the *Default*. For SPOS, we test *Fully Shared* weights and *Not Shared* with different number of paths. Results are comparable to the default setting. Only our *Balanced Mixtures* of supernets clearly outperform the default

NAS Method	**Mixtures (M)**	**Paths**	**Architecture**	**Accuracy**
Default	-	-	[4,3,3]	90.52 ± 0.1
DARTS				
Fully shared	1	19,638	Figure 2.7 (a)	89.23 ± 0.13
Not shared - same init.	4	19,638	Figure 2.7 (b)	89.85 ± 0.18
Not shared - rnd. init.	4	19,638	Figure 2.7 (b)	90.03 ± 0.21
SPOS				
Fully shared	1	36	[3,3,4]	90.61 ± 0.17
Not shared	4	36	[4,2,4]	90.34 ± 0.12
Not shared	4	19,683	[4,2,4]	90.34 ± 0.12
Balanced Mixtures (Ours)	4	36	[5,3,2]	91.55 ± 0.08

induce some filters to be trained much more than others, which would bias the selection of the optimal filters toward those. To avoid that, we also consider a case in which all 19,683 possible configurations are used. In this case, bias is reduced, but the training takes longer and has more noise.

For DARTS, in the case in which each resolution has different parameters (*Not shared*) we consider two variants: the case of initializing filters with the same initialization for all resolutions (*same init.*) or different random initialization (*rnd init.*). As table 2.2 shows, only our adaptive association of pooling configurations and model parameters (*Balanced Mixtures*) manages to obtain better results than the *Default* pooling configuration.

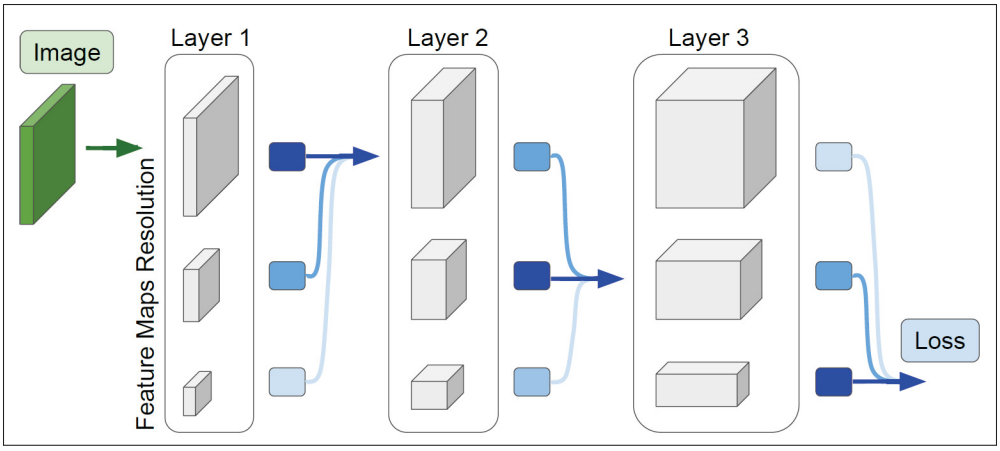


Figure 2.6 Multi-scale resolution layers. We learn the optimal feature map resolution by selecting the importance of each resolution layer

2.3.4 Comparison with NAS-based methods

We compare our method with several variants of commonly used NAS methods: MCTS (Su *et al.*, 2021a), DARTS (Liu *et al.*, 2018b), GAEA (Li *et al.*, 2020d), and Boltzmann Softmax Exploration (BSE) (Asadi & Littman, 2017; Cesa-Bianchi, Gentile, Lugosi & Neu, 2017). For MCTS we used the same tree structure as Su *et al.* (2021a) with and without uniform warm-up with detailed discussion of MCTS and tree structure provided in chapter 3.

DARTS: Differentiable approaches first proposed by DARTS (Liu *et al.*, 2018b) have been commonly used in recent years for NAS problems. As one of the most efficient and reliable NAS methods, we utilize DARTS for our problem. In terms of optimization, we use the same differentiable principle for architecture search as DARTS (Liu *et al.*, 2018b). Instead of learning weights for different network branches, we learn weights for different feature map resolutions by learning the associated architecture parameter α. Since changing the position of the pooling layer in the network changes the size of a feature map and therefore invalidates all the subsequent blocks, we need to find a way to achieve this without changing the feature map resolution. Therefore, we introduce a multi-scale resolution block **M** (figure 2.6) defined as the weighted combination of convolutions **V** at different resolutions r of the same filters f_l at a given layer l:

$$
\begin{aligned}
h_{l+1}(x, y) &= \mathbf{M}(h_l(x, y)) \\
&= \sum_{r=1}^{R} \alpha_{l,r} \mathbf{U}_{2^r}(\mathbf{V}(\mathbf{S}_{2^r}(h_l(x, y)), f_l(h, w))).
\end{aligned} \tag{2.4}
$$

The resulting feature map is the sum of the feature map at each resolution multiplied by a coefficient $\alpha_{l,r}$ and rescaled to the initial feature map resolution (x, y) resolution with an upsampling operation $\mathbf{U}$.

The normalized coefficient $\alpha_{l,r}$ learns the relative strength of a certain resolution with respect to the others for a given layer l and is computed as a softmax over feature map resolutions:

$$
\alpha_{l,r} = \frac{\exp(\alpha_{l,r})}{\sum_s \exp(\alpha_{l,s})}. \tag{2.5}
$$

This approach allows us to train a model that can learn the convolutional filters, but at the same time, with a marginal increase in computation and memory can also learn the best feature resolution to use at each layer.

To make the search space as similar to our method as possible, we manually select the highest resolution for the first layer of the network. However, imposing all restrictions on the search space is more difficult. At the end of training, we select the maximum $\alpha_{l,r}$ for each layer as the final architecture to retrain. We used ADAM (Kingma & Ba, 2014) optimizer to train α and SGD with cosine learning rate for CNN weights. Furthermore, e used DARTS with GAEA(Li *et al.*, 2020d), which both fail on this task as it finds sub-optimal architectures. As seen in figure 2.7, the architectures found by this approach are atypical for classification task as they utilize upsampling in several later layers, resulting in lower accuracy of final architecture.

Boltzmann Softmax Exploration (BSE): BSE is one of the simplest reinforcement learning exploration strategies. For sampling an architecture c we use:

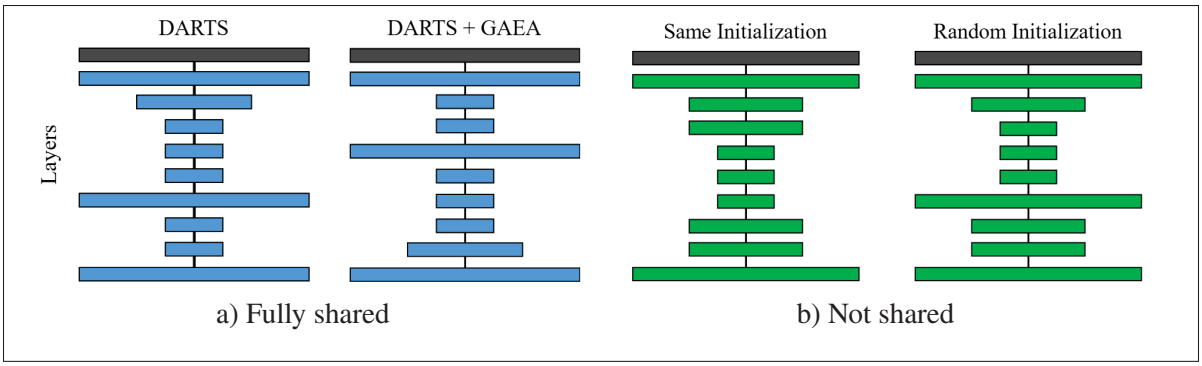


Figure 2.7 Final configurations found by DARTS (Liu *et al.*, 2018b) and its variant DARTS + GAEA (Li *et al.*, 2020d). The first layer in gray is fixed at the maximum input resolution. (a) shared weights per layer. (b) different weights per feature map resolution for each layer, weights are initialized either randomly or with the same values

$$p(c) = Softmax(\tau\ a_c) \tag{2.6}$$

Where $p(c)$ is the probability of selecting architecture c, and a_c is the reward (here the validation accuracy) and τ is the inverse temperature, controlling exploration and exploitation. We increase τ from 1 linearly during the experiment.

With the defined search space of 36 architectures, we choose a linear schedule for the inverse temperature that balances exploration and exploitation. Furthermore, we considered another option with a uniform warm-up. However, the appropriate scheduling is difficult as the model can either continue to explore sub-optimal solutions or commit to one solution too early (Cesa-Bianchi *et al.*, 2017).

In table 2.3, we present results in terms of found architecture and accuracy of the selected configuration. Even if the number of possible configurations is limited, none of the methods manages to obtain the best pooling configuration, which is 1.5 points above the default baseline. DARTS-based methods (as they do not have constraints on the pooling configurations) yield peculiar configurations in which down-sampling is followed by up-sampling (see figure2.7), which brings a loss of information and therefore poor results. Other methods based on SPOS,

Table 2.3 CIFAR10 found architectures, accuracies, and training time for different NAS methods. Results on DARTS are relaxed selections of resolutions and therefore outside the search space we defined in our work

NAS Method	Architecture	Accuracy	GPU hours
DARTS + GAEA	Figure 2.7 (a)	89.12 ± 0.1	12
DARTS	Figure 2.7 (a)	89.23 ± 0.08	12
SBE + Unif. Smp.	[1,6,3]	90.13 ± 0.06	2.5
SPOS	[4,2,4]	90.34 ± 0.12	1.5
MCTS UCB	[4,2,4]	90.34 ± 0.12	2.5
SBE	[2,3,5]	90.42 ± 0.08	2
Default	[4,3,3]	90.52 ± 0.10	-
MCTS UCB + Unif. Smp.	[4,4,2]	90.85 ± 0.09	2.5
Balanced Mixtures (Ours)	[5,3,2]	91.55 ± 0.08	6
Best conf. (Bruteforce)	[7,1,2]	92.01 ± 0.12	98

BSE, and MCTS with different variants obtain results that are comparable and close to the default setting. Our method with $M = 4$ models is the only one that approaches the optimal performance, with an accuracy of 91.55%.

2.3.5 Comparison with Other Methods

We compare our Balanced Mixture of supernets with non-NAS based approaches that aim to improve performance by learning the scale of the feature representation. In contrast to the previous experiments, here we present results provided directly by other papers.

In this case, we noticed that the final performance is highly affected by the performance of the baseline model, which can vary depending on small and difficult to control details. Thus, in order to make the comparison fairer, results of the method (*Improved*) are presented with respect to the corresponding *Baseline*, so that we can consider not only the absolute performance but also the relative *Gap* with respect to the baseline.

To compare with DiffStride (Riad *et al.*, 2022) we use ResNet18 architecture, where the original structure consists of 8 blocks with 4 resolutions as [2,2,2,2], resulting in the search space of 56

Table 2.4 Accuracy comparison between different non-NAS methods that find optimal feature map scale and our method on CIFAR10 and CIFAR100 for ResNet18 and ResNet50 backbone

Method	Backbone	Baseline	Improved	Gap
CIFAR10				
DiffStride (Riad *et al.*, 2022)	ResNet18	91.4 ± 0.2	92.4 ± 0.1	1.0
Balanced Mixtures (Ours)	ResNet18	90.45 ± 0.21	91.51 ± 0.09	1.06
CIFAR100				
DynOPool (Jang *et al.*, 2022)	ResNet50	78.50	80.60	2.1
ShapeAdaptor (Liu *et al.*, 2020)	ResNet50	78.50	80.29	1.8
Balanced Mixtures (Ours)	ResNet50	77.57 ± 0.18	79.61 ± 0.21	2.04

configurations. To compare with DynOPool (Jang *et al.*, 2022) and ShapeAdaptor (Liu *et al.*, 2020) we used ResNet50. Since ResNet50 for CIFAR does not include initial downsampling layers, the search space consists of 560 configurations with the default configuration of [4,4,6,3]. It should be noted that the search space of these methods is not identical to ours.

In table 2.4 we compare our results with DiffStride (Riad *et al.*, 2022) on CIFAR10 with ResNet18. We also compare with DynOPool (Jang *et al.*, 2022) and ShapeAdaptor (Liu *et al.*, 2020) on CIFAR100 with ResNet50. In all experiments, our approach performs comparably to other methods that explicitly change and improve the pooling layers.

2.3.6 Larger Dataset and Models

To evaluate our method on new domains, we use our method for classification on Food101 and ImageNet, which contain more images than CIFAR and at higher resolutions.

Food101 Experiments: We adapt a deep ResNet network, ResNet50, and fix the first layer and initial downsampling in the architecture. By using a deeper network, the search space size is increased to a combination $\binom{15}{3} = 455$ architectures. We conduct our experiments on an input image resolution of 256. Food101 is a challenging fine-grained object classification dataset that consists of 101 food categories with 75,750/25,250 training/test split. We show

Table 2.5 Resnet50 on Food101 dataset. We report the best architectures, their accuracy after retraining for different M. Increasing M leads to an architecture with better accuracy

Models	Best Architecture	Accuracy
Default	[3,4,6,3]	84.00 ± 0.10
M = 1	[6,4,5,1]	84.24 ± 0.09
M = 2	[4,5,6,1]	84.34 ± 0.18
M = 4	[8,3,3,2]	84.35 ± 0.14
M = 8	[9,4,2,1]	84.73 ± 0.09

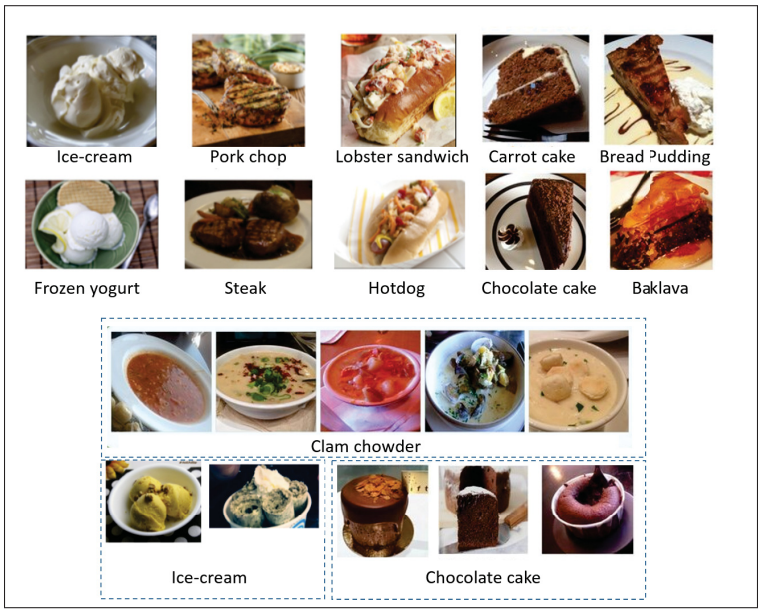


Figure 2.8 Food101 shows high inter-class similarity (top) and high intra-class diversity (bottom)

the results in Table 2.5. The results clearly show increased improvement by increasing the number of models. Food101 has high intra-class variance (see figure 2.8), which does not show distinguishing spatial layout, and the classification would need to rely on colors, textures, and local information to distinguish them (Bossard, Guillaumin & Van Gool, 2014). Therefore, unsurprisingly, identified architectures show a tendency towards adopting high resolution feature maps in early layers.

Table 2.6 Resnet18 on ImageNet. We report the best architectures, their accuracy after retraining for different number of Balanced Mixtures (M) of our supernets. As the ranking is noisy, we retrained the best 3 architectures based on our ranking and report top-1 and top-3 accuracies

Models	**Top-1 Architecture**	**Top-1 Accuracy**	**Top-3 Best**	**Top-3 Average**
Default	[2,2,2,2]	68.32 ± 0.24	-	-
M = 1	[1,3,1,3]	62.21 ± 0.26	65.91 ± 0.21	63.51 ± 1.56
M = 2	[3,1,1,3]	62.56 ± 0.18	68.32 ± 0.24	64.70 ± 2.57
M = 4	[5,1,1,1]	65.88 ± 0.24	66.12 ± 0.18	64.73 ± 1.78
M = 8	[2,3,2,1]	64.81 ± 0.11	66.12 ± 0.23	64.15 ± 2.98

ImageNet Experiments: We used ResNet18 architecture as a backbone for ImageNet dataset. We trained the top-1 and top-3 best architectures found by our method with $M = \{1, 2, 4, 8\}$ for 100 epochs and report the mean and std of 3 runs in table 2.6. We note that the baseline is the superior architecture among the trained architectures. Nevertheless, using our method with $M = 2$, we were able to recover the default architecture. We hypothesize that the reason for the default architecture having the best performance is that the current ResNet architecture is highly optimized for ImageNet dataset.

2.4 Experimental Setup and Details

Datasets and Hyperparameters: All datasets in our experiments were split 50/50 for NAS training and validation. Unless otherwise specified, all experiments were run 3 times with random seeds, and average and standard deviations are reported. For Resnet18 and Resnet50 tests we use mixed-precision operations and FFCV (Leclerc *et al.*, 2023) library to increase training efficiency. We tuned the hyperparameters either by grid search for our experiments or, when compared with other work, used similar hyperparameters. We used SGD with learning rate scheduling and weight decay for all our experiments. For ResNet20 we used a learning rate of 0.1 with cosine annealing, weight decay 1e-3, and batch size 256. For DARTS experiments, we used Adam for architecture parameters with learning rate 1e-2.

Table 2.7 Search space design for experiments conducted in this chapter. For larger input images (ImageNet and Food101), we keep the first layer (convolution and maxpooling) predefined for computational efficiency and only search the pooling locations among layers after the predefined stem layers. The search space size is the combination $\binom{Layers-1}{pooling}$

Backbone	Dataset	Feature Map Sizes	Searched Layers	No. Pooling	Search Space Size
ResNet20	CIFAR10	[32,16,8]	10	2	36
ResNet18	CIFAR10	[32,16,8,4]	9	3	56
ResNet50	CIFAR100	[32,16,8,4]	17	3	560
ResNet50	Food101	[56,28,14,7]	16	3	455
ResNet18	ImageNet	[56,28,14,7]	8	3	35

For our Balance Mixture of supernets, the number of training epochs is set proportional to number of models to ensure sufficient training. Furthermore, we initialize $\tau = 1$ and decrease it linearly during the training with a minimum value of $1/(100M)$ with M the number of models. For experiments on CIFAR10 and CIFAR100 with ResNet18, we train for 400 epochs, with a learning rate of 0.1 and reduced it by a factor of 0.1 on epochs [200,300] and weight decay of 5e-3. For ResNet50 experiments on CIFAR100 we trained for 250 epochs and changed the scheduling to reduce by factor 0.2 at epochs [60,120,160]. For Food101, we used a learning rate of 0.1, cosine annealing, and batch size 256.

Search Space Details: Summary of search space design for the experiments is provided in Table 2.7. ResNet20 (He *et al.*, 2016a) architecture consists of one convolutional layer followed by 9 ResNet layers. The original structure consists of [32, 16, 8] feature map sizes and [16, 32, 64] number of filters respectively, with [3,3,3] blocks per resolution. In our implementation, fully connected layer is removed, and strided convolutions are replaced by maxpooling.

2.5 Conclusion

In this chapter, we presented the problem of learning the optimal scale for CNN feature maps by learning pooling/downsampling configurations. We constructed a small Pooling benchmark that enables us to investigate the effect of pooling configurations on performance. We showed that

current NAS methods (single-path uniform sampling (SPOS), differentiable methods (DARTS), and MCTS) are insufficient for this problem. We established empirically the importance of appropriate search space design by an extensive evaluation on CIFAR10 and introduced "Balanced Mixture" of supernets to alleviate the weight-sharing issue of poor ranking correlations for this problem. Finally, we compared our method with several non NAS-based approaches and evaluated it on larger and more challenging datasets (Food101 and ImageNet) with larger models and search spaces.

CHAPTER 3

NEURAL ARCHITECTURE SEARCH BY LEARNING A HIERARCHICAL SEARCH SPACE

MCTS is a powerful tool for many non-differentiable search related problems, such as adversarial games. However, the performance of such an approach highly depends on the order of the nodes that are considered at each branching of the tree. If the first branches cannot distinguish between promising and deceiving configurations for the final task, the efficiency of the search is significantly reduced. In single-objective NAS, as only the final architecture matters, the visiting order of the branching can be optimized to improve learning.

In this chapter, we study the application of MCTS to NAS for image classification task. We analyze several sampling methods and branching alternatives for MCTS and propose to learn the branching by hierarchical clustering of architectures based on their similarity. We propose to measure the similarity using the pairwise distance of output vectors of architectures. Extensive experiments on two challenging benchmarks on CIFAR10 and ImageNet show that MCTS, if provided with a good branching hierarchy, often yields better solutions more efficiently than other approaches for NAS problems.

3.1 Introduction

In this section, we briefly revisit the issues of weight sharing in NAS and the challenges of importance sampling. We then discuss the node independence assumption and how different factorizations of the search space affect the efficiency of the search. We highlight the benefits of appropriate factorization and search space design, and finally, the benefits and challenges of performing MCTS and supernet training jointly.

One-shot NAS methods based on weight sharing (Pham *et al.*, 2018) have reduced computational cost of NAS by avoiding to train individual architectures by using a single supernet containing all possible operations/architectures. This allows for efficient reuse of training iterations among compatible architectures (Cha, Kim, Lee & Yun, 2022; Pham *et al.*, 2018; Bender *et al.*, 2018).

However, for vastly different architectures, it may lead to interference, i.e. weights beneficial to one harm others, and vice-versa.

Reducing weight sharing during training can alleviate the detrimental effect of this interference. Prior work has explored multiple specialized models for different search space regions (Su *et al.*, 2021b; Zhao *et al.*, 2021a; Roshtkhari *et al.*, 2023) or importance sampling (Liu *et al.*, 2018b; Ye *et al.*, 2022; Xu *et al.*, 2019b; Wang, Li, Gong & Chandra, 2021a) where the likelihood of an architecture's performance is estimated during training. This estimation is used to identify and sample more frequently promising architectures, gradually reducing possible interference. The challenge of using importance sampling is to robustly estimate and identify superior architectures as early as possible in the training cycle. This early identification is vital for efficient resource allocation by minimizing wasting training resources on unpromising architectures. This requires fast and reliable estimation of the probability distribution in search space with minimal training iteration.

An architecture can be viewed as a graph, where nodes defining the architecture are connected by edges. These nodes present a choice of operations, which are the processes applied to the data (e.g. convolution, fully connected, etc.). For efficient probability estimation in importance sampling, a common assumption is *"node independence"*, where nodes are considered statistically independent variables. For instance, in a neural network, the operation choice for the second layer would not depend on the operation choice in the first layer. This simplifies the architecture probability estimation to a product of nodes' probabilities (see figure 3.1 (a)) and reduces the scale of the problem to learning individual node probabilities. Popular differentiable NAS methods (DARTS (Liu *et al.*, 2018b) and followup works (Ye *et al.*, 2022; Xu *et al.*, 2019b; Li *et al.*, 2020a)) rely on this assumption. However, recent work (Ma, Zhang, Xia & Tao, 2023; Zhang, Wu, Miao, Guo & Yang, 2024b) shows that overlooking the joint contribution of nodes to performance can lead to a poor node selection for the final architecture.

Removing node independence assumption requires estimating the joint probabilities of all configurations (see figure 3.1 (b)). In SPOS methods (Guo *et al.*, 2020b; Chu *et al.*, 2021c;

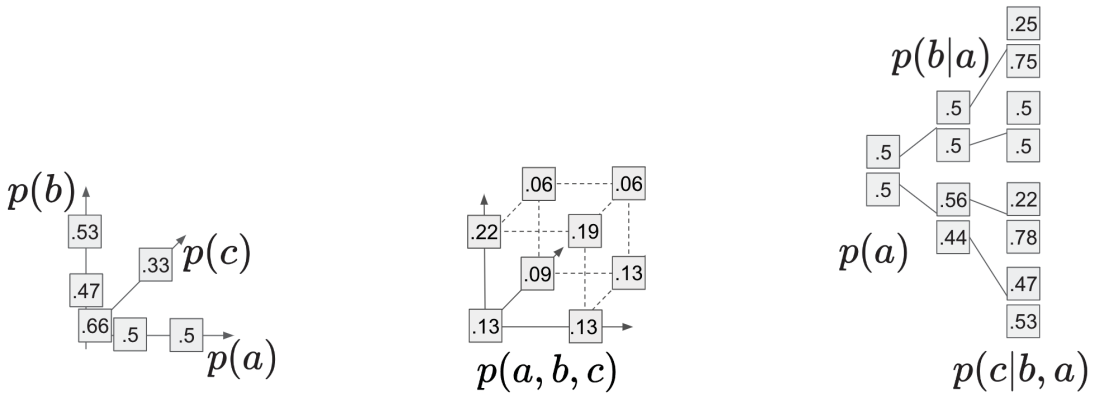


Figure 3.1 Probability factorization of 8 architectures. We show different ways to approximate the discrete probability distribution of architectures for a toy example of search space with N=3 nodes (a,b,c in the figure) each one with O=2 possible operations for a total of 2^3 architectures. (a) Assuming the nodes independent (as in DARTS (Liu *et al.*, 2018b)) allows the model to estimate only $N \times O$ probabilities. (b) Considering the joint probabilities would require to estimate O^N different probabilities (as in Boltzmann sampling). (c) The joint probability can be factorized into the product of conditional probabilities (in a hierarchy such as in MCTS). This does not reduce the probabilities to estimate, but allows a more efficient exploration of the search space

Li & Talwalkar, 2020; Stamoulis *et al.*, 2020), at each iteration one architecture from the supernet is sampled, trained, and estimated. While node independence allows updating all node probabilities at each iteration, estimating the joint probability only updates the probability of the sampled architecture. Thus, the full update is inefficient with the estimation cost proportional to the number of architectures, making it unscalable to large search spaces.

To explore more wisely, a compelling option is to factorize the joint probability into conditionals, such as a tree structure for MCTS (figure 3.1 (c)) (Świechowski *et al.*, 2023; Costa & Pedreira, 2023). While the number of probabilities to estimate remains the same as the joint probability, it offers several key advantages. A well-designed hierarchical search space enables more efficient tree traversal by reducing the unnecessary exploration of unpromising branches. Prioritizing promising branches can lead to faster convergence, improved solution quality and better scalability. However, standard predefined hierarchies do not guarantee a more efficient exploration, as

they are defined by construction and without taking into account the semantic similarity of the corresponding architectures.

While in many MCTS problems the search hierarchy is defined by the sequentiality of the problem (e.g. the moves in chess), for NAS there is no constraint in the order of exploring architectures. Prior work (Zhao *et al.*, 2021b; Wang, Xie, Li, Fonseca & Tian, 2021b; Wang, Fonseca & Tian, 2020c) leveraged this flexibility to reduce the unnecessary exploration by using the classification accuracy of the nodes to partition the search space into *"good"* and *"bad"* nodes. However, their approach utilize MCTS only as a sampler for improving search on a fixed, already learned, recognition model (i.e. the CNN weights), rather than using it during supernet training. In fact, Zhao *et al.* (2021b) use MCTS for searching the best performing model on a supernet pre-trained with uniform sampling, Wang *et al.* (2021b) perform MCTS for NAS using the already trained models provided by NAS-Bench-201 (Dong & Yang, 2020), and Zhao, Liang, Lu & Cheng (2024) utilize both the benchmarks and a uniformly trained supernet.

In this work, we tackle the more challenging problem of learning the recognition model and the tree partitioning jointly. Utilizing MCTS for sampling during supernet training enables benefits of focused search on favorable regions. In following sections, show that an intelligent factorization of search space is enough to find good architectures, without the need for further regularization such as Su *et al.* (2021a) to compensate for low sampling efficiency.

We propose to factorize the search space in a unsupervised manner based on the semantic similarity of the architectures. The output vector of each architecture, sampled from this supernet, is used to calculate a pair-wise distance matrix of architectures and used for hierarchical clustering and generating tree partitions. The resulting hierarchy implicitly enforces the early nodes of tree to be semantically related, without directly factoring their performance. This approach has the advantage of not relying on architecture performances estimates from supernet which are often inaccurate (Bender *et al.*, 2018; Li *et al.*, 2020e; Zhang *et al.*, 2020c). This performance agnostic clustering enhances learning and consequently accelerates the search process.

The main contributions of our work are the following:

- We present a new understanding of classical choices of models and strategies for NAS based on the sampling approach and the estimation of the underlying probability of a given architecture. We show that overly restrictive assumptions (e.g. node independence) enables faster estimation, but converges to suboptimal solutions. In contrast, adopting more realistic approach based on conditional probabilities and a hierarchical search space can lead to better solutions.
- We propose an efficient method to sample from the search tree by learning to build a good hierarchy that avoids low-performing architectures. To build this hierarchy, we evaluate several approaches and show that the most promising is based on pairwise distances between architectures, derived from a supernet after MCTS warm-up phase with uniform sampling.
- We empirically validate our findings on two NAS benchmarks on CIFAR10 dataset and MobileNet search space for ImageNet. Our results show that the proposed approach improves over previous MCTS approaches and can discover promising architectures within a limited computational budget.

3.2 Related Work

In this section we discuss previous work related to node independence and MCTS for NAS.

3.2.1 One-shot methods

One-shot methods (Pham *et al.*, 2018; Bender *et al.*, 2018) have become very popular in NAS (Liu *et al.*, 2018b; Guo *et al.*, 2020b; Su *et al.*, 2021a) due to their efficiency and flexibility. Generally, the training of the supernet and searching for the best architecture can be decoupled (Guo *et al.*, 2020b; Wang *et al.*, 2021b) or performed simultaneously (Liu *et al.*, 2018b). In the former, the search can be performed by various methods, such as random search (Bender *et al.*, 2018; Li & Talwalkar, 2020), evolutionary algorithms (Guo *et al.*, 2020b) or MCTS (Wang *et al.*, 2021b) and the supernet is static during this phase. The latter alternates between training the supernet and updating the reward to guide the search, such as updating architecture

weights in differentiable methods (Liu *et al.*, 2018b), controller in RL (Pham *et al.*, 2018) or probability distribution in MCTS (Su *et al.*, 2021a). However, the quality of supernet as a proxy for architecture evaluation has been the subject of scrutiny in recent years, with various results in different settings (Yu *et al.*, 2019; Wang *et al.*, 2021c; Zela *et al.*, 2019; Termritthikun, Jamtsho, Ieamsaard, Muneesawang & Lee, 2021; Zhang, Wang, Qin & Yan, 2024a). A proposed solution is to explicitly reduce the weight sharing among architectures by non-hierarchical factorization of the search space (Roshtkhari *et al.*, 2023; Su *et al.*, 2021b; Zhao *et al.*, 2021a; Ly-Manson, Leonardon, El Bey, Hacene & Mauch, 2024). Nevertheless, in general, these methods are computationally more expensive as they require training additional models. Tree-based approaches can be viewed as a form of hierarchical factorization of the search space that reduces weight sharing.

3.2.2 Node independence

Early NAS methods using reinforcement learning (Zoph & Le, 2016; Pham *et al.*, 2018), or evolution (Real *et al.*, 2019; Sun *et al.*, 2020b, 2019) do not treat nodes as independent, but they were computationally expensive. More efficient and widely adopted differentiable methods (based on DARTS (Liu *et al.*, 2018b)) use back-propagation to learn both node weights (probabilities) and supernet weights. However, one of their known issues is that the learned weights for the nodes fail to accurately reflect their contribution to the ground truth performance and ranking (Wang *et al.*, 2021c; Yu *et al.*, 2019). While several studies have attempted to improve DARTS (Xu *et al.*, 2019b; Ye *et al.*, 2022; Chu *et al.*, 2020c; Chen, Xie, Wu & Tian, 2021f; Chu *et al.*, 2020b), few have directly explored the contribution of node independence assumption to this problem (Ma *et al.*, 2023; Xiao, Wang, Zhu, Zhou & Lu, 2022).

Shapley-NAS (Xiao *et al.*, 2022) highlights the underlying relationship between nodes by showing that the joint contribution of node pairs often differs from the accumulation of their separate contributions, due to potential collaboration/competition. They propose to reweigh the learned architecture weights using Shapley value. However, estimating the Shapely value can be costly as it requires training the supernet multiple times. ITNAS (Ma *et al.*, 2023) explicitly

models the relationship between nodes via a transition matrix and an attention vector that denotes the node probability translation to successor nodes. The matrix and vector are optimized in a bi-level framework alongside node probabilities. However, this method is currently limited to cell-level and extension to a more general macro search space is not straightforward. While these works try to incorporate node dependencies within differentiable NAS framework, an alternative approach is to directly learn either joint or conditional probabilities of the sampled architectures.

3.2.3 Architecture encoding

Some works have shown that architecture encoding can affect the performance of NAS (White, Neiswanger, Nolen & Savani, 2020; Ying *et al.*, 2019) and good encoding of architectures enables efficient calculation of relationships or distances among architectures. The most common encoding represents the architecture as a directed acyclic graph (DAG) and adjacency matrix along with a list of operations (Ying *et al.*, 2019; Zoph & Le, 2016). For using performance predictors, BANANAS (White *et al.*, 2021) proposes a path-based encoding instead of adjacency matrix and GATES (Ning, Zheng, Zhao, Wang & Yang, 2020) proposed a graph based encoding scheme that better mode the flow information in the network.

Encoding can also be learned by unsupervised training prior to NAS often utilizing an autoencoder (Li, Liu, Liu & Wang, 2020c; Lukasik, Friede, Zela, Hutter & Keuper, 2021; Lukasik, Jung & Keuper, 2022; Yan, Zheng, Ao, Zeng & Zhang, 2020; Zhang, Jiang, Cui, Garnett & Chen, 2019a) or a transformer (Yan, Song, Liu & Zhang, 2021). In our work, we make use and compare different ways of encoding architectures for our approach as ablation and show that measuring distances based on the network output seems to be the fundamental for good results.

3.2.4 Monte-Carlo tree search

MCTS with UCT (Auer *et al.*, 2002) has been used previously for NAS (Negrinho & Gordon, 2017; Wistuba, 2017). AlphaX (Wang *et al.*, 2019b) was a significant early work that utilized UCT (Auer *et al.*, 2002) for MCTS-NAS. They proposed the use of MCTS to balance exploration

and exploitation and increase the sample efficiency for NAS. They train a predictive model (Meta-DNN) to estimate the accuracy of architectures based on their encoding and guide MCTS. However, training a high quality Meta-DNN, while reducing architecture evaluation cost, requires sufficient data (architecture-prediction pairs) which adds computational overhead.

Among methods that factorize the search space manually, TNAS (Qian *et al.*, 2022) proposed to improve exploration by partitioning it into two tree structures (operation and architecture). They utilized a bi-level breadth-first search algorithm to navigate the search space more efficiently. However, the proposed operation tree is unbalanced (Le, Vo & Luong, 2024) and the breadth-first search process requires additional training epochs as the network deepens.

LaMOO (Zhao *et al.*, 2021b) and LaNAS (Wang *et al.*, 2021b) aim to tackle the problem of finding the best architecture and assume that the deep learning model is given either from a trained supernet (Wang *et al.*, 2021b) or using precomputed benchmarks (Zhao *et al.*, 2021b). Wang *et al.* (2021b) uses performance of architectures, with weights inherited from a pre-trained supernet, to partition search space into "good" and "bad" regions. Zhao *et al.* (2021b) aimed to find architectures close to the Pareto frontier for multi-objective NAS. They use hyper-volume to iteratively partition the search space and MCTS to account for partitioning errors. However, both these methods decouple supernet training from MCTS by relying on a fixed pre-trained supernet or benchmarks.

In our work, we instead aim at training the deep learning model and finding the corresponding optimal architecture with MCTS in the same optimization, which makes the problem more challenging.

The closest works that jointly performs the model training and architecture search with MCTS is Su *et al.* (2021a).Su *et al.* (2021a) proposes to construct a tree branched along operations. During the training, hierarchical sampling is used for node selection, updating the supernet weights and the reward. Node statistics are then used to update a relaxed UCT probability distribution. However, the tree design is manual, and a regularization method (named "node communication") is required to compensate for insufficient visits of nodes.

3.3 Training by Sampling Architectures

We use single-path supernet training, in which, given a neural model f (e.g. a CNN) for each mini-batch of training data $\mathcal{X}$ and corresponding annotations $\mathcal{Y}$, a different architecture a from the search space $\mathcal{S}$ is sampled and back-propagated with the following loss:

$$\mathcal{L}(f_a(\mathcal{X}, w), \mathcal{Y}) = \sum_{(x,y)\in(\mathcal{X},\mathcal{Y})} l(f_a(x, w), y), \tag{3.1}$$

where l is the sample loss (for instance cross-entropy) and w are the network weights. Training speed and model performance can vary significantly depending on how a is sampled. To prevent overfitting on training data in importance sampling methods, we use validation accuracy as the reward to estimate the probability distribution; and use online estimation on mini-batches to accelerate the process. In the following, we present some of the most common sampling techniques, from uniform sampling to our proposed approach.

3.3.1 Uniform Sampling

In the simplest and the original approach of SPOS (Guo *et al.*, 2020b), an architecture is sampled uniformly: $a \sim \mathcal{U}(|\mathcal{S}|)$, where $|\mathcal{S}|$ denotes the cardinality of the search space. Despite its simplicity, this sampling method is unbiased and given enough training, all architectures will have the same importance. This method also requires no additional information storage during the training, and in principle can accommodate even very large $|\mathcal{S}|$. In practice, however, the equal importance can present two possible challenges: i) With strong weight sharing (i.e. most weights are shared among many configurations), the same weight adapts to very different architectures, leading to destructive interference and therefore low performance (see (Zhang *et al.*, 2020c; Roshtkhari *et al.*, 2023)). ii) If weight sharing is minimal, architectures are almost independent and the training time would increase proportionally to $|\mathcal{S}|$. While uniform sampling may be combined with search space partitioning in a trade-off (Roshtkhari *et al.*, 2023; Su *et al.*, 2021b; Zhao *et al.*, 2021a), it requires training multiple models, demanding higher memory

consumption and computational cost. A different direction is to find ways to prioritize the sampling of the more promising architectures.

3.3.2 Importance Sampling with Independent Probabilities

A simplest way to estimate the importance of each operation is assuming each node a_i as independent. Thus, the probability of an architecture a with t operations is approximated as $p(a) = p(a_1)p(a_2)...p(a_t)$. This simplifying assumption improves sampling efficiency by reducing the number of probabilities to estimate. However, the solution quality is compromised, as it disregards the joint influence of nodes on the performance (Ma *et al.*, 2023; Zhang *et al.*, 2024b).

3.3.3 Importance Sampling with Joint Probabilities: Boltzmann Sampling

In Boltzmann sampling, architecture a is sampled from a Boltzmann distribution with probability $p(a) \propto \exp(\frac{\epsilon_a}{T})$, where ϵ_a is the estimated rewards (here accuracy) of a, and T is the temperature. Sampling is performed with an annealing temperature, starting from a high value (almost uniform), so that the initial phase of the training is unbiased, to a low value (almost categorical) so that the training focuses on high-performing architectures. While more efficient than uniform sampling, estimating ϵ_a remains time consuming, particularly for large search spaces, and it is difficult to balance exploration/exploitation trade-off in Boltzmann exploration (Cesa-Bianchi *et al.*, 2017).

3.3.4 Sampling with Conditional Probabilities: Tree Search

Instead of a flat vector of probabilities, we consider a tree of conditional probabilities: $p(a) = p(a_t|a_{(t'\leq t-1)})p(a_{t-1}|a_{(t'\leq t-2)})...p(a_1)$. Each a_t represents a level of the tree that partitions the set of possible architectures into disjoint subsets. The commonly used structure of the tree (Figure3.2 (b)) is defined by factorizing the model architectures layer by layer (Su *et al.*, 2021a), starting from the first layer to the last one. Assuming for simplicity a symmetric binary tree,

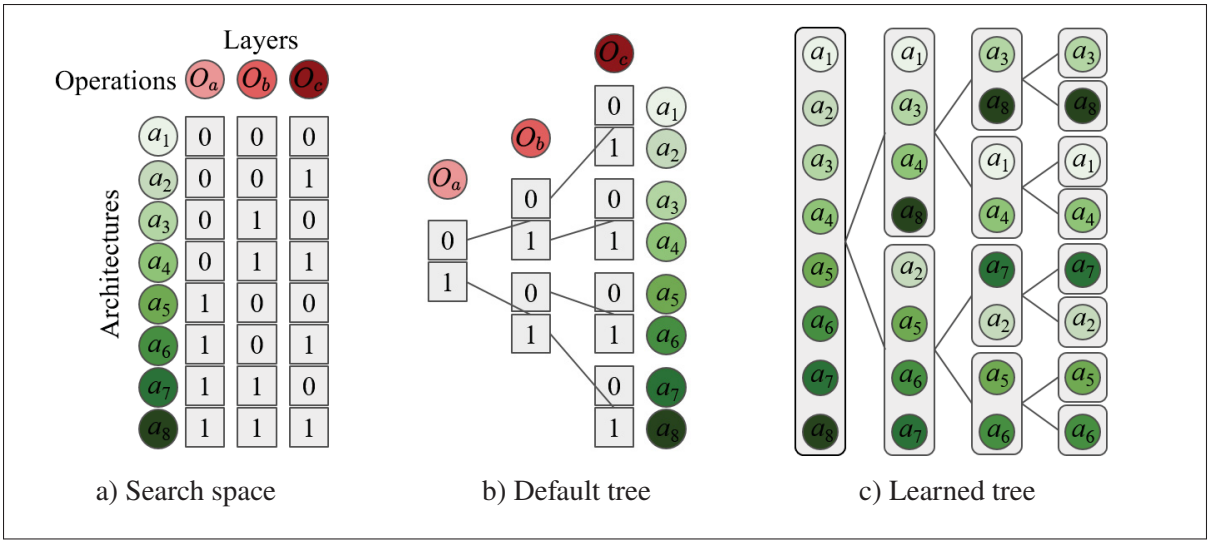


Figure 3.2 Comparison of the standard tree structure and our learned structure on a 3 binary operations search space. (a) The search space consists of architectures with 3 binary operations (o_a, o_b, o_c) which leads to 8 architectures $(a_1, a_2, ..., a_8)$. (b) The default tree structure uses the order of operations (e.g. layers) to build the tree, however this is not necessarily optimal. (c) Our learned tree structure uses a tree that is generated by an agglomerative clustering on the model outputs

the first level would split the configurations into two disjoint groups. This process of recursive partitioning continues at each subsequent level. With uniform sampling, at each iteration the nodes in level t are sampled with probability $(1/2)^t$.

Thus, probability estimates for early nodes tend to be sufficiently accurate because of high sampling rate. In contrast, for Boltzmann, the sampling rate is $1/|\mathcal{S}|$, which can be extremely small for a large search space. However, if initial nodes maintain a near-uniform probability distribution (not sufficiently discriminative), the estimation of the posterior nodes would suffer from low sampling rate. A possible solution is the regularization proposed by Su *et al.* (2021a), in which at each update of a specific node, all other equivalent nodes (nodes at the same level with the same operation) are updated similarly using an exponential moving average. This mechanism of multiple simultaneous updates allows for faster probability estimation and more efficient exploration.

While seemingly an adequate solution, this regularization comes with limitations:

- It assumes homogeneous tree structure at each level, i.e. all nodes at a given level have similar structure (identical children), which limits the approach to specific of search spaces. For instance, this approach would not be suitable for search spaces where the operations in a node are conditioned to the choice of operation at the previous node.
- Reusing the same probabilities for equivalent nodes implies treating nodes independently. In this case, the node independence assumption is enforced in a soft way by a regularization coefficient. Thus, the method attempts to find a compromise between full independence and conditional dependence, but it remains unclear if this trade-off is optimal.

3.4 Our Approach: Learning Hierarchical Search Space

Our approach tackles the low sampling rate issue in posterior nodes of MCTS from another perspective. We aim to find node ordering that enables early decisions (early nodes) to effectively separate large groups of potential solutions from bad ones. This requires imbalanced probabilities for early nodes that concentrates training on a reduced set of architectures. This increased sampling rate improves local performance estimation from supernet within this region, as interference from lower-performing architectures are reduced. Considering the example in Figure 3.1 (conditional), we could start from different nodes to obtain different hierarchy (see figure 3.3) Instead of building the tree starting from node a, one could start from node c . The imbalanced probability of branches containing c titrating form the root allows for more efficient sampling compared to a. This reordering is possible because, unlike problems in which the ordering of nodes is defined as part of the problem, NAS only cares about the final architecture and not the specific order used to reach it. Therefore, rather than employing a predefined hierarchy, we propose to learn an improved node ordering for MCTS.

In this section, we present a different approach to build a tree of architectures. As shown in Figure 3.2 (c), our tree is built based on hierarchical clustering of architectures. Each node of the tree represents a cluster of architectures, going from the root that contains all architectures in a single cluster to the leaves that each contains a single architecture. Through this approach, we

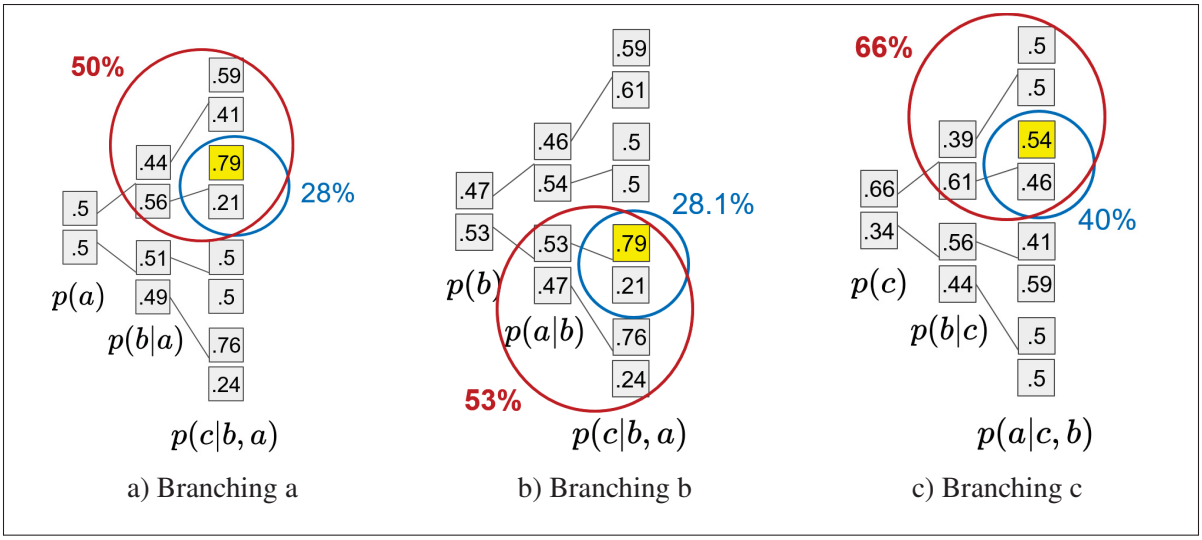


Figure 3.3 Conditional probabilities can lead to different node orders. The probability of selecting the branch containing the target node (yellow) is shown at each level of the tree

release the tree from dependence on the binary operations and allow any possible hierarchical grouping of architectures.

Hierarchy: The root node of the tree corresponds to the entire search space $\mathcal{S}$. At each node j, we partition the search space into two disjoint subsets $\mathcal{S}_j = \cup_{k\in(1,2)}\mathcal{S}_k$. We want to recursively split those into subsets, until reaching the leaf nodes that contain only one architecture. For the sake of simplicity we limited ourselves to binary splits, so that we obtain a binary tree. This splitting can be performed with different heuristics. The simplest criteria is using validation performance, with $\mathcal{S}_j = \cup_{k\in(good,bad)}\mathcal{S}_k$, such that for accuracy: $Acc(f_{a\in\mathcal{S}_{good}}(x, w)) > Acc(f_{a\in\mathcal{S}_{bad}}(x, w))$ for each node for validation samples $x \in \mathcal{X}_{val}$.

While this approach seems meaningful, it relies on the assumption that supernet can accurately differentiate $Acc(f_a(x, w))$ for various architectures. However, the supernet obtained from uniform sampling often does not adequately reflect the true ranking of architectures. Instead, in this work we propose to use an agglomerative clustering based on model distances as detailed in the next paragraph.

Clustering: A common method to identify hierarchical relationships of data is using agglomerative hierarchical clustering (Murtagh & Legendre, 2014). Given a set of leaf nodes (architectures $a \in \mathcal{S}$) and distances $d(a_i, a_j)$, the algorithm iteratively merges two pair of clusters $C_{new} = C_j \cup C_j$ that are the closest based on a linkage criteria $(C_i, C_j) = \arg\min\ L(C_k, C_{l \neq k})$. The sequence of these merges defines the hierarchical structure for tree. To measure distance, we seek a representation of architectures that adequately summarizes and captures the relevant information for the given task, independent of the quality of supernet. The output vector of the supernet $f_a(x, w)$ for a given set of validation samples $x \in \mathcal{X}_{val}$ is a suitable representation, as distances between architectures would have semantic (functional) meaning (Klabunde, Schumacher, Strohmaier & Lemmerich, 2025) in the class space (See section 3.5.2 for ablation studies on tree design). With this learned representation, a clustering algorithm can then be used to build the hierarchy based on the distances between architectures.

To ensure that output vectors provide a meaningful basis for architectural similarity (or distance), a partial training of supernet is necessary to allow for adequately capturing how different architectures process data. In our methods, we use a warm-up phase for MCTS using uniform sampling. At the end of warm-up phase, we use the uniformly trained supernet to obtain output vectors by performing a forward pass with a mini-batch of validation data and concatenating the outputs. Next, The pairwise distances of architectures are calculated and the resulting distance matrix is used for hierarchical agglomerative clustering to construct a binary tree. We argue that this method allows us to effectively cluster architectures with similar overall functionality, even if they might differ in their structure in term of their operations. For more details about the construction of the tree, see algorithm 3.1.

Search and training: We use a modified MCTS for both supernet training and architecture search. Similar to Su *et al.* (2021a) and Wang *et al.* (2021b), the tree is fully pre-expanded and thus expansion and roll-out stages of traditional MCTS are skipped. Similar to Su *et al.* (2021a), we use Boltzmann sampling for the selection stage. The Boltzmann distribution allows sampling proportional to the probability of the reward function, producing better exploration (Painter, Baioumy, Hawes & Lacerda, 2024), which is fundamental for good training of the model. For a

node in the tree a_i, we perform importance sampling with a Boltzmann sampling relative to the node:

$$p(a_i) = \frac{\exp(R(a_i)/T)}{\sum_j \exp(R(a_j)/T)}, \tag{3.2}$$

where R is our reward function and T is temperature, determining the sharpness of distribution and the normalization sum on j is taken on the sibling nodes of i. Training consists of sampling each level of the tree from the root to the leaf, followed by a gradient update of the recognition model w with the sampled architecture a on a mini-batch of training data $\mathcal{X}_{tr}$ and an update of the reward function for the explored nodes, from the leaf to the root to the tree based on the obtained accuracy of the architecture on a mini-batch of the validation set $\mathcal{X}_{val}$. To balance exploration/exploitation, the Upper Confidence bound applied to Trees (UCT) (Kocsis & Szepesvári, 2006) is used as reward for sampling. Considering a node in tree a_i, our reward is defined as:

$$R(a_i) = C(a_i) + \lambda\sqrt{log(|parent(a_i)|)/|a_i|}, \tag{3.3}$$

in which the second term is for exploration. We use function $|a_i|$ to show number of times node a_i is visited, with the constant λ controlling the exploration/exploitation trade-off. $parent(a_i)$ indicates the parent node of a_i. We define C as:

$$C(a_i) = \beta\, C(a_i) + (1-\beta)\, Acc(f_a(\mathcal{X}_{val}, w)), \tag{3.4}$$

which is a smoothed version of the validation accuracy of architecture a, with smoothing factor β, to account for the noisy on-line estimation on mini-batches.

Training Algorithm: The training pipeline for our method is shown in algorithm 3.1.

Algorithm 3.1 Simplified pseudo-code of our training pipeline

Algorithm: NAS by learning a hierarchical search space

Input: $\mathcal{S}$: Search Space; $\mathcal{X}_t, \mathcal{X}_v$: mini-batches of training and validation data; f_a: model with architecture a; w_p, w: weights of the model after warm-up and and final model initialized randomly; e_{pt}, e_{MCTS}: warm-up and MCTS iterations; α: learning rate; β: smoothing factor; λ : exploration parameter of UCT.

#MCTS warm-up
while $epochs \leq e_{pt}$ **do**
  $a \leftarrow$ sample from $\mathcal{U}(|\mathcal{S}|)$
  $w_p \leftarrow w_p - \alpha \nabla_{w_p} \mathcal{L}(f_a(\mathcal{X}_t, w_p))$
end while
#build the search tree
for $a^i \in \mathcal{S}$ **do**
  Output vector $o^i \leftarrow f_{a^i}(\mathcal{X}_v, w_p)$
end for
Distance matrix $D_{ij} = dist(o^i, o^j)$
Binary tree $\mathcal{T} \leftarrow aggl\_clustering(D)$
#main training with MCTS
while $epochs \leq e_{MCTS}$ **do**
  #sample an architecture a
  a = []
  $node$ = "root"
  push(a,$node$)
  while $not(is\_leaf(node))$ **do**
    $node \leftarrow sample(next(node))$ with Boltzmann sampling as in Eq.(3.2)
    push(a,$node$)
  end while
  #update model w and accuracy
  $w \leftarrow w - \alpha \nabla_w \mathcal{L}(f_a(\mathcal{X}_t, w))$
  $accuracy \leftarrow \text{Acc}(f_a(\mathcal{X}_{val}, w))$
  $node \leftarrow$ **pop**(a)
  #update rewards
  while $not(is\_root(node))$ **do**
    $parent \leftarrow$ **pop**(a)
    $C(node) \leftarrow \beta\, C(node) + (1 - \beta)\, accuracy$
    $R(node) \leftarrow C(node) + \lambda \sqrt{log(|parent|)/|node|}$
    $node \leftarrow parent$
  end while
end while

Output: Best architecture from $\mathcal{T}$ by sampling with $\lambda = 0$

First, a warm-up with random sampling of the architectures is performed in order to train an initial model f with parameters w_p. With this model we build a pairwise matrix $D_{i,j}$ that measures the distance of configuration i and j on the output space of the model. With this matrix, we use agglomerative clustering to build a binary tree that represents the hierarchy that will be used for the subsequent MCTS training. During training, an architecture is sampled from the tree, where at each node Boltzmann sampling with the learned probabilities is used.

Then, this architecture is used to update the model on a mini-batch of training data (for simplicity we did not include momentum in the gradient updates) and to estimate its accuracy on a validation mini-batch. The validation accuracy is smoothed with an exponential moving average and used as reward with UCT regularization for updating the node probabilities of the sampled architecture. To search for the final architecture after training, we sample k architectures without exploration ($\lambda = 0$) and rank them based on their performance on validation dataset, selecting the best as the final architecture.

3.5 Experiments

We evaluate our method on two macro search space benchmarks using CIFAR10 dataset (Krizhevsky *et al.*, 2009), and on ImageNet (Russakovsky *et al.*, 2015) with MobileNetV2-like (Sandler *et al.*, 2018) search space. We compare our proposed method with various sampling methods discussed in section 3.3. For MCTS methods, we start by a warm-up phase of uniform sampling. After this phase, we utilize learned representation of architecture for hierarchical clustering. We also use the recorded statistics to calculate UCT (eq. 3.3) and sample using eq. 3.2. For MCTS default tree, used by Su *et al.* (2021a), each layer of CNN is considered as a level of tree, with operations providing the branching. We compare with this method with and without soft node independence assumption (regularization).

Table 3.1 Accuracy and ranking on the Pooling benchmark on CIFAR10. We report the found architecture (represented with number of layers per feature map sizes), best and average of 3 training accuracy and ranking and search time for different methods

Method	Arch.	Best Acc.	Avg. Acc.	Best Rank	Avg. Rank	GPU hour
Default Arch.	[4,3,3]	90.52	-	15	-	-
Uniform	[4,3,3]	90.52	90.40 ± 0.08	15	17	1.5
MCTS	[4,4,2]	90.85	90.57 ± 0.21	12	15.3	2
Boltzmann	[3,5,2]	90.88	90.51 ± 0.12	11	15.3	3
Independent	[3,5,2]	90.88	90.86 ± 0.01	11	11.7	2
Mixtures (Roshtkhari *et al.*, 2023)	[5,3,2]	91.55	91.36 ± 0.27	4	5	6
MCTS + Reg.(Su *et al.*, 2021a)	[6,1,3]	91.78	91.42 ± 0.11	3	3.6	2
MCTS + Learned (ours)	[6,2,2]	91.83	91.72 ±0.12	2	3	2
Best	[7,1,2]	92.01	-	1	-	-

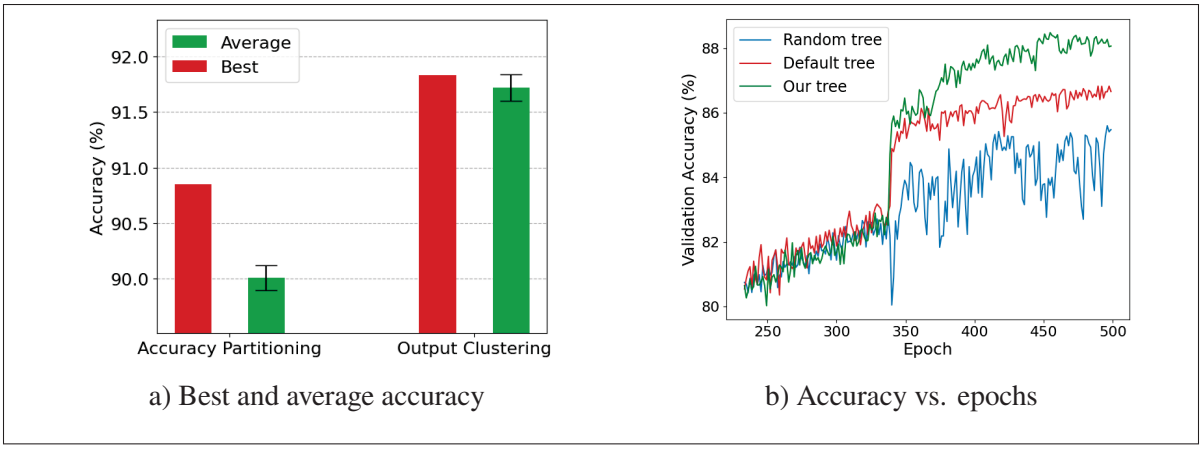
3.5.1 Pooling Search Space

To thoroughly investigate our proposed method, we use Pooling search space, a small yet challenging CIFAR10 benchmark consisting of 36 Resnet20-like (He *et al.*, 2016a) architectures. The only architecture parameter optimized is feature map sizes at each layer determined by placement of downsamplings operations. Specifically, each layer has the binary choice of performing downsampling (pooling) or maintaining resolution (identity). The main challenge of this search space lies in full weight sharing across all architectures that contributes to the inadequacy of several common methods (Roshtkhari *et al.*, 2023).

To demonstration each architectures, we use number of layers per feature map sizes (e.g. [4,3,3] meaning 4/3/3 layers in high/middle/low resolution). Our method achieves better results in comparison within similar or shorter search time (Table 3.1). While for MCTS (default tree design), the regularization proposed in Su *et al.* (2021a) appears to help, our method obtains better performance without requiring regularization.

3.5.2 Ablation Studies

We conduct several ablations to assess the convergence and performance of our method and explore alternative ways to design the hierarchy in Pooling search space.



a) Best and average accuracy

b) Accuracy vs. epochs

Figure 3.4 (a) Best and average accuracy for partitioning based on validation accuracy vs. output clustering. (b) Accuracy over epochs for several training strategies. After warm-up, our approach is constantly better than default or a random tree

Partitioning with accuracy: We compare our method with using architecture accuracy to partition search space (See section 3.4 (Hierarchy)). We used validation accuracy estimated from the supernet to construct a binary tree by recursively splitting architectures based on their accuracy (top 50% vs. bottom 50%). The resulting MCTS on this tree is notably worse (Figure 3.4.(a)). This highlights the shortcoming of using inaccurate supernet performance estimation as the basis for search space factorization.

Branching quality and NAS convergence: To demonstrate the crucial role of learning tree structure for MCTS efficiency and convergence, we compared MCTS using our learned tree with default tree, and with a binary tree created from a random distance matrix (see Figure 3.4.(b). For fair comparison we use the number of iterations for all methods. While the average accuracy of the supernet increases in general over epochs, the branching quality can affect how the search space is explored. After a warm-up period for UCT with uniform sampling (epoch 340), our tree converges more rapidly to higher accuracy compared to default and random tree. This highlights the importance of a meaningful hierarchy for efficient exploration and faster convergence in NAS.

Alternative architecture representations: We explore two types of encodings as zero-cost representation alternatives to supernet outputs. While distances based on outputs measure semantic (functional similarity) distances, using encoding measures structural distances. Representing an architecture as a graph, the general encoding is the adjacency matrix, corresponding to the edges (or one-hot encoding of operations per layer). We also consider the categorical representation of the one-hot encoding as vectors as other alternative.

Table 3.2 Zero cost branching methods. We consider one-hot encoding of operations per layer or categorical vector representation. We also consider an exponential weighting scheme to increase the influence of earlier layers on distance

Encoding	Weighted	Best Arch.	Best Acc.	Avg. Acc.
Vector		[2,5,3]	90.89	90.63 ± 0.68
Vector	✓	[3,4,3]	90.92	90.81 ± 0.15
One-hot		[5,2,3]	90.96	90.60 ± 0.22
One-hot	✓	[5,1,4]	91.05	90.78 ± 0.21

Table 3.2 shows diminishing results compared to using outputs in both cases. We note that naively calculating the distances (same weight for all layers) is equivalent to considering each layer as an independent variable, while in fact the posterior layers have less importance than early layers. Therefore, we also consider an exponentially weighted encoding which leads to slight performance increase.

3.5.3 NAS-Bench-Macro Search Space

This benchmark consists of 8 layers of MobileNetV2 (Sandler *et al.*, 2018) blocks with operation choices {*Identity*, *MB3_K3*, *MB6_K5}* resulting in $3^8 = 6,561$ architectures. Table 3.3 presents results of several sampling based NAS methods. Our approach finds the best architecture in this search space.

Table 3.3 Accuracy and ranking on NAS-Bench-Macro. We compare our method and several approaches in terms of best, average accuracy and ranking. The architectures are represented with operation index per layer

Sampling Method	Arch.	Best Acc.	Avg. Acc.	Best Rank	Avg. Rank
Boltzmann	[12220111]	92.39	92.30 ± 0.10	406	453
Independent	[22120211]	92.44	92.29 ± 0.21	347	412
MCTS	[22221210]	92.74	92.51 ± 0.18	80	246
Uniform	[21222220]	92.79	92.58 ± 0.20	56	197
MCTS + Reg. (Su *et al.*, 2021a)	[12222222]	92.92	92.67 ± 0.18	21	112
MCTS + Learned (ours)	[22212220]	93.13	92.97 ± 0.12	1	6
Best	[22212220]	93.13	-	1	-

3.5.4 Search on MobileNetV2 Search Space

ImageNet (Russakovsky *et al.*, 2015) consists of 1.28 million training images in 1000 categories. In our experiments, we use 50k images of validation set as the test data to compare with other methods. To accelerate our training we use mixed precision and FFCV (Leclerc *et al.*, 2023) library. We use similar macro search space to previous works (Su *et al.*, 2021a; You *et al.*, 2020; Chu *et al.*, 2021c; Guo *et al.*, 2020b), based on MobileNetV2 blocks with optional Squeeze-Excitation (SE) (Hu *et al.*, 2018b) module. The total operation choice per layer is 13 resulting in 13^{21} search space size for 21 layers. The choices are convolution kernel size of $\{3, 5, 7\}$ and expansion ratio of $\{3, 6\}$, identity and SE option.

Similar to Su *et al.* (2021a), we use FLOPS reduction by defining a budget for our search. Leveraging the fact that FLOPs can be used as a zero-cost proxy for architecture performance (Chen, Lin, Sun & Li, 2021b), we sample only within a certain range of target budget ($[0.99, 1]\times$ budget). In Table 3.4, we compare our method with several NAS approaches (taken from Su *et al.* (2021a)) on the upper part of the table, while we compare with our sampling based approaches on the bottom part. Our method shows improved performance (within FLOPs budget) with lower computational cost than compared methods. We attribute this advantage to the learned structure of the tree, that allows a quicker learning of the promising architectures.

Table 3.4 Comparison of accuracy and computational cost on ImageNet classification task. The architecture are searched on MobilenetV2-based search space. We consider light weight models with target budget of 280 (left) and 330 (right) MFLOPs. In the top part of the table we directly report results of other NAS methods, while at the bottom we report results of our baselines and our approach

Method	**Best Acc.**	**FLOPs (M)**	**Params. (M)**	**GPU days**	**Method**	**Best Acc.**	**FLOPs (M)**	**Params. (M)**	**GPU days**
MobileNetV2	72.0	300	3.4	-	Proxyless-R	74.6	320	4	15
MnasNet-A1	75.2	312	3.9	288	SPOS	76.2	328	-	13
SCARLET-C	75.6	280	6.0	10	SCARLET-B	76.3	326	5.2	22
GreedyNAS-C	76.2	284	4.7	7	FairNAS-C	76.7	325	5.6	-
MTC_NAS-C	76.3	280	4.9	12	MCTS_NAS-B	76.9	330	6.3	12
Uniform	72.2	277	4.6	~ 5	Uniform	73.1	319	4.8	~ 6
Boltzmann	73.1	278	4.7	~ 5	Boltzmann	73.8	330	6.3	~ 5
MCTS + Reg.	76.0	280	4.9	> 12	MCTS + Reg.	76.8	330	6.3	> 12
Ours	76.7	280	4.9	~ 7	Ours	77.4	330	6.2	~ 8

3.6 Conclusion

In this work, we have introduced a novel method to design a hierarchical search space for NAS. We have highlighted the shortcomings of node independence assumption used in popular NAS methods and the impact of the hierarchical search space design on search quality and efficiency. We have shown that by simply learning an appropriate hierarchical representation of the architectures in the search space, we achieve state-of-the-art results with MCTS, without requiring any other form of regularization and with a reduced amount of training. In future work, we will investigate other and more efficient ways to build our hierarchy, focusing in particular to zero cost approaches, which seem promising but still inferior to the used representation.

CHAPTER 4

ITERATIVE MONTE CARLO TREE SEARCH FOR NEURAL ARCHITECTURE SEARCH

Recent work has shown MCTS as an effective approach for NAS in producing competitive architectures. However, the performance of the tree search is highly sensitive to the node visiting order. If the initial nodes are highly discriminative, good configurations can be efficiently found with minimal sampling. In contrast, non-discriminative initial nodes require exploring an exponential number of nodes before finding good solutions. In this chapter, we present an iterative NAS approach to jointly train the recognition model with MCTS and learn the optimal node ordering of the tree. With our approach, the order of node visits in the tree is iteratively refined based on the estimated performance of the nodes on the validation set. With this approach, good architectures are more likely to naturally emerge at the beginning of the tree, improving the search process. Experiments on two image classification benchmarks and a semantic segmentation task show that the proposed method can improve the performance of MCTS, compared to state-of-the-art MCTS approaches for NAS.

4.1 Introduction

MCTS is a powerful approach for non-differentiable problems, particularly those involving discrete actions (Browne *et al.*, 2012; Costa & Pedreira, 2023). However, sampling efficiency is crucial for MCTS to minimize unnecessary exploration and achieve faster convergence to good solutions (Świechowski *et al.*, 2023). Poor sampling efficiency, especially with large search spaces and hard-to-distinguish branches, can hinder its efficient application. MCTS is a compelling approach for NAS due to its inherent exploration-exploitation mechanism and ability to handle imperfect evaluations. This is particularly important in one-shot NAS methods, where the recognition model training and architecture search are performed simultaneously.

One-shot methods based on weight-sharing (Pham *et al.*, 2018), reduce the cost of evaluating candidate architectures. Essentially, an over-parameterized model containing all possible architectures, called "supernet", is trained. The supernet is then used to estimate the performance

of an architecture by inheriting the weights. This eliminates the need to train individual architectures. However, shared weights can introduce bias and interference during supernet training and lead to rank inconsistency for the sub-nets compared to standalone training performance (Yu *et al.*, 2019; Bender *et al.*, 2018; Zhao *et al.*, 2021a). Jointly performing MCTS and supernet training may benefit the search, by gradually reducing the number of sampled architectures with shared weights.

Several works have explored MCTS for NAS, with various designs and search methods (Wang *et al.*, 2021b; Zhao *et al.*, 2021b, 2024; Su *et al.*, 2021a; Roshtkhari, Toews & Pedersoli, 2025b). Some have used MCTS only for the search phase (Wang *et al.*, 2021b; Zhao *et al.*, 2021b), while others have utilized it for both training and search (Su *et al.*, 2021a; Roshtkhari *et al.*, 2025b) as in our case. Design of the search tree is crucial for NAS efficiency: Su *et al.* (2021a) use a manual tree design, considering each layer of the CNN as a level in the tree, with operation choices as branches. Wang *et al.* (2021b) and Zhao *et al.* (2021b) partition the search space based on the performance of a trained supernet.

In general, applying MCTS for NAS without additional constraints and regularization leads to poor performance (Su *et al.*, 2021a). The manual tree design proposed by Su *et al.* (2021a) requires additional regularization to compensate for low sampling rates of nodes. This regularization (soft independence assumption) undermines the joint contribution of operations in layers by sharing reward information among nodes.

A promising solution is to improve branching quality by learning an optimized tree structure from the data (Roshtkhari *et al.*, 2025b). As it is computationally prohibitive to sample the entire tree with high frequency, an optimized tree should prioritize sampling regions with high ground truth performance more frequently.

A reasonable approach is partitioning "good" and "bad" regions of the search space by clustering based on estimated performance. However, the quality of this pre-ordered tree relies on accurate rankings of architectures, which depend on the quality of the sampling, generating a typical chicken-and-egg problem. Previous approaches (Wang *et al.*, 2021b; Zhao *et al.*, 2021b) tackle

this problem by learning how to separate the search space while performing the architecture search. In those works, the recognition model is given, assuming it already provides good estimations, which renders the factorization of the search space based on static estimations. Nevertheless, when the recognition model is trained while learning the search space hierarchy, the problem becomes much more complex and does not allow for static solutions.

In this work, we propose a simple approach in which the tree structure is reorganized as the supernet training progresses. At each iteration, MCTS is used to guide supernet training, and the performance estimated from this supernet is used to refine and reorganize the search tree. We show that, while initial performance estimates may not be a reliable measure for constructing the search tree, an iterative application of MCTS and tree reorganization can gradually guide the search towards high-performing architectures by gradually increasing their sampling rates.

The main contributions of our work are as follows:

- We present a new method of partitioning NAS search space into a search tree, based on performance estimates obtained from the supernet. We show that by iterative application of MCTS and tree reorganization, we can obtain competitive architectures without the prohibitive cost or constraints of previous methods.
- We show that with a careful balance of exploration and exploitation, the number of iterations needed is small and the overhead cost is negligible.
- We empirically validate our method for two computer vision tasks of image classification and semantic segmentation and on three datasets. We show that compared to other MCTS-NAS methods that perform supernet training and MCTS jointly, our approach achieves competitive performance without regularization and with linear computational complexity.

4.2 Background on NAS for Semantic Segmentation

The main focus of NAS for computer vision has been on image classification task with CNNs, leaving other tasks (e.g. dense prediction) less developed (Mohan *et al.*, 2023). NAS for semantic segmentation is more challenging: Compared to classification, it requires higher

computational cost and memory since the input images and feature maps generally have higher resolution, and the architectures are deeper and more complex to enable per pixel prediction. In the case of medical images, data can be 3D (Ali, Essaid, Moalic & Idoumghar, 2024), while some applications require real-time inference. Therefore, a good trade-off between performance and efficiency is essential.

Furthermore, fewer benchmarks (Duan *et al.*, 2021; Mehta *et al.*, 2022; Zhao *et al.*, 2024) are available for segmentation task (Chitty-Venkata *et al.*, 2023), adding to the computational cost of evaluating NAS methods and reproducibility. Contrary to classification, the architectures require a decoder or task-specific head, which can be searched separately (Chen *et al.*, 2018a; Ghiasi, Lin & Le, 2019; Xu, Yao, Zhang, Liang & Li, 2019a) or jointly (Guo *et al.*, 2020a; Yao, Xu, Zhang, Liang & Li, 2020) with the encoder part. To improve efficiency, many works use differentiable methods (Liu *et al.*, 2019a; Saikia, Marrakchi, Zela, Hutter & Brox, 2019; Xu *et al.*, 2019a; Guo *et al.*, 2020a), while others use reinforcement learning or evolutionary algorithm (EA) (Ghiasi *et al.*, 2019; Du *et al.*, 2020b; Wang *et al.*, 2020e; Bender *et al.*, 2020).

Auto-DeepLab (Liu *et al.*, 2019a), one of the most prominent NAS works for segmentation, proposed to use a bi-level (hierarchical) search space (macro-level: resolution and channels ; micro-level: cell or blocks) and applied DARTS iteratively to these two levels. Application of differentiable approach resulted in significant reduction in computational cost compared to DPC (Chen *et al.*, 2018a), making NAS feasible for segmentation. DCNAS (Zhang *et al.*, 2021b) builds upon this, constructing a densely connected search space and using path and channel level sampling to reduce the computational cost. This enabled to directly search on the target task without using a proxy task or dataset.

Several works aim for real-time applications by applying latency constraints (Shaw, Hunter, Landola & Sidhu, 2019; Chen *et al.*, 2019a; Lin *et al.*, 2020). The latency of each architecture is often estimated and incorporated into the loss function for a differentiable search. Other works apply multi-objective NAS for efficient segmentation (Lu *et al.*, 2022; Yu *et al.*, 2024). Another approach is to extend the application of zero-cost proxies developed for classification

(Abdelfattah *et al.*, 2021; Lee & Ham, 2024) to segmentation. SasWOT (Zhu, Li, Wu & Sun, 2024) proposes to use EA and learn a zero-cost proxy specifically for semantic segmentation. This zero-cost proxy is then used to evaluate architectures at initialization, greatly reducing the computational cost of NAS.

4.3 Method

We propose an iterative MCTS algorithm where the tree structure is refined at each iteration using accuracy estimates of the leaf nodes on a validation set. This progressive tuning of the tree structure allows compensating for noisy and inaccurate estimation obtained from the supernet. Our iterative algorithm is depicted in figure 4.1.

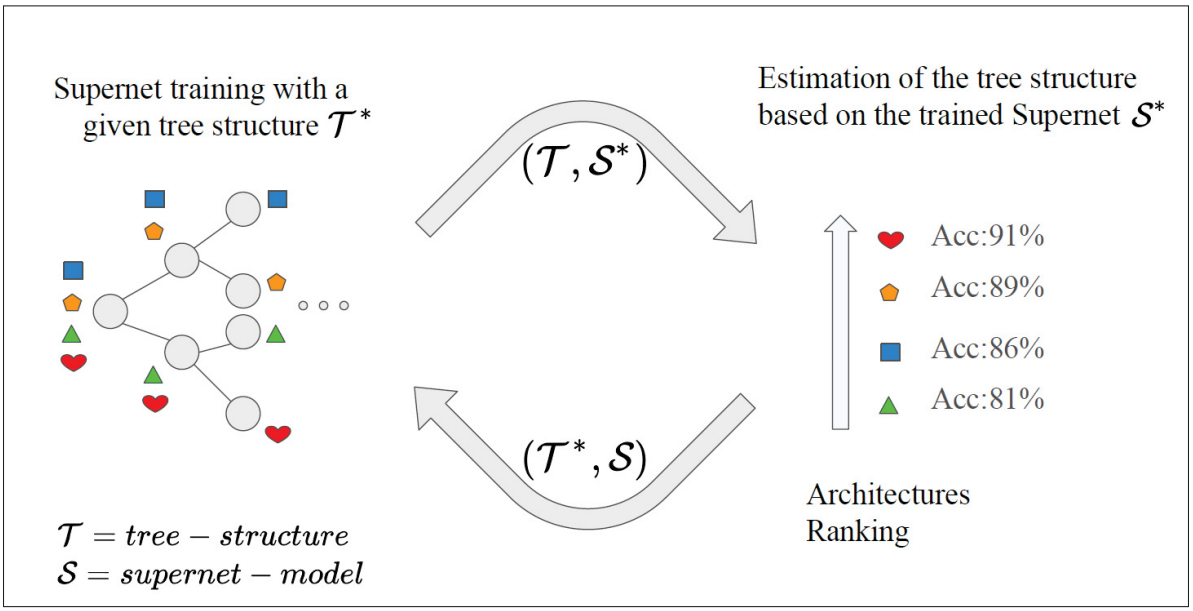


Figure 4.1 Overview of our iterative MCTS. At each iteration, the tree structure $\mathcal{T}^*$ is provided, and a new supernet $\mathcal{S}$ is trained. With the trained supernet $\mathcal{S}^*$, a new tree structure $\mathcal{T}$ is estimated based on the performance (accuracy) of architectures on validation data using supernet. These iterations are repeated until convergence

During the recognition model training, MCTS samples from the supernet, while estimating the probabilities of each node, such that the architectures with higher estimated accuracy are generally sampled more often. With the constructed MCTS, we estimate the accuracy of each

architecture to establish a comprehensive ranking. This ranking is then used to build a new search tree by simply organizing the architectures based on their validation scores. Given the new tree structure, we can repeat the first phase of training. This iterative procedure continues for a predefined number of iterations. In the following subsections, we will explain each part of this algorithm in detail.

4.3.1 Initialization

In order to start the iterative procedure, we need to provide an initial tree structure, $\mathcal{T}_{init}$, and an initial model for the supernet, $\mathcal{S}_{init}$. In our experiments, we initialize the supernet with single-path uniform training: for each minibatch, we train the supernet by randomly sampling an architecture a from the search space. With $\mathcal{S}_{init}$, we can compute the accuracy of each architecture on the validation data. This allows us to rank the architectures and provides an initial structure for the tree $\mathcal{T}_{init}$.

4.3.2 Training the supernet by Sampling with MCTS

We define a supernet $\mathcal{S}$ as a recognition model that includes all operations and parameters required to build any feasible architecture within our search space. We also use a tree structure $\mathcal{T}^*$ that defines how the set of architectures is divided into smaller subgroups, going from entire search space at root to leaf nodes representing individual architectures. Given a tree structure $\mathcal{T}^*$, each leaf node corresponds to an architecture a. Architecture a is sampled by traversing $\mathcal{T}^*$ from the root and making node selections among successors (children). A common method to balance exploration and exploitation for node selection is to use UCT (Kocsis & Szepesvári, 2006). For each visited node i, two values are recorded: visit count $n(i)$ and the average reward of the node $\tilde{A}(i)$. The UCT value is then calculated as:

$$UCT(i) = \tilde{A}(i) + C\sqrt{\frac{log(n_p(i))}{n(i)}} \tag{4.1}$$

and $n_p(i)$ is the function representing the number of times i's parent node was visited, and $C \in \mathbb{R}_+$ is the constant that balances exploitation (the first term) and exploration (the second term). As the node with the highest UCT score is generally selected among sibling nodes, we encourage further exploration by applying Boltzmann sampling (Painter *et al.*, 2024). Therefore, the probability of a sampling node i is calculated as:

$$P(i) = \frac{exp(UCT(i)/T)}{\sum_j exp(UCT(j)/T)} \tag{4.2}$$

where the summation is performed over all sibling nodes of i. The temperature term T controls the probability distribution, with $T \to 0$ corresponding to categorical distribution.

Using single-path approach (Guo *et al.*, 2020b), the supernet $\mathcal{S}$ is trained on sampled architecture a for one iteration. To avoid overfitting on training set, we use validation performance to calculate the reward for each architecture. At each iteration, $\mathcal{S}$ is trained on one minibatch of training data and then evaluated on one minibatch of validation data to yield performance $A(a)$. This value is then backpropagated to update the rewards along the selected path in the tree. The reward is calculated using a weighted moving average:

$$\tilde{A}(i) \leftarrow \beta \cdot \tilde{A}(i) + (1 - \beta) \cdot A(a) \tag{4.3}$$

where $\beta \in [0, 1]$ is the weighting factor. The process of sampling, training, evaluation, and reward backpropagation is repeated for a specified number of epochs. Since a static tree is used, the expansion and rollout phases of traditional MCTS are omitted. Once this phase is finished, the trained supernet $\mathcal{S}^*$ is passed to the next phase to update the tree structure. To select the final architectures for evaluation, we use Equation 4.1 with $C = 0$, since we do not need exploration in this stage.

4.3.3 Updating the Tree Structure

Our goal is to leverage the trained supernet $\mathcal{S}^*$ to construct an improved hierarchy that guides exploration towards promising nodes. By placing superior nodes in preferred paths, fewer nodes need to be explored, allowing the allocation of resources to these nodes and faster convergence of search. To achieve this, we construct a binary tree that represents this hierarchical structure based on the ranking of leaf nodes. Given a trained supernet $\mathcal{S}^*$, we rank sampled architectures based on their validation performance (accuracy). Since evaluating on the entire validation set is computationally expensive, we approximate the performance by evaluating architectures on only a few minibatches of validation data. With this ranking, a bottom-up approach then iteratively merges the two nodes (architectures or existing clusters) with the lowest average ranks, and the process is continued until a new tree, $\mathcal{T}^*$ is constructed.

4.3.4 Iterative MCTS

We treat the tree structure $\mathcal{T}$ as a heuristic, which provides a good starting point, but is updated and reorganized at each iteration of MCTS as new information comes in. At each iteration of MCTS, with a good balance of exploration/exploitation, value estimates of the nodes are refined, reflecting the algorithm's learned understanding of the tree. Therefore, at each iteration, we update $\mathcal{T}$ based on the newly acquired ranking. In section 4.4.1, we use the same ranking criteria for tree initialization and analyze alternatives in section 4.4.2.

Algorithm 4.1 outlines our approach. To start iterative MCTS method, an initial tree structure and supernet are provided to the algorithm (Section 4.3.1). The main loop is composed of a first loop in which a branch of the tree is stochastically sampled based on node probabilities (Equation 4.2). This process selects an architecture a, which is then used for training the supernet $\mathcal{S}$ for one minibatch. The probabilities of the tree P are then updated based on the accuracy of the given architecture on a validation minibatch. Finally, after several training iterations, the architectures are ranked and used to update the tree structure, and the training of the supernet is started again with the new tree structure.

Algorithm 4.1 Simplified pseudo-code of our iterative MCTS

Algorithm: Iterative MCTS for NAS

Input: M: number of MCTS iteration; K: iteration of each MCTS; $\mathcal{X}_t, \mathcal{X}_v$: minibatches of training and validation data; $\mathcal{S}_{init}$: Initial supernet, $\mathcal{T}_{init}$: Initial tree structure.

$m = 0, k = 0$
$\mathcal{T} \leftarrow \mathcal{T}_{init}, \mathcal{S} \leftarrow \mathcal{S}_{init}$
while $m \leq M$ **do**
    $P \leftarrow init(\mathcal{T})$ #initialize the tree probabilities with the new structure
    while $k \leq K$ **do**
        $i \leftarrow i_{root}$ #start from the root node
        $path = []$ #keep the entire path to backpropagate probabilities
        while i *not* $leaf$ **do**
            $path.add(i)$
            $a = sample(P(i))$ #sample based on the tree probabilities
            $update(n(a))$ #update count for child node
            $a \leftarrow i^*$
        end while
        $\mathcal{S}.train(\mathcal{X}_t, a)$ #train the supernet
        $Acc(a) = \mathcal{S}.evaluate(\mathcal{X}_v, a)$ #estimate the accuracy of the architecture a
        $P \leftarrow backpropagate(Acc(a), path)$ #update the tree probabilities
    end while
    $\mathcal{T} \leftarrow Rank(Acc)$ #rank the architectures based on accuracies and build the new tree
end while

Output: Best architecture from $\mathcal{T}$ by sampling with $C = 0$

In a static tree for MCTS, the UCT does allow for some exploration of initially misclassified "bad" branches of the tree (by tuning hyperparameter C in Equation 4.1). However, the hierarchy does not have a chance to improve itself based on the information learned from this exploration; a potentially good architecture can get permanently placed in a bad region. Manual tree design (Su *et al.*, 2021a) or relying on potentially inaccurate supernet performance estimates to partition search space (Wang *et al.*, 2020c), also do not guarantee an optimized tree structure. We propose that with well-balanced exploration and exploitation, good architectures can be identified and the tree can be restructured iteratively to prioritize these architectures. To achieve this, we propose to re-rank architectures and reorganize the search tree in each iteration of MCTS.

4.4 Experiments

In this section, we first apply our iterative MCTS for NAS on two image classification search spaces: Pooling search space (Roshtkhari *et al.*, 2023) on CIFAR10 dataset and NAS-Bench-201 (Dong & Yang, 2020) on ImageNet-16-120 dataset, and perform ablation studies on these tasks. We then show a promising application of our method to a semantic segmentation task in a trellis search space inspired by Auto-DeepLab (Liu *et al.*, 2019a). In our experiments, to obtain a higher quality ranking, we evaluate architectures on few minibatches of validation data. In ablation studies, we show that this approach provides higher quality final architectures.

4.4.1 Image Classification

We performed experiments on two NAS benchmarks: the Pooling benchmark (Roshtkhari *et al.*, 2023) and NAS-Bench-201 (Dong & Yang, 2020). We compare our methods with non-hierarchical and MCTS methods. Uniform sampling serves as a baseline for comparison. Boltzmann softmax sampling (Cesa-Bianchi *et al.*, 2017) is a simple method that offers a biased search by adjusting uniform probability distribution, with a temperature hyperparameter controlling the balance of exploration/exploitation.

For comparison with MCTS methods, we consider MCTS-default (the manual design proposed by Su *et al.* (2021a)) and MCTS-prioritized (same manual design with their proposed additional regularization). Additionally, for both benchmarks, we compare with the differentiable method DARTS (Liu *et al.*, 2018b). For the Pooling benchmark, we also report results of DARTS+GAEA (Li *et al.*, 2020d) and "Balanced Mixtures" (chapter 2 and (Roshtkhari *et al.*, 2023)), a method that learns non-hierarchical search space partitioning with a specialized supernet per partition. For NAS-Bench-201, we compare with various methods: GDAS (differentiable), ENAS (RL), RSPS (random search), and NASWOT (zero-cost proxy).

The Pooling search space is a small yet challenging search space (introduce in chapter 2) featuring Resnet-like (He *et al.*, 2016a) architectures, with the goal of optimizing feature map sizes at each layer. Due to notable low rank correlation between supernet estimates and ground truth, it is a

suitable benchmark for demonstrating the effectiveness of our approach. As presented in table 4.1, our method outperforms its counterparts with similar or less search time. We also note that in this benchmark, several methods achieve performance close to the upper bound, and therefore, significant net improvements in accuracy is not possible. We report results on NAS-Bench-201 dataset for ImageNet-16-120 in table 4.2. In this benchmark, our method outperforms other common NAS methods.

Table 4.1 Comparison results on CIFAR-10 using pooling search space

	Accuracy		
Method	**Best**	**Average**	**Search Time**
Default (Resnet20)	90.52 ± 0.15	-	-
DARTS (Liu *et al.*, 2018b)	89.23 ± 0.08	-	12
DARTS + GAEA (Li *et al.*, 2020d)	89.12 ± 0.10	-	12
Balanced Mixtures (Roshtkhari *et al.*, 2023)	91.55 ± 0.12	-	6
Uniform Sampling	90.52 ± 0.15	90.40 ± 0.08	1.5
Boltzmann Sampling	90.88 ± 0.08	90.51 ± 0.12	3
MCTS-default	90.85 ± 0.12	90.57 ± 0.21	2
MCTS-prioritized (Su *et al.*, 2021a)	91.78 ± 0.11	91.42 ± 0.11	2
Iterative MCTS (ours)	91.83 ± 0.12	91.81 ± 0.02	~ 2
Best Architecture	92.02 ± 0.18	-	-

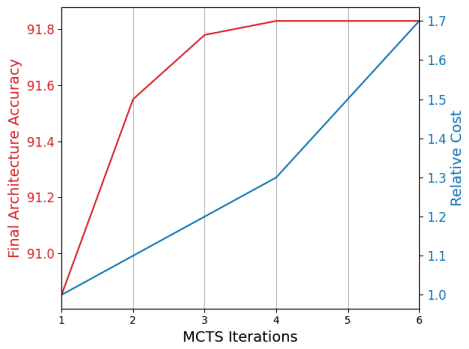
4.4.2 Ablation Studies

Number of MCTS iterations: We analyze the optimal number of iterations for our method on Pooling benchmark in figure 4.2 (a). For each additional iteration of MCTS, total training steps is slightly increased to allow adequate sampling rate for nodes. Iterative MCTS is clearly superior to non-iterative MCTS in terms of the found architecture, and there seems to be an optimal number of iterations, beyond which the final result do not improve. The number of iterations and training cost can be treated as a trade-off when the training budget is limited.

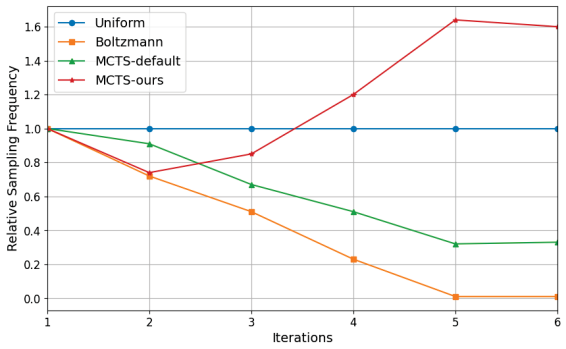
The effectiveness of iterative MCTS: To demonstrate that the iterative process helps in guiding the search toward promising architectures, we calculated the sampling frequency of the top-5 architectures in the Pooling benchmark throughout the supernet training. In figure 4.2 (b), we

Table 4.2 Comparison results on ImageNet-16-120 using NAS-Bench-201 (Dong & Yang, 2020). Results for non-MCTS methods are taken from papers

	Accuracy		
Method	**Best**	**Average**	**Relative Search Time**
DARTS (Liu *et al.*, 2018b)	-	16.4	3
ENAS (Pham *et al.*, 2018)	-	16.3	3.7
RSPS (Li & Talwalkar, 2020)	-	31.1	2.1
GDAS (Dong & Yang, 2019b)	-	41.8	8
NASWOT (Mellor *et al.*, 2021)	-	38.3	-
Uniform Sampling	31.2	31.0 ± 0.2	3.8
Boltzmann Sampling	31.1	30.8 ± 0.3	4.5
MCTS-default	41.7	40.2 ± 0.4	4.1
MCTS-prioritized (Su *et al.*, 2021a)	41.7	41.4 ± 0.2	3.1
Iterative MCTS (ours)	42.2	41.9 ± 0.2	3.1
Best Architecture	47.3	-	-



a) Quality of architectures



b) Relative sampling frequency

Figure 4.2 (a) Effect of the number of MCTS iterations on the quality of found architectures. Multiple iterations of MCTS can find superior architectures compared to a single iteration, with only a slight increase in training time. (b) Relative sampling frequency of the top-5 architectures. The x-axis corresponds to the number MCTS iterations used in our method. For other methods, we use equivalent time during training. The first iteration corresponds to the uniform sampling for warm-up or pretraining of various methods

compare sampling frequency of several methods. The frequency is recorded and averaged over 5 runs for each method. For fairness, we considered a fixed number of training iterations for

all methods. For Boltzmann and MCTS-default, where the search converges to suboptimal configurations, the frequency unsurprisingly decreases. By increasingly sampling architectures outside top-5, supernet is guided towards those architectures, leading to lower final architecture accuracy. Our method show gradual increase in sampling these architectures, demonstrating its ability to improve sampling rate for good architectures.

Rank-preserving ability: While our method is able to concentrate training on promising architectures, we further analyze the ability of the supernet to distinguish and correctly rank these architectures correctly. In other words, we would like to know if the trained supernet has high enough quality to distinguish the top architectures. Calculating Kendall's tau coefficient of top architectures can indicate rank preservation (Zhang *et al.*, 2024a). In table 4.3, we investigate the rank correlation of the top-10 architectures in Pooling benchmark with ground truth ranking by calculating Kendall's tau coefficient. We note that compared Boltzmann and MCTS-default our method achieves better rank correlation.

Table 4.3 Comparison of ranking correlation between ground truth accuracy and supernet prediction for top-10 architectures

Method	Kendall's tau
Uniform Sampling	0.14
Boltzmann Sampling	0.11
MCTS-default	0.32
Iterative MCTS (ours)	0.41

Ranking metric for tree reconstruction: At each iteration of MCTS, the performance of architectures need to be evaluated to calculate ranking. Evaluating on few minibatches of validation data provides a balance of accuracy and computational cost. Alternatively (at $M > 1$) one can use moving average from equation 4.3 which provides a smoother estimates and does not require further validation. In table 4.4 we compare various metrics for NAS-Bench-201.

Table 4.4 Comparison of final accuracy based on evaluation on B minibatches of validation data and using moving average of accuracy (equation 4.3)

Performance Metric	Final Accuracy
Validation Accuracy ($B = 1$)	41.6
Validation Accuracy ($B = 2$)	42.1
Validation Accuracy ($B = 3$)	42.2
Moving Avg. Accuracy	40.5

4.4.3 Semantic Segmentation

To evaluate our approach for segmentation task, we perform our experiments on a search space inspired by Auto-DeepLab (Liu *et al.*, 2019a). This search space is based on DeepLabV3+ (Chen *et al.*, 2018b), in which the encoder consists of a found architecture and the decoder is not altered. Auto-DeepLab uses a bi-level search space (macro and micro) and gradient descent (DARTS) to optimize both levels iteratively. In this work, we focus on the network skeleton (macro) portion of the search space. This reduces the search space size from 10^{19} to 2.9×10^4. For semantic segmentation, mean Intersection Over Union (mIOU) is the standard performance metric; therefore we replace accuracy $A(a)$ in equation 4.3 with mIOU for this task. We report results of our implementations of several methods in table 4.5 on Cityscapes (Cordts *et al.*, 2016) dataset.

Table 4.5 Comparison of searched architectures with various NAS methods for semantic segmentation task on Cityscapes dataset. Search space consists of macro (network) level of Auto-DeepLab (Liu *et al.*, 2019a). For consistency we report results from our own implementation

Method	Best	Average	Time (GPU Days)
Uniform Sampling	53.11	50.42	~ 4
MCTS-prioritized (Su *et al.*, 2021a)	75.32	73.1	~ 3
Auto-DeepLab-S (Liu *et al.*, 2019a)	76.91	76.73	-
Iterative MCTS (ours)	77.11	77.07	~ 2.5

4.4.4 Implementation Details

Datasets and Hyperparameters For all datasets in our experiments, we split training data 50/50 to use as training/validation. Unless otherwise mentioned, our experiments were run 3 times to report average and standard deviations. To tune hyperparameters, we performed either grid search or used similar hyperparameters when comparing with other papers.

For all our experiments $\beta = 0.95$. To train supernet on pooling search space (Javan, Toews & Pedersoli, 2023) for our experiments for classification task, we used SGD with learning rate 0.1 with cosine annealing, weight decay $1e-2$ and batch size 256 and we train for 500 epochs. For experiments on NAS-Bench-201 (Dong & Yang, 2020), we train for 50 epochs with SGD with learning rate 0.025 and cosine annealing. For image segmentation on Auto-DeepLab, we use same hyperparameters as original paper, using SGD with initial learning rate 0.025 decayed by annealing and weight decay 0.0003. Furthermore, we utilize mixed-precision operations and FFCV (Leclerc *et al.*, 2023) library to accelerate training in our experiments.

For the benchmarks for classification, we directly reported the searched architecture performance. For segmentation task, we retained all architectures in table 4.5 with same setting as Liu *et al.* (2019a).

4.5 Conclusion

In this chapter, we presented a novel MCTS approach for NAS. We developed an iterative method that progressively refines the search space hierarchy based on the observations from the supernet. Compared to previous applications of MCTS for NAS, the proposed approach does not use any specific knowledge to refine the search, making it more general and flexible. Our proposed approach iteratively updates the structure of the tree to favor high-accuracy architectures. We empirically evaluated our method on two classification tasks (CIFAR-10 on Pooling benchmark, ImageNet-16-120 on NAS-Bench-201) and a semantic segmentation on Cityscapes dataset. The proposed approach shows how to improve the performance of a supernet

by iteratively estimating the best sampling tree and the recognition model. However, the approach assumes that the iterative refinement starts from a relatively good initialization of the supernet. In our experiments, we use as initialization a supernet trained with uniform sampling which performed adequately well. Nevertheless, if the initial recognition model ranking estimates are not sufficiently correlated with the true architecture ranking, the self-refining approach may not lead to improved results.

CHAPTER 5

DEBIASED ONE-SHOT NAS VIA DENSITY-AWARE SAMPLING

One-shot NAS is based on training a supernet, a single model from which many different architectures with shared weights are sampled and updated during training. Training the shared weights of the supernet is much more computationally efficient than training each architecture independently. However, during the supernet training, architectures with similar gradients (dense regions in the gradient space) would cooperate with each other, increasing their training, while architectures in sparse (low-density) regions would not receive as much training benefit from others. This does not allow all architectures to be trained with the same amount of effective updates, producing a training bias that favors architectures in denser regions. As a consequence, the correlation between the supernet estimations and the actual performance of models independently trained is reduced. This negatively affects the supernet's ability to select good architectures once trained.

In this chapter, we propose two computationally feasible ways for different computational budgets and search spaces to approximate the architecture densities and implement a density-aware debiasing mechanism for supernet training. We propose a fine-grained density calculation for numerable smaller search spaces and an online density approximation based on density prototypes from a clustering algorithm for larger spaces. We validate our method on CIFAR10, CIFAR100, and ImageNet datasets using various search strategies and show consistently improved results compared to several single-path one-shot supernet training methods.

5.1 Introduction

Common one-shot NAS methods based on weight-sharing use the supernet as a proxy to efficiently estimate architecture performances on a validation set. By training the supernet, a single set of shared weights learns to effectively parametrize a large set of diverse architectures, so that the inherited weights can adequately evaluate and rank those architectures. Therefore, the

core assumption is that in single-path one-shot NAS, similar architectures can mutually benefit when one of them is sampled, requiring fewer training iterations to jointly train all architectures.

However, in one-shot NAS, the updates of certain architectures might be detrimental to others (Zhang *et al.*, 2020c; Xu *et al.*, 2022). This effect creates the well-known performance gap and poor rank correlation between supernet evaluations and independently trained architectures (Ning *et al.*, 2021; Roshtkhari *et al.*, 2023). Most crucially, this interference is not random: it depends on the similarity of the gradient updates. As demonstrated in figure 5.1, architectures with similar gradients will produce a positive interference, while architectures with different gradients will produce a negative interference. Therefore, architectures that have gradients similar to several neighbors (i.e., high-density regions) tend to help each other. In contrast, architectures with different gradients from their neighbors (i.e., low-density regions) have a destructive effect on their gradient updates. Globally, this effect produces a clear bias in which denser regions will produce over-trained architectures, while lower-density regions will have under-trained architectures. The main goal of this chapter is to tackle this bias.

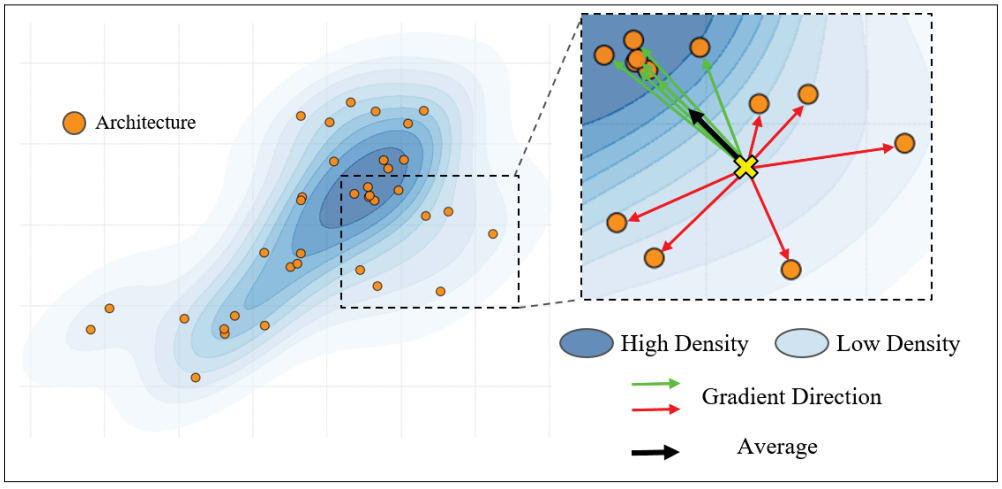


Figure 5.1 Training bias in uniform sampling supernet training. 2D projections of architecture gradients from the Pooling benchmark. When training with a uniform sampling of architectures, we found that architectures that share similar gradients (high-density gradient regions) tend to be globally over-trained, while the ones with different gradients (low-density gradient regions) will be under-trained. This produces a bias that favors architectures that lie in denser gradient regions

We propose a computationally affordable approach to reduce the bias induced by weight sharing during supernet training. To achieve this, we present two algorithms to estimate the density of architectures in gradient space. The first, designed for smaller search spaces, directly computes gradient density following a brief warm-up period of uniform training. The second is designed for large search spaces, which approximates the density online using a learned set of prototypes. The estimated density is then used for importance sampling by prioritizing architectures with low density and down-sampling architectures in high-density regions. Finally, as computing gradients of large neural networks can be expensive, we show that similar results can be obtained by substituting the gradient with the output layer of a network. We empirically show that architecture distances on this much smaller space are highly correlated with gradient distances, but are cheaper to compute.

The main contributions of this chapter are the following:

- We show that for one-shot NAS, architectures with similar gradients (denser regions) are over-trained and architectures with different gradients (low-density regions) are under-trained. This affects the capability of the sampling method (such as uniform sampling) to approximate the real performance of the NAS architectures.
- We devise a density-aware sampling framework to mitigate this bias, and introduce two algorithms for efficiently estimating the gradient density. We further improve the framework by showing that gradient density can be adequately replaced with functional density based on model outputs, reducing the computational overhead with little loss in performance.
- We validate our model using various search strategies on different datasets and search spaces and show consistent performance improvement compared to other methods that aim to improve fairness or supernet consistency.

In the following sections, we present several strategies and design choices for efficient implementation of density-aware debiasing for supernet training (section 5.3). We then compare our approach with the related work using experiments and analysis on different search spaces and models (section 5.4).

5.2 Related Work

We briefly discuss related work on weight sharing and proposed improvements in previous works. We then focus on single-path sampling methods for supernet training.

5.2.1 Weight sharing in NAS

Early NAS approaches based on reinforcement learning (Zoph & Le, 2016) or evolutionary algorithms (EA) (Real *et al.*, 2019) encountered substantial computational cost to evaluate candidate architectures. To reduce the cost, Pham *et al.* (2018) proposed one-shot NAS based on weight sharing, which has since become one of the most popular NAS techniques.

However, the ranking correlation obtained from the supernet with independent training (ground truth) can be poor (Yu *et al.*, 2019; Zela *et al.*, 2019) due to co-adaptation and interference of optimizing shared weights for different architectures. This interference increases as more weights are shared (Shu, Wang & Cai, 2019). Therefore, one strategy is to reduce weight-sharing among architectures, such as by shrinking the search space (Hu *et al.*, 2020; Chen, Fu & Ling, 2021d; Xia, Xiao, Wang & Zheng, 2022b), or using specialized supernets for different parts of the search space (Zhao *et al.*, 2021a; Hu *et al.*, 2022). In general, methods using weight sharing can be divided into two main approaches:

- One stage methods, such as DARTS (Liu *et al.*, 2018b) that perform supernet training and architecture search simultaneously.
- Two stage methods such as SPOS (Guo *et al.*, 2020b), FairNas (Chu *et al.*, 2021c), and PA&DA (Lu *et al.*, 2023) that perform NAS in two stages: 1) supernet training followed by 2) architecture search (e.g., random, EA, Monte-Carlo tree search methods).

One stage methods allow the training to focus from the beginning on certain parts of the search space, effectively reducing the weight sharing at the cost of introducing bias. Two stage methods ensure more fairness in the training stage and can readily support searching multiple architectures under different constraints (Cai *et al.*, 2019). However, as we discuss in the following sections, they are not entirely bias-free.

In general, the aim of the first stage is to train a reliable and unbiased supernet to apply a search strategy in the second stage to obtain the final architecture. Therefore, the effectiveness of these methods relies on both the quality of the supernet and the efficiency of the search algorithm. Many works focus on improving the search stage (Zhao *et al.*, 2021b; Wang *et al.*, 2021b; Zhang *et al.*, 2024a), our work focuses on improving the supernet training for less bias and better exploration of the search space.

5.2.2 Improving Single-Path supernet Training

Single-path sampling is the most commonly used sampling strategy for supernet training. It adapts a discrete search space and only samples one subnet from the supernet at each iteration, reducing the memory cost to a single subnet as well as helping in decoupling of shared weights (Guo *et al.*, 2020b). The standard and widely used sampling distribution is uniform, since it gives an equal chance of training to all architectures. It can be used either as the first stage supernet training (Guo *et al.*, 2020b; Cai *et al.*, 2019; Chu *et al.*, 2021c; Wang *et al.*, 2021b; Zhao *et al.*, 2021b) in two-stage methods, or as a warm-up for further supernet training (Su *et al.*, 2021a; Roshtkhari *et al.*, 2025b; Roshtkhari, Toews & Pedersoli, 2025a). While uniform sampling is simple and provides a strong baseline, it is not bias-free.

FairNas (Chu *et al.*, 2021c) shows that uniformly sampling architectures causes unfair training opportunities among different operations in the search space. This imbalanced training results in inaccurate architecture estimations and poor rank correlation. They propose to enforce strict fairness to ensure that every operation is updated the same number of times at any point during training. ShiftNas (Zhang *et al.*, 2023) addresses bias in terms of resource allocation to subnets, where most uniformly sampled architectures have intermediate computational costs. They propose to adjust sampling probability based on subnet complexity to accommodate more complex subnets with more resources, while Jeon *et al.* (2025) uses a dynamic complexity-aware learning rate scheduler.

Ning *et al.* (2021) analyzed the gradient dynamics in supernet training, showing both positive and negative effects of gradient interference. Several other works aim to improve training using supernet gradients. Hu *et al.* (2022) and Ly-Manson *et al.* (2024) reduce gradient conflict directly by grouping operations using gradient matching and assigning a separate set of weights to each group. Other works modify the sampling: PA&DA (Lu *et al.*, 2023) propose to increasingly sample subnets/data that minimize gradient variance to increase ranking consistency, Ma *et al.* (2025) propose to project the gradient of a newly sampled architecture to be orthogonal to previously sampled architectures, GreedyNAS (You *et al.*, 2020; Huang *et al.*, 2022) samples high-performing architectures more frequently. However, these methods introduce more bias in sampling toward certain architectures. Our method improves the supernet from a different direction than these methods that directly focus on stability and increased consistency.

Our work focuses on improving the supernet training by reducing the bias from the perspective of density. FairNas is concerned with fairness in training opportunity, and other works consider fairness based on complexity (Zhang *et al.*, 2023; Jeon *et al.*, 2025). We are concerned with the fairness in *effective* training of each architecture. Finally, while not the main objective, our debiasing method can be potentially seen as a form of increasing novelty and better exploration of the search space by prioritizing outliers (low-density areas) during training.

5.3 Method

The supernet training bias is a direct consequence of weight-sharing, independent of the training approach and search space design. While our density-aware debiasing method may work in combination with other training algorithms, for the sake of simplicity and generality, we show its principle by analyzing standard supernet training with uniform sampling.

5.3.1 Uniform Sampling

Uniform sampling is the simplest approach for single-path one-shot (SPOS) NAS. It does not require keeping track of any additional parameters for operations/subnets and can scale well

to very large search spaces. This makes it the default choice for the supernet training stage or as a warm-up to more complex algorithms. Therefore, improving this baseline can directly boost many different NAS methods. Following Guo *et al.* (2020b), in single-path supernet training, at each iteration, one architecture a is uniformly sampled from the search space $\mathcal{A}$. For a given minibatch of training data $\mathcal{X}_{tr} = \{x_0, x_1, ..., x_B\}$ and the corresponding labels $\mathcal{Y}_{tr} = \{y_0, y_1, ..., y_B\}$, the supernet f_a with weights w is trained to minimize the training loss:

$$\sum_{i=0}^{B} \mathcal{L}(f_a(w, x_i), y_i), \quad a \sim \mathcal{U}(0, |\mathcal{A}| - 1). \tag{5.1}$$

After the supernet training stage, to find the final architecture a^*, any search algorithm may be used to efficiently evaluate architectures by inheriting the weights w^* of the trained supernet:

$$a^* = \arg\max_{a \in \mathcal{A}} \; Acc(f_a(w^*, \mathcal{X}_{val})), \tag{5.2}$$

where Acc is the accuracy or any metric used to estimate the quality of the architecture a on the validation data $\mathcal{X}_{val}$.

5.3.2 Biased Sampling

Sampling from a uniform distribution implies equal attention to each subnet; however, due to gradient dynamics of shared weights, the effective optimization among subnets is not equal.

Consider the gradient of the model weights g_a for the architecture a on a training minibatch:

$$g_a = \nabla_w \sum_{i=0}^{B} \mathcal{L}(f_a(w, x_i), y_i), \quad a \sim \mathcal{U}(0, |\mathcal{A}| - 1). \tag{5.3}$$

Assuming for simplicity that all weights w are fully shared across all subnets (for instance, when using convolutional filters and changing the dilations or strides. In our experiments section, we address how to handle when parameters are not fully shared). For two architectures i and j, their

gradient alignment can be defined using cosine similarity:

$$s(g_i, g_j) = \frac{g_i \cdot g_j}{||g_i||_2 ||g_j||_2}. \tag{5.4}$$

By definition, if the gradients of two architectures align ($s(g_i, g_j) \approx 1$), a training step for i will have a positive transferring effect on j, effectively jointly optimizing for j, even though j is not directly sampled. Considering this communication among gradients, the only way that every architecture would receive equal effective updates on the weights would be with mutually orthogonal gradients ($s(g_i, g_j) = 0$). However, this implies independent training of subnets, which goes against the core premise of weight sharing, in which multiple architectures leverage their similar gradient updates to accelerate training and learn a set of generalized weights. In contrast, in our proposed approach, we still train with shared weights, but we estimate the effective gradient update and compensate for that to make the training fairer.

5.3.3 Bias Estimation Based on Gradient Density

To reduce the bias in supernet training, we need to estimate if a given architecture is generally over-trained or under-trained. To globally assess an architecture, we estimate the density of its gradients. We consider a dense region as where there are many architectures with similar gradients, with corresponding architectures over-trained. In lower-density regions, few architectures have similar gradients, and they are under-trained.

Density estimation: A straightforward and intuitive way to measure the density of a given architecture a, is then to use its mean distance to other architectures j in the search space:

$$\bar{d_a} = \frac{1}{|\mathcal{A}| - 1} \sum_{j \in \mathcal{A}, j \neq a} d(a, j), \tag{5.5}$$

where $d(a, j)$ is a suitable distance metric, such as cosine distance of gradients $d(a, j) = 1 - s(g_a, g_j)$ calculated on the same batch of data. Density α_a is then estimated as $\alpha_a = \frac{1}{\bar{d_a}}$.

5.3.4 Debiased Sampling for Numerable Architectures

In this section, we consider the case of a search space with a known and numerable set of architectures. To reduce the bias in supernet training, we need to account for the density distribution in the search space. The density-aware sampling rate of architecture a should be inversely proportional to its estimated density α_a. The debiased training is therefore:

$$\sum_i \mathcal{L}(f_a(w, x_i), y_i), \quad a \sim \mathcal{M}_\tau(\bar{d}), \tag{5.6}$$

with $\mathcal{M}_\tau$ the density-aware probability distribution and its sharpness controlled by the temperature parameter τ. For sufficiently small search spaces (depending on available computational resources), it is possible to estimate α_a for every architecture. Therefore, we define $\mathcal{M}$ directly using categorical distribution:

$$\mathcal{M}_\tau(a) = \frac{\exp(\bar{d}_a/\tau)}{\sum_{i \in A} \exp(\bar{d}_i/\tau)}. \tag{5.7}$$

Algorithm 5.1 outlines the debiased supernet training process.

5.3.5 Debiased Sampling for Larger Search Spaces

For some search spaces, it is feasible to estimate the density of each architecture and store it in memory, leading to a fine-grained probability distribution $\mathcal{M}(\alpha)$. However, in very large search spaces, the entire search space cannot be covered. Furthermore, in some search spaces, as the supernet is trained, the density distribution of the search space may evolve. To account for these, we propose an online density approximation strategy coupled with supernet training.

Density Prototypes: To avoid comparing each architecture with every other one, we define a small, fixed set of representative prototypes $C = \{c_1, c_2, ..., c_K\}$. These prototypes are designed to capture the overall structure of the search space. The set can be initialized by running a K-Means clustering algorithm on an adequately large, random sample of architecture

Algorithm 5.1 Supernet training with density-aware sampling for small search spaces

Algorithm: Density-aware sampling for small search spaces
Input: $\mathcal{A}$: search space, $f_a(w)$: supernet with sampled subnet a, $iter_w$/$iter_t$: warm-up/total iterations, d: distance metric, τ: softmax temperature
# warm-up
while $iter < iter_w$ **do**
    sample $a \sim \mathcal{U}(0, |\mathcal{A}| - 1))$
    train $f_a(w)$
end while
# density calculation
for $i \in \mathcal{A}$ **do**
    $\bar{d}_a = \frac{1}{|\mathcal{A}|-1} \sum_{j \in \mathcal{A}, j \neq a} d(a, j)$
end for
# probability distribution estimation
$\mathcal{M}_\tau(a) = \exp(\bar{d}_a/\tau)/\sum_{i \in A} \exp(\bar{d}_i/\tau)$
# train with debiased sampling
while $iter < iter_t$ **do**
    sample $a \sim \mathcal{M}_\tau$
    train $f_a(w)$
end while
Output: trained supernet $f(w^*)$

representations (e.g., gradients) to obtain centroids and form C. The density of each architecture is then approximated by measuring its relationship only to those prototypes, by replacing $\mathcal{A}$ with C in equation 5.5. This reduces the density computational cost from $\propto |\mathcal{A}|$ to $\propto K$.

Updating Prototypes: The prototypes are continuously updated throughout training, to adapt to more samples becoming available and changing representations as supernet weights are optimized. Given an architecture a, sampled during training, C is updated by a soft-assignment rule. The normalized assignment for prototype c_k update is determined by:

$$\beta_k = \frac{d^2(a, c_k)}{\sum_{j=0}^{K-1} d^2(a, c_j)}, \ \forall k \in [0, K-1]. \tag{5.8}$$

Each prototype c_k is then updated using an exponential moving average with λ controlling the weight of the new sample as:

$$c_k \leftarrow c_k + \lambda \cdot \beta_k \cdot (a - c_k),\ \forall c_k \in \mathcal{C}. \tag{5.9}$$

supernet Training: To achieve a debiased training of the supernet, the contribution of an architecture should be scaled inversely to its density:

$$P_{train}(a) = \text{sigmoid}\left(\frac{\alpha_{\text{mid}} - \alpha_a}{\tau}\right). \tag{5.10}$$

$P_{train}(a)$ can be used to make probabilistic decision to accept or reject a sample for training. Given enough iterations, this sampling method will approximate the density-aware sampling distribution. We show the supernet training pipeline in algorithm 5.2.

5.3.6 Density Estimation with Functional Similarity

As discussed, a suitable distance metric $d(\cdot, \cdot)$ for equation 5.5 is the cosine distance of gradients, which provides the most direct measure of bias in supernet training.

However, they can only be reliably calculated on shared weights and require costly backpropagation. Therefore, we seek a cheaper alternative representation for density estimations. We use output features as the summary of architectures and calculate functional distance (Klabunde *et al.*, 2025) as:

$$d(a_i, a_j) = 1 - \frac{f(a_i) \cdot f(a_j)}{||f(a_i)||||f(a_j)||} \tag{5.11}$$

In figure 5.2, we show the calculated gradient and functional density from equation 5.5 for the Pooling search space. The correlation implies that functional density can be used as an

Algorithm 5.2 Debiased supernet training with density-aware sampling for large search spaces

Algorithm: Density-aware sampling for larger search spaces
Input: $\mathcal{A}$: search space, $f_a(w)$: supernet with sampled subnet a, $iter_w$/$iter_t$: warm-up/total iterations, d: distance metric, τ: softmax temperature, x_v: validation batch, K: number of prototypes, N: initial sample size
# warm-up
while $iter < iter_w$ **do**
    sample $a \sim \mathcal{U}(0, |\mathcal{A}| - 1))$
    train $f_a(w)$
end while
# prototype initialization
sample $\{a_1, ..., a_N\} \sim \mathcal{U}(0, |\mathcal{A}| - 1)$
$C \leftarrow$ K-Means clustering $\{a_1, ..., a_N\}$ on x_v
# debiased training
while $iter < iter_t$ **do**
    sample $a \sim \mathcal{U}(0, |\mathcal{A}| - 1))$
    $\bar{d}_a = \frac{1}{K-1} \sum_{j \in C} d(a, j)$
    $\alpha_a \leftarrow 1/\bar{d}_a$
    # supernet training
    train $f(a)$ with $sigmoid\left(\frac{\alpha_{\text{mid}} - \alpha_a}{\tau}\right)$
    # prototype update
    $\beta_k = d^2(a, c_k) / \sum_{j=0}^{K-1} d^2(a, c_j)$
    $c_k \leftarrow c_k + \lambda \cdot \beta_k \cdot (a - c_k)$
end while
Output: trained supernet $f(w^*)$

alternative to gradient density. On NAS-Bench-Macro, another benchmark, our estimation shows a strong correlation (~ 0.9) for these values. We use functional density as the default for our experiments and explore other representations for density estimation in the experiments section.

5.4 Experiments

For numerable search spaces, we evaluate our method (algorithm 5.1) on two benchmarks for CIFAR10 (Krizhevsky *et al.*, 2009): Pooling benchmark (Roshtkhari *et al.*, 2023), and NAS-Bench-Macro (Su *et al.*, 2021a). For larger search space (algorithm 5.2), we first use our

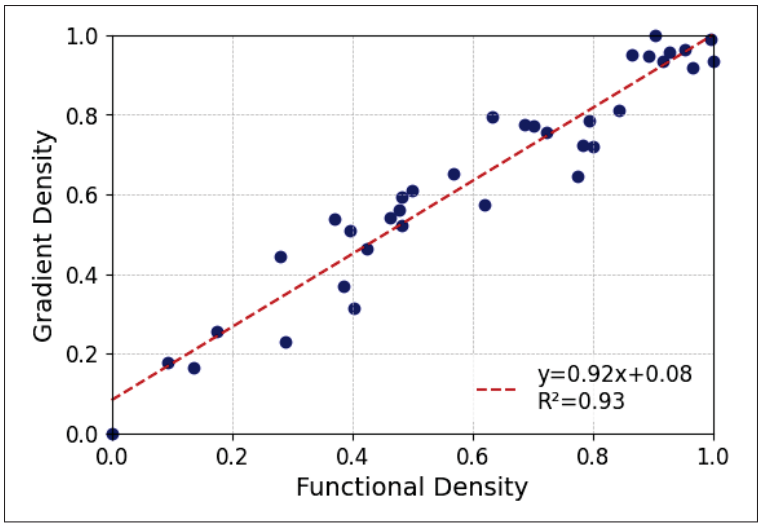


Figure 5.2 Gradient density vs. functional density for Pooling benchmark. Density estimated from gradients shows a strong correlation with functional density from outputs

method for NAS-Bench-201 (Dong & Yang, 2020) and then expand to MobileNet-V2 (Sandler *et al.*, 2018) search space on ImageNet (Deng *et al.*, 2009). For each search space, we compare our method with various combinations of sampling methods during supernet training (stage 1), and different search methods (stage 2) after training. For more information on search spaces, search methods, and experimental details, see appendix II.

5.4.1 Experiments on Numerable Search Spaces

Search Space: Pooling benchmark is a Resnet-based (He *et al.*, 2016a) macro search space where only downsampling (pooling) is optimized. Weights are fully shared among architectures, making one-shot NAS very challenging on this search space (Roshtkhari *et al.*, 2023), and suitable for investigating weight sharing mechanism. NAS-Bench-Macro is another benchmark that optimizes macro structure by searching among 3 candidate operations for 8 layers, resulting in a total size of $3^8 = 6,561$.

Supernet Training: We compare debiased supernet training (our method) with uniform training and training with strict fairness (FairNas). In table 5.1, we show the results of top-k architectures

Table 5.1 CIFAR10 results on (top) Pooling search space(bottom) NAS-Bench-Macro (Su *et al.*, 2021a). We report the average performance of top-k architectures, final architecture accuracy found by BSE, and rank correlation with ground truth (Kendall's Tau) for different supernet sampling methods

Training Method	Top-1	Top-5	Top-10	BSE	KT
Pooling Benchmark					
Default	90.52	-	-	-	-
Uniform	90.51	90.46	90.54	90.42	0.16
Strict fairness	91.05	90.63	90.31	91.70	0.21
Debiased (ours)	91.83	90.92	90.59	92.01	0.34
Best	92.01	-	-	-	-
NAS-Bench-Macro					
Uniform	92.03	91.72	91.52	92.09	0.72
Strict fairness	92.47	91.45	91.22	92.51	0.72
Debiased (ours)	92.56	92.02	91.64	93.13	0.74
Best	93.13	-	-	-	-

found after supernet training. Due to the search space size, the cost of density estimation for our method is negligible, and the cost of all compared models is similar. We report rank correlation with ground truth using Kendall's Tau. In all cases, our method shows a clear improvement. To obtain the final architectures, we apply Boltzmann Softmax Exploration (BSE) (Sutton, Barto *et al.*, 1998) to the trained supernet as the search method. Using BSE, only our method is able to output the best architecture in the search space as the final result for both benchmarks.

In figure 5.3, we compare the accuracy of architectures when trained independently (ground truth) with the accuracy obtained for supernet training with uniform and debiased sampling. Uniform sampling shows bias in the evaluation of high-density architectures, which is not consistent with ground truth. Debiased sampling mitigates this bias, improves overall accuracy in low-density regions, and crucially increases the ranking of top-performing architectures.

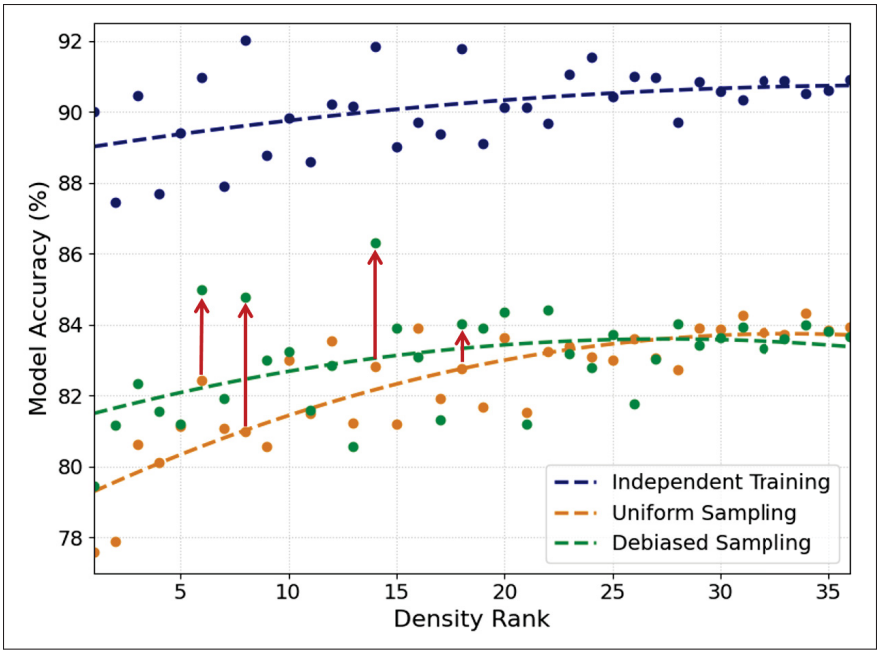


Figure 5.3 NAS accuracy on Pooling benchmark for models trained independently (blue), a uniformly trained supernet (orange), and our debiased training (green). Models are sorted from low to high density. Uniform sampling shows higher correlation with density. Our approach removes that density bias and shows a more similar correlation to the independent training, which we aim to approximate. Debiased training shows improvement, especially for top-ranking architectures (red arrows)

5.4.2 Ablations

Alternative representations for density estimation: We proposed to use functional (output) density as an alternative to gradient density. In table 5.2, we compare different architecture representations and distance metric $d(\cdot, \cdot)$ for density estimation in equation 5.5.

We directly use the gradients for the Pooling benchmark. For NAS-Bench-Macro, in which weights are not fully shared, we consider gradients as orthogonal $d(i, j) = 1$ when layers i and j do not share weights. From table 5.2, we observe that gradient density and functional density perform similarly.

Table 5.2 CIFAR10 results on (top) Pooling search space (bottom) NAS-Bench-Macro (Su *et al.*, 2021a). We compare using average gradients of layers, feature maps, and encodings to estimate density. We report the average and the final architecture accuracy found by Boltzmann search

Debiasing Method	Top-1	Top-5	BSE
Pooling Benchmark			
Gradient	91.54	91.37 ± 0.21	91.77
Representational	90.96	90.81 ± 0.31	91.04
Encoding	90.52	90.61 ± 0.27	91.01
Functional (ours)	91.83	91.59 ± 0.24	92.01
NAS-Bench-Macro			
Gradient	92.68	92.58 ± 0.11	93.05
Representational	92.39	92.32 ± 0.09	92.90
Encoding	91.95	90.59 ± 0.05	92.02
Functional (ours)	92.56	92.21 ± 0.16	93.13

We also explore density estimation from a representational similarity/distance measure. The representational similarity is complementary to functional similarity and compares the pattern of activations in hidden layers of the feature maps for a given input (Klabunde *et al.*, 2025). Since comparing large feature maps for all layers is costly, we only apply Centered Kernel Alignment (CKA) (Kornblith, Norouzi, Lee & Hinton, 2019) to a selected number of layers to estimate density. Functional distances achieve great performance with a lower cost than both. We also compare with the cheapest representations, the encoding using Hamming distances. For more details about these representations/distances, see appendix II.

Density during training: In figure 5.4, we show density variation during supernet training. At initialization, shared weights are initialized randomly from the same distribution, resulting in indistinguishable architectures. After only a few epochs of training, density estimations mostly stabilize, meaning that a short warm-up for density estimation is feasible.

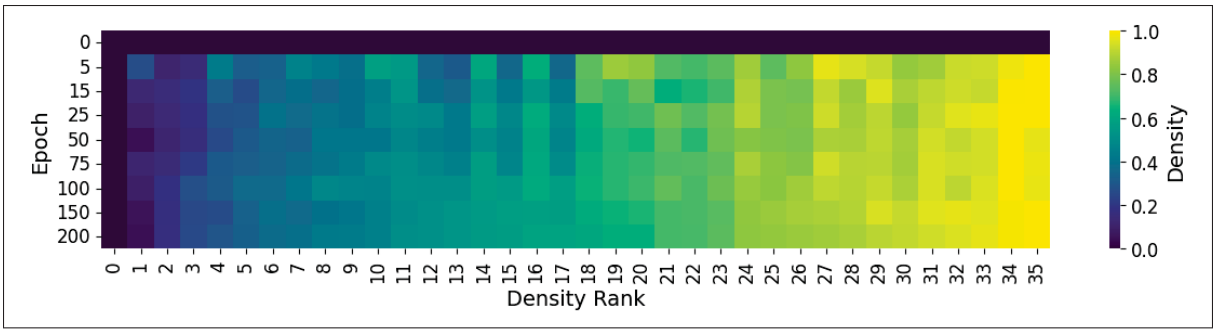


Figure 5.4 Density variations during training. Density estimations during supernet training stabilize after a few training epochs

Debiasing temperature: Temperature (τ) in equation 5.7 controls the strength of debiasing with $\tau \to \infty$ corresponding to uniform sampling. In figure 5.5, we compare various temperatures with uniform sampling. Very low τ enforces debiasing strongly and under-trains high-density architectures significantly, which introduces instability and lowers the overall performance of the supernet.

5.4.3 Experiments on Large Search Spaces

To validate our method for large search spaces (algorithm 5.2), we performed experiments on NAS-Bench-201 and the MobileNet search space on ImageNet.

NAS-Bench-201 experiments: NAS-Bench-201 (Dong & Yang, 2020) is a tabular cell-based benchmark based on a DARTS-like (Liu *et al.*, 2018b) search space. It searches for 4 nodes with 5 possible operations in a cell, totaling $15,625$ architectures. In table 5.3 we report top-1 accuracy and Kendall's Tau for CIFAR10, CIAFR100, and ImageNet16-120 datasets. Our method shows improvement in terms of both accuracy and rank correlation.

Number of Prototypes: To initialize prototypes, we need to apply K-Means clustering to a population of N samples. We note that uniformly sampling architectures draws more samples from dense regions of the search space. Sparse regions are then less likely to be assigned their own prototypes by the subsequent clustering. While the prototype set size K can be very large

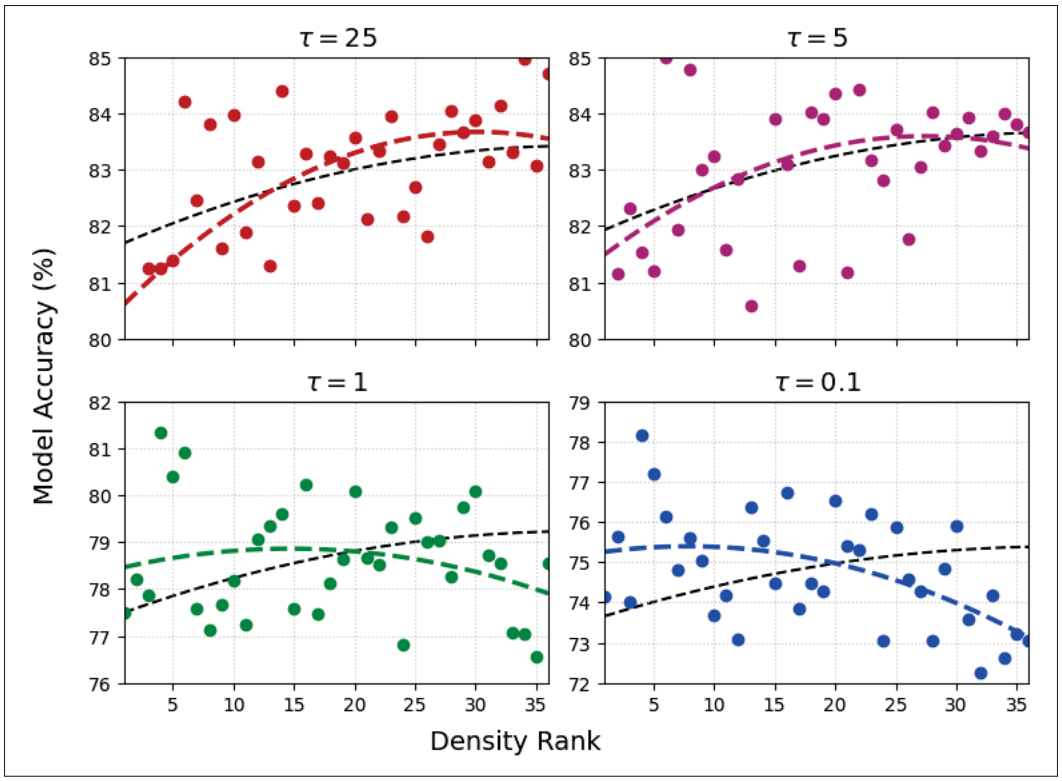


Figure 5.5 Supernet evaluation with different debiasing temperatures. Independent training (ground truth) slope is shown with a black line as a reference. At a high temperature $\tau = 25$, the supernet behaves similarly to uniform sampling. Decreasing the temperature to $\tau = 5$ improves the correlation with ground truth. When $\tau < 1$, density-aware debiasing is over-enforced, resulting in a strong bias towards lower density architectures

Table 5.3 CIFAR10, CIFAR100 and ImageNet16-120 results for NAS-Bench-201 (Dong & Yang, 2020). We compare Top-1 and Kendall's Tau for various sampling methods

Training	**CIFAR10**		**CIFAR100**		**ImageNet16-120**		**GPU**
Method	Top-1 Acc.	Kendall's τ	Top-1 Acc.	Kendall's τ	Top-1 Acc.	Kendall's τ	(hours)
Uniform	84.02 ± 2.31	0.42	52.31 ± 3.49	0.39	28.14 ± 6.15	0.49	2.59
FairNas	87.33 ± 0.89	0.53	54.07 ± 2.45	0.49	37.03 ± 1.96	0.51	2.64
Debiased (ours)	91.63 ± 1.21	0.59	66.48 ± 2.08	0.58	42.71 ± 0.79	0.63	2.70
Best	94.37	-	73.51	-	47.31	-	-

for distance calculation in equation 5.11, it is effectively limited by computational budget to calculate N. To measure how density estimations using a prototype model the fine-grained

accuracy, we report the correlation of density estimation using K prototypes with fine-grained estimation in table 5.4.

Table 5.4 Accuracy and rank correlation for NAS-Bench-201 for various K

K	Top-1 Acc.	Pearson
5	54.12 ± 5.16	0.21
50	86.71 ± 4.36	0.42
100	91.63 ± 1.21	0.86
200	91.52 ± 1.24	0.88

MobileNet Search Space: The search space consists of 21 layers of mobile inverted bottleneck convolutions (MBConv) with kernel sizes $(3, 5, 7)$ and expansion ratio of $(3, 6)$ and choice of skip connection, resulting in a size of 12^{21}. To accelerate training on ImageNet, we utilize mixed-precision and FFCV (Leclerc *et al.*, 2023) library.

For comparison with supernet sampling methods during training (stage 1), we consider the following sampling methods: **1)** default uniform sampling (original SPOS), **2)** FairNas (sampling with fairness), **3)** PA&DA (sampling probability adjustments based on gradient norm of path and data).

To find the final architecture (stage 2), we consider the following search methods:

- **Random search:** Not using supernet. Randomly select ten architectures as a baseline and report the best.
- **One-shot:** Use random search on trained supernet and select the best architecture based on supernet evaluations from 50 candidates.
- **EA with shifting:** Use evolutionary algorithm with supernet shifting (Zhang *et al.*, 2024a) for architecture search on the trained supernet.

In table 5.5, we show that under the same setting, training the supernet with density-aware debiasing increases the overall performance of the found architectures with both search methods.

Table 5.5 ImageNet results on MobileNet search space. We compare with 3 other sampling methods. We apply one-shot and EA with shifting on trained supernet as search algorithm to obtain the final results

Training Method	Top-1	Top-5	FLOPs (M)	Params (M)
MobileNetV2	72.0	91.0	300	3.4
Random search	70.27	88.1	300	4.1
One-shot				
Uniform	73.08	91.8	312	3.2
FairNas	73.14	91.9	320	4.0
PA&DA	73.41	92.0	340	4.8
Debiased (ours)	74.75	92.2	282	4.2
EA with shifting				
Uniform	73.51	92.0	336	4.8
FairNas	73.65	92.0	324	3.4
PA&DA	73.84	92.1	388	4.9
Debiased (ours)	75.10	92.5	332	4.1

5.5 Conclusion

In this chapter, we have addressed the inherent training bias in one-shot NAS, where commonly used methods (e.g., uniform sampling) over-train architectures in dense regions of the representation space, while under-training those in sparse ones, compromising the supernet training process. We have introduced a flexible density-aware sampling framework that reduces this bias by sampling architectures inversely proportional to their estimated density. With practical algorithms for both numerable and large-scale search spaces, our method consistently improves supernet training across diverse benchmarks, leading to the discovery of superior final architectures. Furthermore, our approach can be easily combined with various search methods such as evolutionary algorithms. This work demonstrates the critical importance of correcting for sampling bias and provides a robust, general-purpose alternative to sampling methods such as uniform for NAS pipelines.

CONCLUSION AND RECOMMENDATIONS

This thesis has explored several fundamental challenges in one-shot NAS, specifically focusing on the issues of weight-sharing, effective search space exploration, weight co-adaptation, and supernet training bias. While one-shot methods significantly reduce the computational burden of NAS by utilizing a shared-weight supernet, the entanglement of different architectural paths often leads to poor performance estimation and unreliable ranking of candidate architectures. Through four complementary contributions - Balanced mixture of supernets, NAS by hierarchical search space design, iterative MCTS for NAS, and density-aware sampling - we demonstrated several approaches to improve one-shot NAS for computer vision applications.

Each contribution addresses one-shot NAS from a different angle. Balanced mixture of supernets directly reduces the weight sharing to improve rank consistency by learning a mixture of multiple supernets. NAS by learning a hierarchical search space and iterative MCTS, both improve MCTS for NAS. In the former, we proposed an unsupervised method to learn hierarchical search space design, while the latter uses a supervised approach with iterative refinement during training. Finally, density-aware sampling was proposed as an alternative to the standard uniform sampling strategy for NAS, to promote less bias and increase fairness in optimization of different architectures.

Together, these contributions provide additional insight and understanding of one-shot NAS method and particularly the supernet training and weight-sharing mechanism. Aligned with the contributions of this thesis, several avenues of future research remain open:

- **Zero-Cost Hierarchy Construction:** Recently, zero-cost proxies have gained popularity due to their high efficiency. As a future direction, one can explore utilizing them to build the hierarchical search space, removing the cost of supernet training and the overall search time.
- **Combinations with Density-Aware Sampling:** While our main comparison was with the standard uniform sampling, several other sampling strategies (greedy sampling methods,

gradient variance reduction methods, etc.) can be potentially combined with density-aware sampling to improve architecture sampling.

- **Hardware-Aware Multi-Objective Search:** Extending hierarchical MCTS and density-aware sampling to incorporate hardware constraints (e.g., latency, memory, etc.) would increase the practicality for deployment on edge devices.
- **Cross-Domain Transferability:** Investigation of how the learned hierarchy or the specialized supernets (mixtures) can be potentially transferred across domains.

Automating the design of optimal neural architectures is essential for the continued expansion of AI into diverse and resource-constrained applications. By addressing the core limitations of one-shot NAS, this thesis provides a set of tools and methodologies that enhance the reliability and efficiency of automated design. The move toward semantically-aware and gradient-balanced search processes represents a vital step in making NAS a more robust and accessible technology for the computer vision community.

APPENDIX I

SUPPLEMENTARY MATERIAL FOR CHAPTER 3

1. Experimental Setup and Details

Sampling method details: A summary of various methods used in our experiments (Tables 3.1 and 3.3) is presented in Table I-1.

Table-A I-1 Summary of sampling methods used in our experiments

Method	Search Space Structure	Sampling Method
Uniform	Flat	Uniform sampling of architectures
Independent	Flat	Nodes sampled independently
Boltzmann	Flat	Joint sampling of architectures
Mixture	Flat (partitioned)	Uniform with multiple models (Roshtkhari *et al.*, 2023)
MCTS	Hierarchical (def. tree)	Conditional prob. (section 3.3, Tree Search)
MCTS + Reg.	Hierarchical (def. tree)	Conditional prob. + regularization (Su *et al.*, 2021a)
MCTS + Learned	Hierarchical (learned tree)	Conditional prob. with learned tree (section 3.4)

1.1 Dataset and Hyperparameters

For experiments performed on CIFAR10 (Kocsis & Szepesvári, 2006) dataset, we split the training set 50/50 for NAS training and validation. To tune hyperparameters, we either performed grid search or, when comparing with other works, used similar hyperparameters. We used SGD with weight decay and a cosine annealing learning rate schedule. Furthermore, for MCTS methods, we split training iterations to roughly 40/25/35 fractions for uniform sampling/MCTS warm-up/MCTS sampling, respectively. In all experiments, we use $\beta = 0.95$ and $\lambda = 0.5$.

Pooling search space: This search space is based on Resnet20 (He *et al.*, 2016a) architecture. The only CNN parameters to search are where to perform pooling. To calculate the distance matrix, we trained the supernet for 300 epochs using uniform sampling with batch size 512, learning rate 0.1, and weight decay 1e-3. For search, we trained for 400 epochs with batch size 256, learning rate 0.05, and temperature T is set to a linear annealing schedule (0.02, 0.0025).

Since this search space is small, we only consider nodes with the maximum probabilities and report them as the final architecture.

NAS-Bench-Macro search space: This benchmark introduced by (Su *et al.*, 2021a) is based on MobileNetV2 (Sandler *et al.*, 2018) blocks. The supernet warm-up is performed for 80 epochs with a batch size of 512 and a learning rate of 0.05. For search, we use a batch size of 256 for 120 epochs with T linearly annealing from 0.01. At the end of training, we sample 50 architectures from the tree and report the best as the final architecture.

Mobilenet Search Space for ImageNet: To accelerate our training in ImageNet experiments, we use mixed precision and FFCV (Leclerc *et al.*, 2023) library. We sample architectures within a FLOPs budget and discard those outside of it. We train for 100 epochs with SGD and cosine annealing learning rate. Other training strategies are similar to experiments on CIFAR10.

2. Additional Results

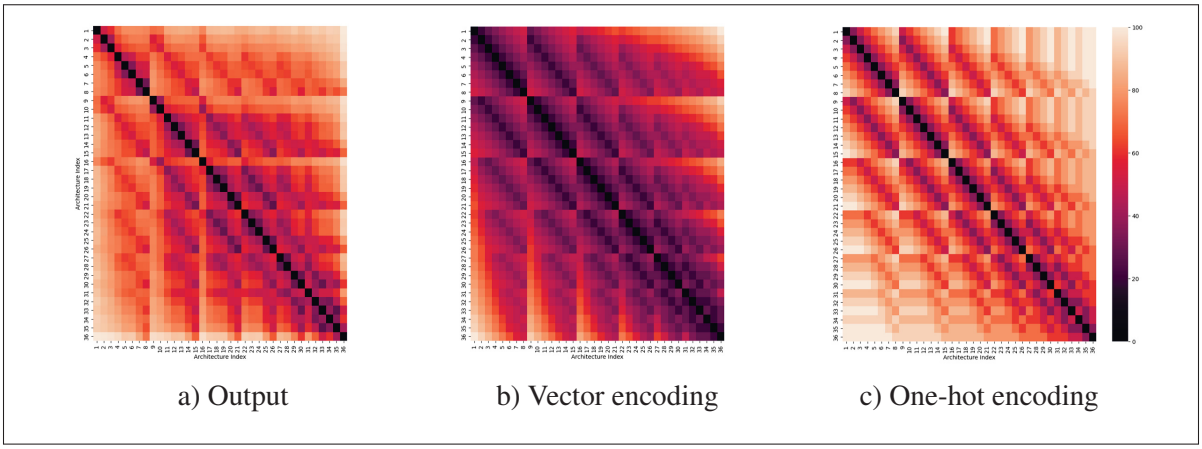
We present additional experimental results, ablations, and visualizations on Pooling search space.

2.1 Reward Ablation for MCTS

The most common rewards used for NAS algorithms are accuracy and loss. While loss is differentiable, accuracy is more aligned with the objective of NAS. Furthermore, either the training or validation can be used to calculate the reward. Instead of absolute values, a relative training loss metric was used in (Su *et al.*, 2021a) to account for unfair reward comparison at different iterations of supernet training. In Table I-2 for NAS-Bench-Macro, we explore some common combinations of options to estimate the reward. In all settings, our approach performs on par or slightly better than Default Tree + Regularization. For both methods, it seems that using the accuracy on the validation set as a metric is the best. However, while for our approach the best performing configuration is obtained with the absolute metric, for Su *et al.* (2021a) the relative metric seems slightly better.

Table-A I-2 CIFAR10 results on NAS-Bench-Macro (Su *et al.*, 2021a) search space with several various rewards. Relative rewards are calculated according to Su *et al.* (2021a). The rewards can be can be calculated on either training or validation set

Search Structure	Metric	Reward Data	Reward Measure	Arch.	Best Acc.	Best Rank	Avg. Rank
Default Tree + Reg	rel.	train	loss	[22121222]	92.74	85	97
Learned Tree (ours)	rel.	train	loss	[22122220]	92.78	61	67
Default Tree + Reg	abs.	train	acc.	[22121210]	92.55	227	278
Learned Tree (ours)	abs.	train	acc.	[22110222]	92.56	209	301
Default Tree + Reg	rel.	val.	loss	[22222022]	92.71	98	120
Learned Tree (ours)	rel.	val.	loss	[21211220]	92.76	71	95
Default Tree + Reg	rel.	val.	acc.	[12222222]	92.92	21	112
Learned Tree (ours)	rel.	val.	acc.	[22212200]	92.94	19	67
Default Tree + Reg	abs.	val.	acc.	[22221200]	92.86	34	54
Learned Tree (ours)	abs.	val.	acc.	[22212220]	93.13	1	6

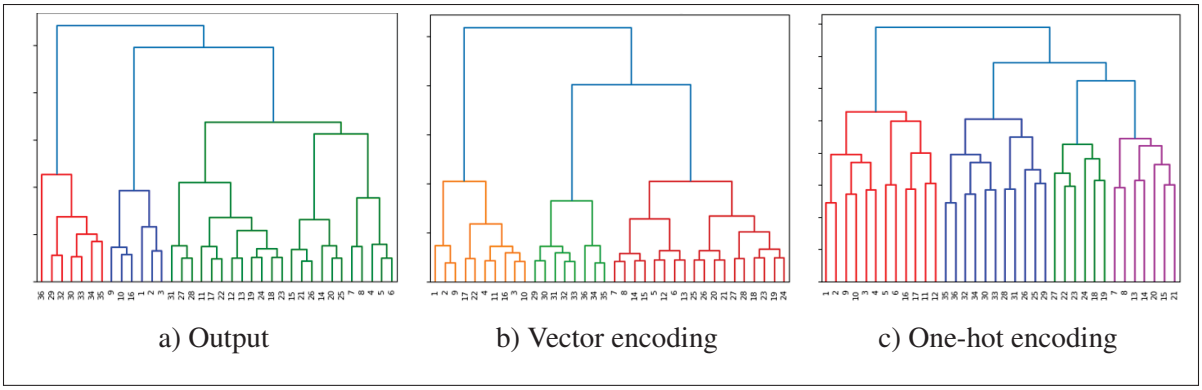


a) Output b) Vector encoding c) One-hot encoding

Figure-A I-1 Normalized distance matrices calculatedthe with various methods.(a) Distance matrix calculated from output vectors (our method); (b) From vector encoding; (c) From one-hot encoding. The architecture indices on leaves correspond to indices used in Pooling benchmark

2.2 Distance Matrices

We visualize the distance matrices calculated using the output vector and various encodings in Figure I-1.



a) Output b) Vector encoding c) One-hot encoding

Figure-A I-2 Tree branching for Pooling search space by hierarchical clustering. The architecture indices on leaves correspond to indices used in Pooling benchmark (Roshtkhari *et al.*, 2023). (a) Tree learned from output vectors (our method) (b) From vector encoding (c) From one-hot encoding

2.3 Tree Visualizations

For pooling search space, we visualize tree structure based on our proposed method and architecture encodings in Figure I-2. The tree is presented with architecture indices on leaves. The architecture indices and the corresponding performance can be found in Roshtkhari *et al.* (2023).

APPENDIX II

SUPPLEMENTARY MATERIAL FOR CHAPTER 5

1. Search Spaces

Pooling Benchmark This search space introduced by Roshtkhari *et al.* (2023) is built using Resnet (He *et al.*, 2016a) blocks. The search involves changing the position of the pooling layers within this architecture. It considers architectures with 2 pooling operations with 9 available locations within Resnet20 architecture. Other architecture components, such as channels, types of convolutions, etc., are fixed.

NAS-Bench-Macro This benchmark presented by Su *et al.* (2021a) is based on MobileNetV2 (Sandler *et al.*, 2018) blocks. It offers 3 operational choices $\{ID, MB3\_K3, MB6\_K5\}$ for 8 layers, resulting in 6,561 architectures.

NAS-Bench-201 NAS-Bench-201 (Dong & Yang, 2020) is a cell-based search space. Each cell contains 4 nodes with 5 possible operations per node, resulting in $15,626$ cells/architectures in total. The macro skeleton of the network is fixed, and the cell structure is repeated to form the architecture. The structure has a 3×3 convolution with 16 channels as the stem. The architecture has three stacks of cells, each containing 5 cells with 16, 32, and 64 channels. Between each stack, a basic residual block (He *et al.*, 2016a) of stride 2 is used to halve the feature map size.

MobileNet Search Space The search space used for the ImageNet experiments is a macro-structure based on the MobileNetV2 architecture (table II-1). This design choice was made for fair comparison with other one-shot NAS methods. The architecture consists of a supernet with 21 searchable blocks, where each block is a MobileNetV2 inverted bottleneck. For each of these 21 blocks, one operation must be selected from a set of 13 candidates. The combination of these choices results in a total search space size of 13^{21} possible architectures. The full list of operation candidates is detailed in Table II-2.

Table-A II-1 The macro-structure of MobileNetV2 search space. Two stem layers are followed by stacks of choice operations (Op.) from table II-2. "No." determines the number of stacked layers and "Input" determines input feature map size

No.	Op.	Input	Channels	Stride
1	3×3 conv	224	32	2
1	MB1_K3	112	16	1
4	choice	112	32	2
4	choice	56	40	2
4	choice	28	80	2
4	choice	14	96	1
4	choice	14	192	2
1	choice	7	320	1
1	1×1 conv	7	1280	1
1	avg. pool	7	-	-
1	fc	-	-	-

Table-A II-2 The operation choices for the MobileNetV2 search space. ID is identity mapping. Operations have the option for SE modules

Block Type	Exp. Ratio	Kernel Size	SE
MB1_K3	1	3	-
ID	-	-	-
MB3_K3	3	3	yes
MB3_K5	3	5	yes
MB3_K7	3	7	yes
MB6_K3	6	3	yes
MB6_K5	6	5	yes
MB6_K7	6	7	yes

2. Implementation Details

For pooling search space (Roshtkhari *et al.*, 2023) we used the publicly available code. We used a batch size of 512 and trained for 300 epochs. We used 50 epochs for warm-up for uniform

sampling. For the rest of the hyperparameters, we use the same setting as the original paper: we used a learning rate of 0.1 with cosine annealing and weight decay 1e-3.

For NAS-Bench-Macro, we implemented one-shot NAS based on the benchmark from the original paper (Su *et al.*, 2021a). We followed similar hyperparameter settings for our training: training for 20/50 epochs for warm-up/training with batch size 512 and SGD with an initial learning rate of 0.1.

For ImageNet experiments, we performed our experiments on an A100 GPU. For EA with shift (Zhang *et al.*, 2024a), PA&DA (Lu *et al.*, 2023), and FairNas (Chu *et al.*, 2021c) methods, we used the available implementations of the papers. We used FFCV (Leclerc *et al.*, 2023) and mixed-precision in our runs to accelerate the experiments. We use a batch size of 512, the model is trained using an SGD optimizer with 0.9 momentum. A cosine annealing strategy is used with an initial learning rate of 0.1, which decays over 120 epochs.

3. More Details on Architectures Similarity/Distances

More Related Work: Distance (or similarity) of architectures has been used for NAS, most notably for EA NAS methods. They are used mainly to maintain diversity (exploration), and/or guide the search towards promising candidates (exploitation) or a combination of both. These objectives are complementary in how they promote sampling, as increasing diversity requires maximizing a distance measure, while guiding the search requires sampling similar to a desirable architecture.

Distances have been used to maintain population diversity in EA (Zhang, Li, Pan, Liu & Su, 2020b; Sinha & Chen, 2022) in early stages and avoid premature convergence to suboptimal solutions. Furthermore, similarity measures are often used to guide the search towards local regions with promising architectures (Chen *et al.*, 2024).

To calculate similarity, the architectures are often represented as strings with distances calculated based on edit distance. This is the computationally cheapest method; however, it generally

ignores the relationship among operations/layers (Chen *et al.*, 2024), giving the same importance to vastly different operational changes (particularly when skip connections are present), and does not use any knowledge gained from training.

Functional and representational similarity are two complementary measures to quantify neural network similarity (Klabunde *et al.*, 2025). Functional (semantic) similarity is based on output vectors has been used for clustering architectures (Roshtkhari *et al.*, 2025b), and for selecting parent individuals in EA (Xue, Zha, Wahib, Ouyang & Wang, 2024) in NAS.

Representation (activation) based similarity compares the pattern of activations in hidden-layer feature maps for a given input. Measures such as Centered Kernel Alignment (CKA) (Kornblith *et al.*, 2019) have been used to guide training of individual architectures towards a well-performing reference architecture by adding the term to the objective loss function (Zheng *et al.*, 2022).

Representational distances: Centered Kernel Alignment (CKA) (Kornblith *et al.*, 2019) is a technique used to measure the similarity between the learned representations of two neural network layers. Considering centered samples X_1 and X_2, the normalized Hilbert-Schmidt Independence Criterion (HSIC) (Gretton, Bousquet, Smola & Schölkopf, 2005; Zheng *et al.*, 2025) is defined as:

$$CKA(X_1, X_2) = \frac{||X_2^T X_1||_F^2}{||X_1^T X_1||_F ||X_2^T X_2||_F}, \tag{A II-1}$$

where $||\cdot||_F$ is the Frobenius norm. We use this similarity to estimate representational distances.

Encoding distances: A very common encoding used is vector or string encoding. For an architecture a with L layers is represented as vector $v_a = [a^{(1)}, a^{(2)}, ..., a^{(L)}]$. Each a^i denotes an operation (e.g. convolution, pooling). To calculate distances, we used $d(i, j) = \sum_{k=1}^{L} \mathbb{I}\{v_i^{(k)} \neq v_j^{(k)}\}$, with function $\mathbb{I}$ counting the number of different items in two architectures.

4. Additional Results

In this section, we provide additional experimental results, ablations, and analysis.

Table-A II-3 Comparison of correlation coefficients for different sampling strategies

Sampling Strategy	Kendall's τ	Spearman
Pooling Benchmark		
Uniform	0.16	0.21
Debiased	0.34	0.55
NAS-Bench-Macro		
Uniform	0.72	0.87
Debiased	0.74	0.91

Rank Correlation: For our benchmarks, we calculate the rank correlation with independent training in table II-3. Debiased supernet training shows improvement in rank correlation compared to uniform sampling on both benchmarks. In figure II-1, we show that the debiased method improves correlation, particularly among high-performing architectures.

Table-A II-4 Comparison of density calculation with KDE and inverse of mean distance

Density Calculation	Kendall's τ
Inverse mean distance (eq. 5 + eq. 6)	0.34
KDE (equation A II-2)	0.32

Kernel Density Estimation: In eq. 5 and 6, we used average cosine distances to estimate density. Kernel Density Estimation (KDE) (Parzen, 1962) can also be used to estimate the density:

$$\alpha_a = \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} K_h(f(a), f(i)) \tag{A II-2}$$

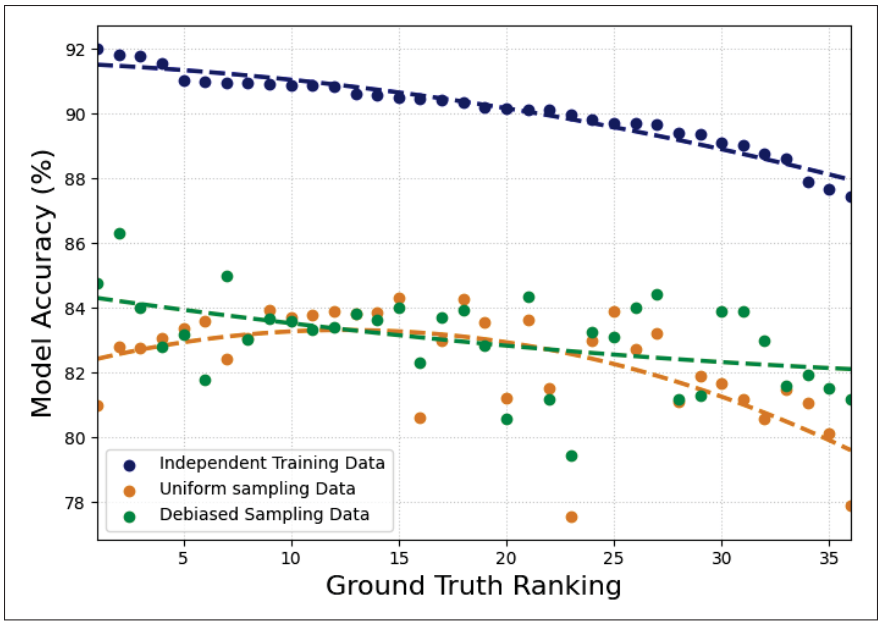


Figure-A II-1 Uniform and debiased sampling. Debiased sampling corrects the negative correlation present in uniform sampling for high rank architectures, leading to better evaluations for these architectures

with K_h, the Gaussian kernel function with a bandwidth h, a hyperparameter that determines the smoothness of the density estimation. Table II-4 shows that both approaches provide similar performances. Therefore, KDE introduces an additional hyperparameter h with no clear benefit to performance.

BIBLIOGRAPHY

Abdelfattah, M. S., Mehrotra, A., Dudziak, Ł. & Lane, N. D. (2021). Zero-cost proxies for lightweight NAS. *arXiv preprint arXiv:2101.08134*.

Ali, M. J., Essaid, M., Moalic, L. & Idoumghar, L. (2024). A review of AutoML optimization techniques for medical image applications. *Computerized Medical Imaging and Graphics*, 102441.

Arber Zela, T. E., Saikia, T., Marrakchi, Y., Brox, T. & Hutter, F. (2020). Understanding and robustifying differentiable architecture search. *International Conference on Learning Representations*, 2.

Asadi, K. & Littman, M. L. (2017). An alternative softmax operator for reinforcement learning. *International Conference on Machine Learning*, pp. 243–252.

Auer, P., Cesa-Bianchi, N. & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2), 235–256.

Badrinarayanan, V., Kendall, A. & Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12), 2481–2495.

Bae, W., Lee, S., Lee, Y., Park, B., Chung, M. & Jung, K.-H. (2019). Resource optimized neural architecture search for 3D medical image segmentation. *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part II 22*, pp. 228–236.

Bai, A., Wu, F. & Chen, X. (2013). Bayesian mixture modelling and inference based Thompson sampling in Monte-Carlo tree search. *Advances in neural information processing systems*, 26.

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

Baker, B., Gupta, O., Raskar, R. & Naik, N. (2017). Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*.

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V. & Le, Q. (2018). Understanding and simplifying one-shot architecture search. *International conference on machine learning*, pp. 550–559.

Bender, G., Liu, H., Chen, B., Chu, G., Cheng, S., Kindermans, P.-J. & Le, Q. V. (2020). Can weight sharing outperform random architecture search? an investigation with tunas. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14323–14332.

Benyahia, Y., Yu, K., Smires, K. B., Jaggi, M., Davison, A. C., Salzmann, M. & Musat, C. (2019). Overcoming multi-model forgetting. *International Conference on Machine Learning*, pp. 594–603.

Beume, N. (2009). S-metric calculation by considering dominated hypervolume as Klee's measure problem. *Evolutionary Computation*, 17(4), 477–492.

Bossard, L., Guillaumin, M. & Van Gool, L. (2014). Food-101–mining discriminative components with random forests. *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, pp. 446–461.

Brock, A., Lim, T., Ritchie, J. M. & Weston, N. (2017). Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*.

Brostow, G. J., Fauqueur, J. & Cipolla, R. (2009). Semantic object classes in video: A high-definition ground truth database. *Pattern recognition letters*, 30(2), 88–97.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1), 1–43.

Cai, H., Chen, T., Zhang, W., Yu, Y. & Wang, J. (2018a). Efficient architecture search by network transformation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

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

Cai, H., Gan, C., Wang, T., Zhang, Z. & Han, S. (2019). Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*.

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

Cesa-Bianchi, N., Gentile, C., Lugosi, G. & Neu, G. (2017). Boltzmann exploration done right. *Advances in neural information processing systems*, 30.

Cha, S., Kim, T., Lee, H. & Yun, S.-Y. (2022). Supernet in neural architecture search: A taxonomic survey. *arXiv preprint arXiv:2204.03916*.

Chau, T. C. P., Dudziak, Ł., Wen, H., Lane, N. D. & Abdelfattah, M. S. (2022). BLOX: Macro Neural Architecture Search Benchmark and Algorithms. *arXiv preprint arXiv:2210.07271*.

Chen, B., Ghiasi, G., Liu, H., Lin, T.-Y., Kalenichenko, D., Adam, H. & Le, Q. V. (2020). Mnasfpn: Learning latency-aware pyramid architecture for object detection on mobile devices. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13607–13616.

Chen, B., Li, P., Li, C., Li, B., Bai, L., Lin, C., Sun, M., Yan, J. & Ouyang, W. (2021a). Glit: Neural architecture search for global and local image transformer. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12–21.

Chen, H., Lin, M., Sun, X. & Li, H. (2021b). Nas-bench-zero: A large scale dataset for understanding zero-shot neural architecture search.

Chen, L.-H., Bampis, C. G., Li, Z., Chen, C. & Bovik, A. C. (2021c). Convolutional Block Design for Learned Fractional Downsampling. *arXiv preprint arXiv:2105.09999*.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K. & Yuille, A. L. (2014). Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K. & Yuille, A. L. (2017a). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4), 834–848.

Chen, L.-C., Papandreou, G., Schroff, F. & Adam, H. (2017b). Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*.

Chen, L.-C., Collins, M., Zhu, Y., Papandreou, G., Zoph, B., Schroff, F., Adam, H. & Shlens, J. (2018a). Searching for efficient multi-scale architectures for dense image prediction. *Advances in neural information processing systems*, 31.

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F. & Adam, H. (2018b). Encoder-decoder with atrous separable convolution for semantic image segmentation. *Proceedings of the European conference on computer vision (ECCV)*, pp. 801–818.

Chen, M., Fu, J. & Ling, H. (2021d). One-shot neural ensemble architecture search by diversity-guided search space shrinking. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16530–16539.

Chen, T., Goodfellow, I. & Shlens, J. (2015). Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*.

Chen, W., Gong, X., Liu, X., Zhang, Q., Li, Y. & Wang, Z. (2019a). Fasterseg: Searching for faster real-time semantic segmentation. *arXiv preprint arXiv:1912.10917*.

Chen, W., Gong, X. & Wang, Z. (2021e). Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. *arXiv preprint arXiv:2102.11535*.

Chen, X. & Hsieh, C.-J. (2020). Stabilizing differentiable architecture search via perturbation-based regularization. *International conference on machine learning*, pp. 1554–1565.

Chen, X., Xie, L., Wu, J. & Tian, Q. (2019b). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1294–1303.

Chen, X., Xie, L., Wu, J. & Tian, Q. (2021f). Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, 129, 638–655.

Chen, Y., Guo, Y., Liao, D., Lv, F., Song, H., Kwok, J. T.-Y. & Tan, M. (2024). Automated dominative subspace mining for efficient neural architecture search. *IEEE Transactions on Circuits and Systems for Video Technology*, 34(10), 9281–9297.

Chen, Y., Yang, T., Zhang, X., Meng, G., Xiao, X. & Sun, J. (2019c). Detnas: Backbone search for object detection. *Advances in neural information processing systems*, 32.

Chen, Z., Xie, L., Niu, J., Liu, X., Wei, L. & Tian, Q. (2021g). Visformer: The vision-friendly transformer. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 589–598.

Cheng, A.-C., Lin, C. H., Juan, D.-C., Wei, W. & Sun, M. (2020a). Instanas: Instance-aware neural architecture search. *Proceedings of the AAAI conference on artificial intelligence*, 34(04), 3577–3584.

Cheng, X., Zhong, Y., Harandi, M., Dai, Y., Chang, X., Li, H., Drummond, T. & Ge, Z. (2020b). Hierarchical neural architecture search for deep stereo matching. *Advances in neural information processing systems*, 33, 22158–22169.

Chitty-Venkata, K. T., Emani, M., Vishwanath, V. & Somani, A. K. (2023). Neural Architecture Search Benchmarks: Insights and Survey. *IEEE Access*, 11, 25217–25236.

Chrabaszcz, P., Loshchilov, I. & Hutter, F. (2017). A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*.

Chu, X., Li, X., Lu, S., Zhang, B. & Li, J. (2020a). Mixpath: A unified approach for one-shot neural architecture search. *arXiv preprint arXiv:2001.05887*.

Chu, X., Wang, X., Zhang, B., Lu, S., Wei, X. & Yan, J. (2020b). Darts-: robustly stepping out of performance collapse without indicators. *arXiv preprint arXiv:2009.01027*.

Chu, X., Zhou, T., Zhang, B. & Li, J. (2020c). Fair darts: Eliminating unfair advantages in differentiable architecture search. *European conference on computer vision*, pp. 465–480.

Chu, X., Zhang, B., Li, Q., Xu, R. & Li, X. (2021a). Scarlet-nas: bridging the gap between stability and scalability in weight-sharing neural architecture search. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 317–325.

Chu, X., Zhang, B., Ma, H., Xu, R. & Li, Q. (2021b). Fast, accurate and lightweight super-resolution with neural architecture search. *2020 25th International conference on pattern recognition (ICPR)*, pp. 59–64.

Chu, X., Zhang, B. & Xu, R. (2021c). Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *Proceedings of the IEEE/CVF International Conference on computer vision*, pp. 12239–12248.

Çiçek, Ö., Abdulkadir, A., Lienkamp, S. S., Brox, T. & Ronneberger, O. (2016). 3D U-Net: learning dense volumetric segmentation from sparse annotation. *International conference on medical image computing and computer-assisted intervention*, pp. 424–432.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S. & Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223.

Costa, V. G. & Pedreira, C. E. (2023). Recent advances in decision trees: An updated survey. *Artificial Intelligence Review*, 56(5), 4765–4800.

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. *International conference on computers and games*, pp. 72–83.

Courbariaux, M., Bengio, Y. & David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28.

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H. & Wei, Y. (2017). Deformable convolutional networks. *Proceedings of the IEEE international conference on computer vision*, pp. 764–773.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255.

Dong, X. & Yang, Y. (2019a). One-shot neural architecture search via self-evaluated template network. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3681–3690.

Dong, X. & Yang, Y. (2019b). Searching for a robust neural architecture in four gpu hours. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1761–1770.

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

Dong, X., Liu, L., Musial, K. & Gabrys, B. (2021). Nats-bench: Benchmarking nas algorithms for architecture topology and size. *IEEE transactions on pattern analysis and machine intelligence*, 44(7), 3634–3646.

Du, J., Wang, L., Liu, Y., Zhou, Z., He, Z. & Jia, Y. (2020a). Brain MRI super-resolution using 3D dilated convolutional encoder–decoder network. *IEEE Access*, 8, 18938–18950.

Du, X., Lin, T.-Y., Jin, P., Ghiasi, G., Tan, M., Cui, Y., Le, Q. V. & Song, X. (2020b). Spinenet: Learning scale-permuted backbone for recognition and localization. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11592–11601.

Duan, Y., Chen, X., Xu, H., Chen, Z., Liang, X., Zhang, T. & Li, Z. (2021). Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5251–5260.

Elfwing, S., Uchibe, E. & Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107, 3–11.

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

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

Elsken, T., Metzen, J. H. & Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55), 1–21.

Elsken, T., Staffler, B., Metzen, J. H. & Hutter, F. (2020). Meta-learning of neural architectures for few-shot learning. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12365–12375.

Everingham, M., Eslami, S. A., Van Gool, L., Williams, C. K., Winn, J. & Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111, 98–136.

Fang, J., Sun, Y., Zhang, Q., Li, Y., Liu, W. & Wang, X. (2020a). Densely connected search space for more flexible neural architecture search. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10628–10637.

Fang, J., Sun, Y., Zhang, Q., Peng, K., Li, Y., Liu, W. & Wang, X. (2020b). FNA++: Fast network adaptation via parameter remapping and architecture search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9), 2990–3004.

Floreano, D., Dürr, P. & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary intelligence*, 1(1), 47–62.

Gelly, S. & Wang, Y. (2006). Exploration exploitation in go: UCT for Monte-Carlo go. *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop*.

Ghiasi, G., Lin, T.-Y. & Le, Q. V. (2019). Nas-fpn: Learning scalable feature pyramid architecture for object detection. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7036–7045.

Gibb, S., La, H. M. & Louis, S. (2018). A genetic algorithm for convolutional network structure optimization for concrete crack detection. *2018 IEEE congress on evolutionary computation (CEC)*, pp. 1–8.

Golberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989(102), 36.

Goodarzi, P. (2020). *Visualizing and Understanding Convolutional Networks for Semantic Segmentation*. (Ph.D. thesis, Saarland University Saarbrücken, Germany).

Goodfellow, I. (2016). Deep learning. MIT press.

Gretton, A., Bousquet, O., Smola, A. & Schölkopf, B. (2005). Measuring statistical dependence with Hilbert-Schmidt norms. *International conference on algorithmic learning theory*, pp. 63–77.

Gu, Y.-C., Wang, L.-J., Liu, Y., Yang, Y., Wu, Y.-H., Lu, S.-P. & Cheng, M.-M. (2021). Dots: Decoupling operation and topology in differentiable architecture search. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12311–12320.

Guo, J., Han, K., Wang, Y., Zhang, C., Yang, Z., Wu, H., Chen, X. & Xu, C. (2020a). Hit-detector: Hierarchical trinity architecture search for object detection. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11405–11414.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y. & Sun, J. (2020b). Single path one-shot neural architecture search with uniform sampling. *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pp. 544–560.

Ha, H., Kim, J.-H., Park, S. & Chun, B.-G. (2021). SUMNAS: Supernet with Unbiased Meta-Features for Neural Architecture Search. *International Conference on Learning Representations*.

Hadash, G., Kermany, E., Carmeli, B., Lavi, O., Kour, G. & Jacovi, A. (2018). Estimate and Replace: A Novel Approach to Integrating Deep Neural Networks with Existing Applications. *arXiv preprint arXiv:1804.09028*.

He, K., Zhang, X., Ren, S. & Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9), 1904–1916.

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

He, K., Zhang, X., Ren, S. & Sun, J. (2016b). Identity mappings in deep residual networks. *European conference on computer vision*, pp. 630–645.

He, X., Zhao, K. & Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212, 106622.

Hendrycks, D. (2016). Gaussian Error Linear Units (Gelus). *arXiv preprint arXiv:1606.08415*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hu, H., Langford, J., Caruana, R., Horvitz, E. & Dey, D. (2018a). Macro neural architecture search revisited. *2nd Workshop on Meta-Learning at NeurIPS*.

Hu, H., Langford, J., Caruana, R., Mukherjee, S., Horvitz, E. J. & Dey, D. (2019). Efficient forward architecture search. *Advances in Neural Information Processing Systems*, 32.

Hu, J., Shen, L. & Sun, G. (2018b). Squeeze-and-excitation networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141.

Hu, S., Wang, R., Hong, L., Li, Z., Hsieh, C.-J. & Feng, J. (2022). Generalizing few-shot nas with gradient matching. *arXiv preprint arXiv:2203.15207*.

Hu, Y., Sun, S., Li, J., Wang, X. & Gu, Q. (2018c). A novel channel pruning method for deep neural network compression. *arXiv preprint arXiv:1805.11394*.

Hu, Y., Liang, Y., Guo, Z., Wan, R., Zhang, X., Wei, Y., Gu, Q. & Sun, J. (2020). Angle-based search space shrinking for neural architecture search. *European conference on computer vision*, pp. 119–134.

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

Huang, T., You, S., Wang, F., Qian, C., Zhang, C., Wang, X. & Xu, C. (2022). Greedynasv2: Greedier search with a greedy path filter. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11902–11911.

Hutter, F., Kotthoff, L. & Vanschoren, J. (2019). *Automated machine learning: methods, systems, challenges*. Springer Nature.

Ioffe, S. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Ishibuchi, H., Matsumoto, T., Masuyama, N. & Nojima, Y. (2020). Effects of dominance resistant solutions on the performance of evolutionary multi-objective and many-objective algorithms. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 507–515.

Jacot, A., Gabriel, F. & Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31.

Jang, D.-H., Chu, S., Kim, J. & Han, B. (2022). Pooling Revisited: Your Receptive Field is Suboptimal. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 549–558.

Javan, M., Toews, M. & Pedersoli, M. (2023). Balanced Mixture of SuperNets for Learning the CNN Pooling Architecture. *arXiv preprint arXiv:2306.11982*.

Jeon, J., Oh, Y., Lee, J., Baek, D., Kim, D., Eom, C. & Ham, B. (2025). Subnet-Aware Dynamic Supernet Training for Neural Architecture Search. *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 30137–30146.

Jie, H. J. & Wanda, P. (2020). RunPool: A dynamic pooling layer for convolution neural network. *International Journal of Computational Intelligence Systems*, 13(1), 66–76.

Jin, C., Tanno, R., Mertzanidou, T., Panagiotaki, E. & Alexander, D. C. (2021). Learning to downsample for segmentation of ultra-high resolution images. *arXiv preprint arXiv:2109.11071*.

Jin, H., Song, Q. & Hu, X. (2019). Auto-keras: An efficient neural architecture search system. *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1946–1956.

Kaiser, L., Gomez, A. N. & Chollet, F. (2017). Depthwise separable convolutions for neural machine translation. *arXiv preprint arXiv:1706.03059*.

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

Kang, D. & Ahn, C. W. (2019). Efficient neural network space with genetic search. *International Conference on Bio-Inspired Computing: Theories and Applications*, pp. 638–646.

Kang, J.-S., Kang, J., Kim, J.-J., Jeon, K.-W., Chung, H.-J. & Park, B.-H. (2023). Neural architecture search survey: A computer vision perspective. *Sensors*, 23(3), 1713.

Kim, S., Kim, I., Lim, S., Baek, W., Kim, C., Cho, H., Yoon, B. & Kim, T. (2019). Scalable neural architecture search for 3d medical image segmentation. *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part III 22*, pp. 220–228.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Klabunde, M., Schumacher, T., Strohmaier, M. & Lemmerich, F. (2025). Similarity of neural network models: A survey of functional and representational measures. *ACM Computing Surveys*, 57(9), 1–52.

Klein, A., Falkner, S., Bartels, S., Hennig, P. & Hutter, F. (2017). Fast bayesian optimization of machine learning hyperparameters on large datasets. *Artificial intelligence and statistics*, pp. 528–536.

Klein, A., Tiao, L. C., Lienart, T., Archambeau, C. & Seeger, M. (2020). Model-based asynchronous hyperparameter and neural architecture search. *arXiv preprint arXiv:2003.10865*.

Kocsis, L. & Szepesvári, C. (2006). Bandit based monte-carlo planning. *European conference on machine learning*, pp. 282–293.

Kornblith, S., Norouzi, M., Lee, H. & Hinton, G. (2019). Similarity of neural network representations revisited. *International conference on machine learning*, pp. 3519–3529.

Kour, G. & Saabne, R. (2014a). Fast classification of handwritten on-line Arabic characters. *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pp. 312–318.

Kour, G. & Saabne, R. (2014b). Real-time segmentation of on-line handwritten arabic script. *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pp. 417–422.

Krizhevsky, A., Hinton, G. et al. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

Kwasigroch, A., Grochowski, M. & Mikołajczyk, A. (2020). Neural architecture search for skin lesion classification. *Ieee Access*, 8, 9061–9071.

Le, H. & Borji, A. (2017). What are the receptive, effective receptive, and projective fields of neurons in convolutional neural networks? *arXiv preprint arXiv:1705.07049*.

Le, N. M., Vo, A. & Luong, N. H. (2024). Zero-Cost Proxy-Based Hierarchical Initialization for Evolutionary Neural Architecture Search. *2024 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8.

Leclerc, G., Ilyas, A., Engstrom, L., Park, S. M., Salman, H. & Madry, A. (2023). FFCV: Accelerating Training by Removing Data Bottlenecks. *Computer Vision and Pattern Recognition (CVPR)*.

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436–444.

Lee, C.-Y., Gallagher, P. W. & Tu, Z. (2016). Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. *Artificial intelligence and statistics*, pp. 464–472.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J. & Sohl-Dickstein, J. (2017). Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*.

Lee, J. & Ham, B. (2024). Az-nas: Assembling zero-cost proxies for network architecture search. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5893–5903.

Lee, N., Ajanthan, T. & Torr, P. H. (2018). Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*.

Li, C., Tang, T., Wang, G., Peng, J., Wang, B., Liang, X. & Chang, X. (2021). Bossnas: Exploring hybrid cnn-transformers with block-wisely self-supervised neural architecture search. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12281–12291.

Li, G., Hoang, D., Bhardwaj, K., Lin, M., Wang, Z. & Marculescu, R. (2023a). Zero-Shot Neural Architecture Search: Challenges, Solutions, and Opportunities. *arXiv preprint arXiv:2307.01998*.

Li, G., Yang, Y., Bhardwaj, K. & Marculescu, R. (2023b). Zico: Zero-shot nas via inverse coefficient of variation on gradients. *arXiv preprint arXiv:2301.11300*.

Li, G., Qian, G., Delgadillo, I. C., Muller, M., Thabet, A. & Ghanem, B. (2020a). Sgas: Sequential greedy architecture search. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1620–1630.

Li, H., Tao, C., Zhu, X., Wang, X., Huang, G. & Dai, J. (2020b). Auto seg-loss: Searching metric surrogates for semantic segmentation. *arXiv preprint arXiv:2010.07930*.

Li, J., Liu, Y., Liu, J. & Wang, W. (2020c). Neural architecture optimization with graph vae. *arXiv preprint arXiv:2006.10310*.

Li, L. & Talwalkar, A. (2020). Random search and reproducibility for neural architecture search. *Uncertainty in artificial intelligence*, pp. 367–377.

Li, L., Khodak, M., Balcan, M.-F. & Talwalkar, A. (2020d). Geometry-aware gradient algorithms for neural architecture search. *arXiv preprint arXiv:2004.07802*.

Li, X., Lin, C., Li, C., Sun, M., Wu, W., Yan, J. & Ouyang, W. (2020e). Improving one-shot nas by suppressing the posterior fading. *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 13836–13845.

Li, Y., Song, L., Chen, Y., Li, Z., Zhang, X., Wang, X. & Sun, J. (2020f). Learning dynamic routing for semantic segmentation. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8553–8562.

Li, Y., Li, J., Hao, C., Li, P., Xiong, J. & Chen, D. (2023c). Extensible and Efficient Proxy for Neural Architecture Search. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6199–6210.

Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K. & Li, Z. (2019). Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*.

Liang, J., Meyerson, E. & Miikkulainen, R. (2018). Evolutionary architecture search for deep multitask networks. *Proceedings of the genetic and evolutionary computation conference*, pp. 466–473.

Lin, M., Wang, P., Sun, Z., Chen, H., Sun, X., Qian, Q., Li, H. & Jin, R. (2021). Zen-nas: A zero-shot nas for high-performance image recognition. *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 347–356.

Lin, P., Sun, P., Cheng, G., Xie, S., Li, X. & Shi, J. (2020). Graph-guided architecture search for real-time semantic segmentation. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4203–4212.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. *European conference on computer vision*, pp. 740–755.

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

Liu, C., Chen, L.-C., Schroff, F., Adam, H., Hua, W., Yuille, A. L. & Fei-Fei, L. (2019a). Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 82–92.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C. & Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*.

Liu, H., Simonyan, K. & Yang, Y. (2018b). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Liu, J., Zhang, K., Hu, W. & Yang, Q. (2022). Improve Ranking Correlation of Supernet through Training Scheme from One-shot NAS to Few-shot NAS. *arXiv preprint arXiv:2206.05896*.

Liu, S., Lin, Z., Wang, Y., Zhang, J., Perazzi, F. & Johns, E. (2020). Shape adaptor: A learnable resizing module. *European Conference on Computer Vision*, pp. 661–677.

Liu, Y., Chen, K., Liu, C., Qin, Z., Luo, Z. & Wang, J. (2019b). Structured knowledge distillation for semantic segmentation. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2604–2613.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S. & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022.

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

Lopes, V. & Alexandre, L. A. (2023). Toward Less Constrained Macro-Neural Architecture Search. *IEEE Transactions on Neural Networks and Learning Systems*.

Lopes, V., Alirezazadeh, S. & Alexandre, L. A. (2021). Epe-nas: Efficient performance estimation without training for neural architecture search. *International conference on artificial neural networks*, pp. 552–563.

Lopes, V., Degardin, B. & Alexandre, L. A. (2023). Are neural architecture search benchmarks well designed? A deeper look into operation importance. *Information Sciences*, 650, 119695.

Lorenzo, P. R. & Nalepa, J. (2018). Memetic evolution of deep neural networks. *Proceedings of the genetic and evolutionary computation conference*, pp. 505–512.

Loshchilov, I. & Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Lu, S., Hu, Y., Yang, L., Sun, Z., Mei, J., Tan, J. & Song, C. (2023). Pa&da: Jointly sampling path and data for consistent NAS. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11940–11949.

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

Lu, Z., Whalen, I., Dhebar, Y., Deb, K., Goodman, E. D., Banzhaf, W. & Boddeti, V. N. (2020). Multiobjective evolutionary design of deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 25(2), 277–291.

Lu, Z., Sreekumar, G., Goodman, E., Banzhaf, W., Deb, K. & Boddeti, V. N. (2021). Neural architecture transfer. *IEEE transactions on pattern analysis and machine intelligence*, 43(9), 2971–2989.

Lu, Z., Cheng, R., Huang, S., Zhang, H., Qiu, C. & Yang, F. (2022). Surrogate-assisted multiobjective neural architecture search for real-time semantic segmentation. *IEEE Transactions on Artificial Intelligence*, 4(6), 1602–1615.

Lukasik, J., Friede, D., Zela, A., Hutter, F. & Keuper, M. (2021). Smooth variational graph embeddings for efficient neural architecture search. *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8.

Lukasik, J., Jung, S. & Keuper, M. (2022). Learning where to look–generative nas is surprisingly efficient. *European Conference on Computer Vision*, pp. 257–273.

Luo, W., Li, Y., Urtasun, R. & Zemel, R. (2016). Understanding the effective receptive field in deep convolutional neural networks. *Advances in neural information processing systems*, 29.

Luo, X., Liu, D., Kong, H., Huai, S., Chen, H. & Liu, W. (2022). SurgeNAS: A comprehensive surgery on hardware-aware differentiable neural architecture search. *IEEE Transactions on Computers*, 72(4), 1081–1094.

Ly-Manson, T., Leonardon, M., El Bey, A. A., Hacene, G. B. & Mauch, L. (2024). Analyzing Few-Shot Neural Architecture Search in a Metric-Driven Framework. *International Conference on Automated Machine Learning AutoML24*, 256, 1–33.

Ma, B., Zhang, J., Xia, Y. & Tao, D. (2023). Inter-layer transition in neural architecture search. *Pattern Recognition*, 143, 109697.

Ma, L., Zhou, Y., Ma, Y., Yu, G., Li, Q., He, Q. & Pei, Y. (2025). Defying Multi-Model Forgetting in One-Shot Neural Architecture Search Using Orthogonal Gradient Learning. *IEEE Transactions on Computers*.

Maas, A. L., Hannun, A. Y., Ng, A. Y. et al. (2013). Rectifier nonlinearities improve neural network acoustic models. *Proc. icml*, 30(1), 3.

Maddison, C. J., Tarlow, D. & Minka, T. (2014). A* sampling. *Advances in neural information processing systems*, 27.

Maziarz, K., Tan, M., Khorlin, A., Georgiev, M. & Gesmundo, A. (2018). Evolutionary-neural hybrid agents for architecture search. *arXiv preprint arXiv:1811.09828*.

Mehta, Y., White, C., Zela, A., Krishnakumar, A., Zabergja, G., Moradian, S., Safari, M., Yu, K. & Hutter, F. (2022). NAS-bench-suite: NAS evaluation is (now) surprisingly easy. *arXiv preprint arXiv:2201.13396*.

Mellor, J., Turner, J., Storkey, A. & Crowley, E. J. (2021). Neural architecture search without training. *International conference on machine learning*, pp. 7588–7598.

Metropolis, N. & Ulam, S. (1949). The monte carlo method. *Journal of the American statistical association*, 44(247), 335–341.

Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

Mohan, R., Elsken, T., Zela, A., Metzen, J. H., Staffler, B., Brox, T., Valada, A. & Hutter, F. (2023). Neural architecture search for dense prediction tasks in computer vision. *International Journal of Computer Vision*, 131(7), 1784–1807.

Murtagh, F. & Legendre, P. (2014). Ward's hierarchical agglomerative clustering method: which algorithms implement Ward's criterion? *Journal of classification*, 31, 274–295.

Nair, V. & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Namekawa, S. & Tezuka, T. (2021). Evolutionary neural architecture search by mutual information analysis. *2021 IEEE Congress on Evolutionary Computation (CEC)*, pp. 966–972.

Negrinho, R. & Gordon, G. (2017). Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*.

Nekrasov, V., Chen, H., Shen, C. & Reid, I. (2019). Fast neural architecture search of compact semantic segmentation models via auxiliary cells. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9126–9135.

Ning, X., Zheng, Y., Zhao, T., Wang, Y. & Yang, H. (2020). A generic graph-based neural architecture encoding scheme for predictor-based nas. *European Conference on Computer Vision*, pp. 189–204.

Ning, X., Tang, C., Li, W., Zhou, Z., Liang, S., Yang, H. & Wang, Y. (2021). Evaluating efficient performance estimators of neural architectures. *Advances in Neural Information Processing Systems*, 34, 12265–12277.

Noh, H., Hong, S. & Han, B. (2015). Learning deconvolution network for semantic segmentation. *Proceedings of the IEEE international conference on computer vision*, pp. 1520–1528.

Painter, M., Baioumy, M., Hawes, N. & Lacerda, B. (2024). Monte carlo tree search with boltzmann exploration. *Advances in Neural Information Processing Systems*, 36.

Parzen, E. (1962). On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3), 1065–1076.

Peng, J., Zhang, J., Li, C., Wang, G., Liang, X. & Lin, L. (2021). Pi-NAS: Improving neural architecture search by reducing supernet training consistency shift. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12354–12364.

Peng, Y., Chen, D. Z. & Sonka, M. (2025). U-net v2: Rethinking the skip connections of u-net for medical image segmentation. *2025 IEEE 22nd International Symposium on Biomedical Imaging (ISBI)*, pp. 1–5.

Perez-Rua, J.-M., Baccouche, M. & Pateux, S. (2018). Efficient progressive neural architecture search. *arXiv preprint arXiv:1808.00391*.

Pham, H., Guan, M., Zoph, B., Le, Q. & Dean, J. (2018). Efficient neural architecture search via parameters sharing. *International conference on machine learning*, pp. 4095–4104.

Pintea, S. L., Tömen, N., Goes, S. F., Loog, M. & van Gemert, J. C. (2021). Resolution learning in deep convolutional networks using scale-space theory. *IEEE Transactions on Image Processing*, 30, 8342–8353.

Pourchot, A., Ducarouge, A. & Sigaud, O. (2020). To share or not to share: A comprehensive appraisal of weight-sharing. *arXiv preprint arXiv:2002.04289*.

Qian, G., Zhang, X., Li, G., Zhao, C., Chen, Y., Zhang, X., Ghanem, B. & Sun, J. (2022). When NAS Meets Trees: An Efficient Algorithm for Neural Architecture Search. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2782–2787.

Radosavovic, I., Johnson, J., Xie, S., Lo, W.-Y. & Dollár, P. (2019). On network design spaces for visual recognition. *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1882–1890.

Rahman, M. H., Rizvee, M. M., Shomaji, S. & Chakraborty, P. (2024). ILASH: A Predictive Neural Architecture Search Framework for Multi-Task Applications. *arXiv preprint arXiv:2412.02116*.

Ramachandran, P., Zoph, B. & Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Rao, X., Zhao, B., Yi, X. & Liu, D. (2022). CR-LSO: Convex neural architecture optimization in the latent space of graph variational autoencoder with input convex neural networks. *arXiv preprint arXiv:2211.05950*.

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

Real, E., Aggarwal, A., Huang, Y. & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. *Proceedings of the aaai conference on artificial intelligence*, 33(01), 4780–4789.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Chen, X. & Wang, X. (2021). A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4), 1–34.

Riad, R., Teboul, O., Grangier, D. & Zeghidour, N. (2022). Learning strides in convolutional neural networks. *arXiv preprint arXiv:2202.01653*.

Richter, M. L. & Pal, C. (2022). Receptive Field Refinement for Convolutional Neural Networks Reliably Improves Predictive Performance. *arXiv preprint arXiv:2211.14487*.

Romero, D. W., Bruintjes, R.-J., Tomczak, J. M., Bekkers, E. J., Hoogendoorn, M. & van Gemert, J. C. (2021). Flexconv: Continuous kernel convolutions with differentiable kernel sizes. *arXiv preprint arXiv:2110.08059*.

Ronneberger, O., Fischer, P. & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pp. 234–241.

Roshtkhari, M. J., Toews, M. & Pedersoli, M. (2023). Balanced Mixture of Supernets for Learning the CNN Pooling Architecture. *International Conference on Automated Machine Learning*, pp. 8–1.

Roshtkhari, M. J., Toews, M. & Pedersoli, M. (2025a). Iterative Monte Carlo Tree Search for Neural Architecture Search. *International Conference on Automated Machine Learning*, pp. 3–1.

Roshtkhari, M. J., Toews, M. & Pedersoli, M. (2025b). Neural Architecture Search by Learning a Hierarchical Search Space. *arXiv preprint arXiv:2503.21061*.

Ru, R., Esperanca, P. & Carlucci, F. M. (2020). Neural architecture generator optimization. *Advances in Neural Information Processing Systems*, 33, 12057–12069.

Rumiantsev, P. & Coates, M. (2023). Performing neural architecture search without gradients. *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3), 211-252. doi: 10.1007/s11263-015-0816-y.

Saikia, T., Marrakchi, Y., Zela, A., Hutter, F. & Brox, T. (2019). Autodispnet: Improving disparity estimation with automl. *Proceedings of the ieee/cvf international conference on computer vision*, pp. 1812–1823.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520.

Saxena, S. & Verbeek, J. (2016). Convolutional neural fabrics. *Advances in neural information processing systems*, 29.

Schrodi, S., Stoll, D., Ru, B., Sukthanker, R., Brox, T. & Hutter, F. (2023). Construction of hierarchical neural architecture search spaces based on context-free grammars. *Advances in Neural Information Processing Systems*, 36, 23172–23223.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Seyrek, E. C. & Uysal, M. (2024). A comparative analysis of various activation functions and optimizers in a convolutional neural network for hyperspectral image classification. *Multimedia Tools and Applications*, 83(18), 53785–53816.

Shaw, A., Hunter, D., Landola, F. & Sidhu, S. (2019). Squeezenas: Fast neural architecture search for faster semantic segmentation. *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pp. 0–0.

Shu, Y., Wang, W. & Cai, S. (2019). Understanding architectures learnt by cell-based neural architecture search. *arXiv preprint arXiv:1909.09569*.

Siems, J., Zimmer, L., Zela, A., Lukasik, J., Keuper, M. & Hutter, F. (2020). Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 11.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al. (2016). Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587), 484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676), 354–359.

Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sinha, N. & Chen, K.-W. (2022). Novelty driven evolutionary neural architecture search. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 671–674.

Sinha, N., El Rahman Shabayek, A., Kacem, A., Rostami, P., Shneider, C. & Aouada, D. (2024a). Hardware aware evolutionary neural architecture search using representation similarity metric. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2628–2637.

Sinha, N., Rostami, P., El Rahman Shabayek, A., Kacem, A. & Aouada, D. (2024b). Multi-objective hardware aware neural architecture search using hardware cost diversity. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8032–8039.

Sinkhorn, R. (1964). A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2), 876–879.

Springenberg, J. T., Dosovitskiy, A., Brox, T. & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1), 1929–1958.

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

Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J. & Marculescu, D. (2020). Single-path nas: Designing hardware-efficient convnets in less than 4 hours. *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part II*, pp. 481–497.

Su, X., Huang, T., Li, Y., You, S., Wang, F., Qian, C., Zhang, C. & Xu, C. (2021a). Prioritized architecture sampling with monto-carlo tree search. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10968–10977.

Su, X., You, S., Zheng, M., Wang, F., Qian, C., Zhang, C. & Xu, C. (2021b). K-shot nas: Learnable weight-sharing for nas with k-shot supernets. *International Conference on Machine Learning*, pp. 9880–9890.

Sun, X., Panda, R., Feris, R. & Saenko, K. (2020a). Adashare: Learning what to share for efficient deep multi-task learning. *Advances in Neural Information Processing Systems*, 33, 8728–8740.

Sun, Y., Xue, B., Zhang, M. & Yen, G. G. (2019). Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24(2), 394–407.

Sun, Y., Xue, B., Zhang, M., Yen, G. G. & Lv, J. (2020b). Automatically designing CNN architectures using the genetic algorithm for image classification. *IEEE transactions on cybernetics*, 50(9), 3840–3854.

Sutton, R. S., Barto, A. G. et al. (1998). *Reinforcement learning: An introduction*. MIT press Cambridge.

Świechowski, M., Godlewski, K., Sawicki, B. & Mańdziuk, J. (2023). Monte Carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3), 2497–2562.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Tan, M. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.

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

Tanaka, H., Kunin, D., Yamins, D. L. & Ganguli, S. (2020). Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in neural information processing systems*, 33, 6377–6389.

Termritthikun, C., Jamtsho, Y., Ieamsaard, J., Muneesawang, P. & Lee, I. (2021). EEEA-Net: An early exit evolutionary neural architecture search. *Engineering Applications of Artificial Intelligence*, 104, 104397.

Turner, J., Crowley, E. J., O'Boyle, M., Storkey, A. & Gray, G. (2019). Blockswap: Fisher-guided block substitution for network compression on a budget. *arXiv preprint arXiv:1906.04113*.

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

Veniat, T. & Denoyer, L. (2018). Learning time/memory-efficient deep architectures with budgeted super networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3492–3500.

Wan, X., Ru, B., Esparança, P. M. & Carlucci, F. M. (2022a). Approximate neural architecture search via operation distribution learning. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2377–2386.

Wan, X., Ru, B., Esperança, P. M. & Li, Z. (2022b). On redundancy and diversity in cell-based neural architecture search. *arXiv preprint arXiv:2203.08887*.

Wang, C., Zhang, G. & Grosse, R. (2020a). Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*.

Wang, D., Li, M., Gong, C. & Chandra, V. (2021a). Attentivenas: Improving neural architecture search via attentive sampling. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6418–6427.

Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X. et al. (2020b). Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 43(10), 3349–3364.

Wang, L., Xie, S., Li, T., Fonseca, R. & Tian, Y. (2019a). Sample-efficient neural architecture search by learning action space. *arXiv preprint arXiv:1906.06832*.

Wang, L., Zhao, Y., Jinnai, Y., Tian, Y. & Fonseca, R. (2019b). Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*.

Wang, L., Fonseca, R. & Tian, Y. (2020c). Learning search space partition for black-box optimization using monte carlo tree search. *Advances in Neural Information Processing Systems*, 33, 19511–19522.

Wang, L., Zhao, Y., Jinnai, Y., Tian, Y. & Fonseca, R. (2020d). Neural architecture search using deep neural networks and monte carlo tree search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06), 9983–9991.

Wang, L., Xie, S., Li, T., Fonseca, R. & Tian, Y. (2021b). Sample-efficient neural architecture search by learning actions for monte carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9), 5503–5515.

Wang, N., Gao, Y., Chen, H., Wang, P., Tian, Z., Shen, C. & Zhang, Y. (2020e). NAS-FCOS: Fast neural architecture search for object detection. *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11943–11951.

Wang, R., Cheng, M., Chen, X., Tang, X. & Hsieh, C.-J. (2021c). Rethinking architecture selection in differentiable NAS. *arXiv preprint arXiv:2108.04392*.

Wang, X., Mi, Y. & Zhang, X. (2024). 3D human pose data augmentation using Generative Adversarial Networks for robotic-assisted movement quality assessment. *Frontiers in Neurorobotics*, 18, 1371385.

Wei, T., Wang, C., Rui, Y. & Chen, C. W. (2016). Network morphism. *International conference on machine learning*, pp. 564–572.

Weng, Y., Zhou, T., Li, Y. & Qiu, X. (2019). Nas-unet: Neural architecture search for medical image segmentation. *IEEE access*, 7, 44247–44257.

White, C., Neiswanger, W., Nolen, S. & Savani, Y. (2020). A study on encodings for neural architecture search. *Advances in neural information processing systems*, 33, 20309–20319.

White, C., Neiswanger, W. & Savani, Y. (2021). Bananas: Bayesian optimization with neural architectures for neural architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12), 10293–10301.

White, C., Khodak, M., Tu, R., Shah, S., Bubeck, S. & Dey, D. (2022). A deeper look at zero-cost proxies for lightweight nas. *ICLR Blog Track*.

White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T., Zela, A., Dey, D. & Hutter, F. (2023). Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3), 229–256.

Wistuba, M. (2017). Finding competitive network architectures within a day using uct. *arXiv preprint arXiv:1712.07420*.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y. & Keutzer, K. (2019a). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10734–10742.

Wu, H., Zhang, J. & Huang, K. (2019b). Sparsemask: Differentiable connectivity learning for dense image prediction. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6768–6777.

Wu, J. & Fan, Y. (2023). Hnas-reg: hierarchical neural architecture search for deformable medical image registration. *2023 IEEE 20th International Symposium on Biomedical Imaging (ISBI)*, pp. 1–4.

Xia, H., Li, C., Zeng, S., Tan, Q., Wang, J. & Yang, S. (2022a). Learning to Search Promising Regions by a Monte-Carlo Tree Model. *2022 IEEE Congress on Evolutionary Computation (CEC)*, pp. 01–08.

Xia, X., Xiao, X., Wang, X. & Zheng, M. (2022b). Progressive automatic design of search space for one-shot neural architecture search. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2455–2464.

Xiao, H., Wang, Z., Zhu, Z., Zhou, J. & Lu, J. (2022). Shapley-NAS: discovering operation contribution for neural architecture search. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11892–11901.

Xiao, S., Zhao, B. & Liu, D. (2023). Latent Space Neural Architecture Search via LambdaNDCGloss-Based Listwise Ranker. *2023 International Annual Conference on Complex Systems and Intelligent Science (CSIS-IAC)*, pp. 160–165.

Xie, L. & Yuille, A. (2017). Genetic cnn. *Proceedings of the IEEE international conference on computer vision*, pp. 1379–1388.

Xie, L., Chen, X., Bi, K., Wei, L., Xu, Y., Wang, L., Chen, Z., Xiao, A., Chang, J., Zhang, X. et al. (2021). Weight-sharing neural architecture search: A battle to shrink the optimization gap. *ACM Computing Surveys (CSUR)*, 54(9), 1–37.

Xie, S., Girshick, R., Dollár, P., Tu, Z. & He, K. (2017). Aggregated residual transformations for deep neural networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500.

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

Xu, H., Yao, L., Zhang, W., Liang, X. & Li, Z. (2019a). Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6649–6658.

Xu, J., Tan, X., Song, K., Luo, R., Leng, Y., Qin, T., Liu, T.-Y. & Li, J. (2022). Analyzing and mitigating interference in neural architecture search. *International Conference on Machine Learning*, pp. 24646–24662.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q. & Xiong, H. (2019b). Pc-darts: Partial channel connections for memory-efficient architecture search. *arXiv preprint arXiv:1907.05737*.

Xu, Z., Zuo, S., Lam, E. Y., Lee, B. & Chen, N. (2020). AutoSegNet: An automated neural network for image segmentation. *Ieee Access*, 8, 92452–92461.

Xue, C., Wang, X., Yan, J., Hu, Y., Yang, X. & Sun, K. (2021). Rethinking Bi-level optimization in neural architecture search: a gibbs sampling perspective. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12), 10551–10559.

Xue, Y., Zha, J., Wahib, M., Ouyang, T. & Wang, X. (2024). Neural architecture search via similarity adaptive guidance. *Applied Soft Computing*, 162, 111821.

Yan, S., Zheng, Y., Ao, W., Zeng, X. & Zhang, M. (2020). Does unsupervised architecture representation learning help neural architecture search? *Advances in neural information processing systems*, 33, 12486–12498.

Yan, S., Song, K., Liu, F. & Zhang, M. (2021). Cate: Computation-aware neural architecture encoding with transformers. *International Conference on Machine Learning*, pp. 11670–11681.

Yang, A., Esperança, P. M. & Carlucci, F. M. (2019). NAS evaluation is frustratingly hard. *arXiv preprint arXiv:1912.12522*.

Yao, L., Xu, H., Zhang, W., Liang, X. & Li, Z. (2020). SM-NAS: Structural-to-modular neural architecture search for object detection. *Proceedings of the AAAI conference on artificial intelligence*, 34(07), 12661–12668.

Ye, P., Li, B., Li, Y., Chen, T., Fan, J. & Ouyang, W. (2022). b-darts: Beta-decay regularization for differentiable architecture search. *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10874–10883.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K. & Hutter, F. (2019). Nas-bench-101: Towards reproducible neural architecture search. *International conference on machine learning*, pp. 7105–7114.

You, S., Huang, T., Yang, M., Wang, F., Qian, C. & Zhang, C. (2020). Greedynas: Towards fast one-shot nas with greedy supernet. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1999–2008.

Yu, F. & Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*.

Yu, H., Wan, C., Dai, X., Liu, M., Chen, D., Xiao, B., Huang, Y., Lu, Y. & Wang, L. (2024). Real-time image segmentation via hybrid convolutional-transformer architecture search. *arXiv preprint arXiv:2403.10413*.

Yu, J. & Huang, T. S. (2019). Universally slimmable networks and improved training techniques. *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1803–1811.

Yu, J., Jin, P., Liu, H., Bender, G., Kindermans, P.-J., Tan, M., Huang, T., Song, X., Pang, R. & Le, Q. (2020a). Bignas: Scaling up neural architecture search with big single-stage models. *European Conference on Computer Vision*, pp. 702–717.

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

Yu, K., Ranftl, R. & Salzmann, M. (2020b). How to train your super-net: An analysis of training heuristics in weight-sharing nas. *arXiv preprint arXiv:2003.04276*.

Yu, Q., Yang, D., Roth, H., Bai, Y., Zhang, Y., Yuille, A. L. & Xu, D. (2020c). C2fnas: Coarse-to-fine neural architecture search for 3d medical image segmentation. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4126–4135.

Yuan, Y., Chen, X. & Wang, J. (2020). Object-contextual representations for semantic segmentation. *European conference on computer vision*, pp. 173–190.

Zagoruyko, S. & Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

Zaniolo, L. & Marques, O. (2020). On the use of variable stride in convolutional neural networks. *Multimedia Tools and Applications*, 79(19), 13581–13598.

Zeiler, M. D. & Fergus, R. (2014). Visualizing and understanding convolutional networks. *European conference on computer vision*, pp. 818–833.

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

Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T. & Hutter, F. (2019). Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656*.

Zhang, B., Wang, X., Qin, X. & Yan, J. (2024a). Boosting Order-Preserving and Transferability for Neural Architecture Search: a Joint Architecture Refined Search and Fine-tuning Approach. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5662–5671.

Zhang, B., Wu, X., Miao, H., Guo, C. & Yang, B. (2024b). Dependency-Aware Differentiable Neural Architecture Search. *European Conference on Computer Vision*, pp. 219–236.

Zhang, M., Li, H., Pan, S., Chang, X. & Su, S. (2020a). Overcoming multi-model forgetting in one-shot NAS with diversity maximization. *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, pp. 7809–7818.

Zhang, M., Li, H., Pan, S., Liu, T. & Su, S. (2020b). One-shot neural architecture search via novelty driven sampling. *29th International Joint Conference on Artificial Intelligence*.

Zhang, M., Su, S. W., Pan, S., Chang, X., Abbasnejad, E. M. & Haffari, R. (2021a). idarts: Differentiable architecture search with stochastic implicit gradients. *International Conference on Machine Learning*, pp. 12557–12566.

Zhang, M., Yu, X., Zhao, H. & Ou, L. (2023). Shiftnas: Improving one-shot nas via probability shift. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5919–5928.

Zhang, M., Jiang, S., Cui, Z., Garnett, R. & Chen, Y. (2019a). D-vae: A variational autoencoder for directed acyclic graphs. *Advances in Neural Information Processing Systems*, 32.

Zhang, W., Zhao, L., Li, Q., Zhao, S., Dong, Q., Jiang, X., Zhang, T. & Liu, T. (2019b). Identify hierarchical structures from task-based fMRI data via hybrid spatiotemporal neural architecture search net. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 745–753.

Zhang, X., Xu, H., Mo, H., Tan, J., Yang, C., Wang, L. & Ren, W. (2021b). Dcnas: Densely connected neural architecture search for semantic image segmentation. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13956–13967.

Zhang, Y., Qiu, Z., Liu, J., Yao, T., Liu, D. & Mei, T. (2019c). Customizable architecture search for semantic segmentation. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11641–11650.

Zhang, Y.-D., Pan, C., Sun, J. & Tang, C. (2018). Multiple sclerosis identification by convolutional neural network with dropout and parametric ReLU. *Journal of computational science*, 28, 1–10.

Zhang, Y., Lin, Z., Jiang, J., Zhang, Q., Wang, Y., Xue, H., Zhang, C. & Yang, Y. (2020c). Deeper insights into weight sharing in neural architecture search. *arXiv preprint arXiv:2001.01431*.

Zhang, Y., Zhang, Q. & Yang, Y. (2020d). How does supernet help in neural architecture search? *arXiv preprint arXiv:2010.08219*.

Zhao, G., Wang, J., Zhang, Z. et al. (2017). Random Shifting for CNN: a Solution to Reduce Information Loss in Down-Sampling Layers. *IJCAI*, pp. 3476–3482.

Zhao, Y., Liang, Z., Lu, Z. & Cheng, R. (2024). A multi-objective optimization benchmark test suite for real-time semantic segmentation. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 163–166.

Zhao, Y., Wang, L., Tian, Y., Fonseca, R. & Guo, T. (2021a). Few-shot neural architecture search. *International Conference on Machine Learning*, pp. 12707–12718.

Zhao, Y., Wang, L., Yang, K., Zhang, T., Guo, T. & Tian, Y. (2021b). Multi-objective optimization by learning space partitions. *arXiv preprint arXiv:2110.03173*.

Zheng, X., Fei, X., Zhang, L., Wu, C., Chao, F., Liu, J., Zeng, W., Tian, Y. & Ji, R. (2022). Neural architecture search with representation mutual information. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11912–11921.

Zheng, X., Ma, Y., Xi, T., Zhang, G., Ding, E., Li, Y., Chen, J., Tian, Y. & Ji, R. (2025). An Information Theory-Inspired Strategy for Automated Network Pruning. *International Journal of Computer Vision*, 1–28.

Zhong, Z., Yan, J., Wu, W., Shao, J. & Liu, C.-L. (2018). Practical block-wise neural network architecture generation. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2423–2432.

Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A. & Torralba, A. (2017). Scene parsing through ade20k dataset. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 633–641.

Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A. & Torralba, A. (2019). Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127, 302–321.

Zhou, D., Zhou, X., Zhang, W., Loy, C. C., Yi, S., Zhang, X. & Ouyang, W. (2020). Econas: Finding proxies for economical neural architecture search. *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 11396–11404.

Zhou, X., Wu, X., Feng, L., Lu, Z. & Tan, K. C. (2025). Design principle transfer in neural architecture search via large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(21), 23000–23008.

Zhou, Y., Xie, X. & Kung, S.-Y. (2021). Exploiting operation importance for differentiable neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*.

Zhou, Z., Rahman Siddiquee, M. M., Tajbakhsh, N. & Liang, J. (2018). Unet++: A nested u-net architecture for medical image segmentation. *International workshop on deep learning in medical image analysis*, pp. 3–11.

Zhu, C., Li, L., Wu, Y. & Sun, Z. (2024). Saswot: Real-time semantic segmentation architecture search without training. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(7), 7722–7730.

Zhu, H., An, Z., Yang, C., Xu, K., Zhao, E. & Xu, Y. (2019a). EENA: efficient evolution of neural architecture. *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0.

Zhu, Z., Liu, C., Yang, D., Yuille, A. & Xu, D. (2019b). V-NAS: Neural architecture search for volumetric medical image segmentation. *2019 International conference on 3d vision (3DV)*, pp. 240–248.

Zoph, B. & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

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