Efficient Deep Neural Architecture Search via Bayesian Optimization : An application to Computer Vision

Houssem Ouertatani

THÈSE

pour l'obtention du grade de

Docteur de l'Université de Lille

Spécialité: Informatique et applications

Efficient Deep Neural Architecture Search via Bayesian Optimization: an application to Computer Vision

Recherche efficace d'architectures neuronales profondes via Optimisation Bayésienne : application à la vision par ordinateur

---

Ecole Doctorale:

Ecole Graduée MADIS-631

Unités de recherche:

IRT SystemX

CRIStAL UMR 9189 - INRIA Lille - Equipe BONUS

Présentée et soutenue par

Houssem Ouertatani

le 6 Décembre 2024 à Lille

Président du jury: **Pr. Mohamed Daoudi**

Jury:

Pr. El-Ghazali Talbi	CRIStAL, Université de Lille	*Co-directeur de thèse*
Pr. Smail Niar	LAMIH, UPHF	*Co-directeur de thèse*
Pr. Amir Nakib	LISSI, Université Paris-Est Créteil	*Rapporteur*
Pr. Saïd Mahmoudi	ILIA, Université de Mons	*Rapporteur*
DR Carola Doerr	LIP6, CNRS, Sorbonne Université	*Examinatrice*
Pr. Mohamed Daoudi	CRIStAL, ULille & IMT Nord Europe	*Examinateur*
Igr. Cristian Maxim	IRT SystemX	*Co-encadrant de thèse*

Thèse préparée à IRT SystemX
sous l'encadrement de **Cristian Maxim**

Acknowledgements

At the tail end of this three-year adventure, I express the deepest gratitude to the people who, personally or professionally, played a role in the completion of this thesis.

Heartfelt thanks are due to my thesis directors, professors El-Ghazali Talbi and Smail Niar, for their expert guidance, constant support, abundant patience and invaluable insight. It was a true privilege to interact with them, to explore research directions, and to exchange ideas and perspectives. Their positive impact goes beyond the contents of this manuscript.

I wish to convey my sincere gratitude to the esteemed thesis committee members, rapporteurs Pr. Saïd Mahmoudi and Pr. Amir Nakib, and examiners Pr. Mohamed Daoudi and Pr. Carola Doerr. Your time, attention, constructive remarks and insights are highly appreciated.

Cristian Maxim was instrumental in advising me and supporting me for the past three years. His unwavering encouragement, kindness and understanding have made it a genuine pleasure to work with him and helped me overcome any hurdles I faced. Our strong rapport grew into a personal friendship, for which I am truly grateful.

Speaking of colleagues becoming friends, Adrien Le Coz deserves my sincere thanks for making the past three years not only more enriching but also much more enjoyable. Our collaboration has led to a publication at the intersection of our respective research topics. Beyond that, our discussions, which spanned everything from cutting-edge AI research to the latest games and movies, shaped my time at IRT in the most positive way.

Pr. Patrice Aknin supported me greatly and facilitated many aspects of this thesis. I want to thank him for his patience, his understanding and kind support. I extend my thanks to everyone who contributed to the *Confiance.ai* program, especially within the EC7 project.

I also have an appreciation for everyone who contributes to the Grid5000 cluster, whose computing resources were instrumental to my work. I would be remiss not to mention my labmates and friends at IRT as well as at INRIA Bonus.

I'm profoundly grateful to my mom, Fatma, for her boundless love and endless support, to my sister, Najla, for her constant encouragement and understanding, and to all my family members who have been a limitless wellspring of optimism, encouragement, and generosity. A heartfelt thanks also goes to my friends, whose companionship, support, and good humor have made all the difference.

Abstract

Neural network architectures are at the heart of the tremendous success deep learning has demonstrated in multitudes of tasks. Neural Architecture Search (NAS) is a critical task in the development of efficient vision models, instrumental in unearthing new efficient architecture patterns or optimally adapting existing architectures to hardware and deployment constraints, for instance via hardware-aware NAS.

While many blackbox optimization algorithms are well-suited for such problems, the high cost of evaluating individual solutions places a special focus on highly sample-efficient methods like Bayesian Optimization (BO).

In this work, we start from the core principles behind the sample efficiency of BO, and leverage certain attributes of NAS and the inherent flexibility and predictive performance of deep ensembles to significantly reduce the time and resources required to effectively explore the search spaces.

On NAS benchmarks, this search strategy achieves a 100x acceleration compared to random search, and up to 50% reduction compared to leading BO-based methods and local search methods.

Creating neural network architectures is a complex design problem which can quickly lead to combinatorial explosion. Well-considered search space design is an important requirement for quickly finding high-performing models.

We showcase the versatility and effectiveness of this search approach on a range of search spaces of varying types and degrees of complexity. Our focus is on finding and improving efficient vision model architectures intended for edge devices. We demonstrate the possibility of finding new and high-performing designs without incurring high computing costs.

Résumé

Les architectures de réseaux neuronaux sont au centre de l'immense succès de l'apprentissage profond dans de nombreuses tâches. La recherche d'architectures de réseaux de neurones (NAS) est une tâche critique dans le développement de modèles d'apprentissage profond efficaces, qui permet de découvrir de nouvelles composantes ou configurations efficaces, ou bien d'adapter de manière optimale les architectures existantes aux contraintes matérielles de déploiement, par exemple par le biais du NAS adapté au matériel.
Bien que les algorithmes d'optimisation en boîte noire soient bien adaptés à ces problèmes, le coût élevé de l'évaluation des solutions individuelles met l'accent sur les méthodes à forte efficacité d'échantillonnage comme l'Optimisation Bayésienne (BO).
Dans ce travail, nous partons des principes fondamentaux qui sous-tendent l'efficacité d'échantillonnage de l'Optimisation Bayésienne, et nous tirons parti de certains attributs de la NAS et de la flexibilité inhérente et de la performance prédictive des Deep Ensembles pour réduire de manière significative le temps de recherche et les ressources nécessaires pour explorer efficacement les espaces de recherche.
Sur les benchmarks NAS, cette stratégie de recherche atteint une accélération de 100x par rapport à la recherche aléatoire, et jusqu'à 50% de réduction du temps de recherche par rapport aux méthodes basées sur l'Optimisation Bayésienne ou la recherche locale.
La conception d'architectures de réseaux neuronaux est un problème complexe qui peut rapidement conduire à une explosion combinatoire. Une conception judicieuse de l'espace de recherche est une condition importante pour trouver rapidement des modèles performants.
Nous démontrons la polyvalence et l'efficacité de cette approche de recherche sur plusieurs espaces de recherche de types et de degrés de complexité différents. Nous nous concentrons sur la recherche et l'amélioration d'architectures de modèles de vision efficaces. Nous démontrons qu'il est possible de trouver de nouveaux modèles très performants tout en limitant les coûts de calcul requis.

Contents

List of Figures

List of Tables

1 Introduction

Virtually every facet of human activity has gone through monumental shifts since the advent of computing technologies. With the introduction of the first digital computers, the emergence of microchips, the meteoric rise of the Internet, the rapid expansion of on-the-go computing with modern smartphones, each of these developments were the culmination of prior progress, and paved the way for future milestones. Density on microchips increased exponentially, doubling every 18 months for decades just as Gordon Moore predicted in his eponymous law (Intel Corporation, 2024). Ideas once confined to science-fiction started finding their way into our daily lives, with the emergence of new devices with continuously improving capabilities.

One of the elusive but powerful ideas which have accompanied the development of computing since the early days is making these systems match or surpass human intelligence. This idea went through a number of important milestones. The Dartmouth summer workshop in 1956 (Dartmouth College, 2024) dubbed it “Artificial Intelligence” and established it as a research field. In 1997, Deep Blue bested a reigning chess champion. We are currently living an unprecedented success of AI products powered by large language models, used by millions of users for multitudes of productive tasks. But this evolution was tumultuous at times, with alternating periods of high optimism and new advances dubbed "AI summers", and phases of relative stagnation or relatively underwhelming achievements, "AI winters".

In the past twelve years, the stars have aligned: with the simultaneous availability of enormous quantities of data and the necessary computing capabilities, a class of machine learning models referred to as deep neural networks (DNNs) have led to incredible growth and impressive successes.

AI is here: the rise and rise of deep neural networks

Deep neural networks have completely revolutionized entire domains. They demonstrated undeniable aptitudes at solving a class of tasks ordinarily trivial for humans and extremely complicated for machines. Image recognition is a prime example, where distinguishing

between pictures of a dog and a cat is simple for small children, but for which classical vision algorithms were far behind human performance. In constrast, deep neural networks (DNNs) have the capability to perform as well as or even better than humans at such tasks. These impressive capabilities are directly related to how these models operate. While we cannot write an explicit sequence of instructions to reliably enable a model to recognize a concept as imprecise and nebulous as "picture of a dog", DNNs can automatically learn to dependably recognize these pictures by learning from a large set of diverse examples. By teaching the model to accurately perform this classification task on high quality training data, we hope it generalizes well to new unseen data. This paradigm of training models is **supervised learning**, and it only constitutes one part of the breadth of advancements in deep learning (DL).

Since 2012 (Krizhevsky et al., 2012), deep learning models have played an ever more important role in many research fields. The remarkable success has also led to widespread adoption in industrial applications and digital tools, enabling AI to become a mainstream subject of discussion. Today, AI discourse and applications are ubiquitous, and important questions about its impact, its limits, the possibilities it offers and the risks it could entail are as relevant as they have ever been.

Efficient vision models and on-device DL

Graphical Processing Units (GPUs) were originally devised to power computer graphics, and achieved commercial success and mainstream availability boosted by gaming. Since they are required to quickly render frames containing a few million pixels during their operation, they have the capacity to handle massively parallel computations. As a result, they were the perfect computing device for DNNs, which have millions (or billions) of parameters to be trained on successions of data batches. Since, DNN workload accelerators have expanded beyond CPUs and GPUs, for instance with TPUs (Jouppi et al., 2017, 2023) and other specialized accelerators. Inference is also performed on much less capable hardware in many use cases, for instance on ASICs or FPGAs in edge devices.

Currently, the vast majority of inference workloads performed using NNs run on cloud servers, dominated by high-performance accelerators (GPUs, TPUs, etc.). However, many analysts, such as in (Gartner, 2024), expect this portion to drop significantly in the next years. In fact, for a variety of reasons, on-device inference with low-power edge devices is rapidly gaining steam. An illustrative example is the recent flagship smartphones which boast the hardware (e.g. Google's Tensor chips (Google, 2024), Apple's Neural Engine (Apple, 2024b)) and software components (e.g. Google's Gemini Nano (DeepMind, 2024), Apple's on-device foundation models (Apple, 2024a)) to perform a variety of tasks on-device, from removing objects and infilling their pixels, to summarizing audio calls and generating images from text prompts. On-device processing is expected to gain importance, both in mobile devices, and other edge devices like smart sensors. Some of the reasons for this trend include:

- **Practicality:** offline availability is one example of an advantage for on-device inference. Another one is speed, as specialized on-device hardware, possibly co-designed with the DNN itself, can perform the task on demand and with little latency, without depending on the network and on server availability. Latency and response times can be critical for interactive scenarios.
- **Privacy:** this is arguably the most important motivation behind on-device inference, as sensitive data (e.g. personal details, health and biometric data) remains exclusively on the device, ensuring the protection of privacy and increasing trust in the safety of the feature.

With this rising demand for on-device DL, efficient models will gain more and more importance. This aspect of course adds to their other important advantages, especially the myriad of use cases they enable by requiring only minimal power consumption, for instance for devices with limited battery life.

Successful deployment of deep learning models is the outcome of a number of important steps, including data collection and preparation, hyperparameter tuning, training and testing, pre or post training quantization and pruning. A key aspect is the selection of a suitable model architecture. This choice is highly consequential for the performance of the model. Designing the architecture can be performed manually or using optimization, in a process known as Neural Architecture Search (NAS).

Thesis Plan

With this context in mind, this thesis is focused on the automatic optimization of neural architectures efficient vision models, specifically using Bayesian Optimization.

In part I, we present a general overview of the important concepts relevant to our contributions and to the research context they belong in. We begin with a concise introduction of efficient vision models in chapter 2. We then move on to an exploration of Neural Architecture Search (NAS) in chapter 3. Finally, chapter 4 centers on Bayesian Optimization.

Part 2 II introduces a fast NAS method based on Bayesian Optimization and deep ensembles. It is accelerated using a pretraining method and multi-fidelity search, and a NAS benchmark is used to test its performance. Then, an architecture tuning tool based on this search strategy is presented, enabling a simple way to efficiently search for models using existing architecture code or new designs with few restrictions or constraints.

Part 3 III applies this architecture optimization method and search tool to a diverse range of custom search spaces of varying complexity levels. Using hybrids of existing architectures, or constructing search spaces of custom operators (e.g. a search space of efficient self-attention mechanisms), this part focuses on examples of using the BO-based search strategy to find new

designs of efficient vision models.

Related Works Part I

2 Efficient vision models: A brief overview

In this chapter, we present a brief overview of efficient vision models, destined for edge AI and mobile use cases.

At first, similarly to other areas of computer vision, ConvNets dominated this area via efficient variants of the convolution operation.

The advent of the vision transformer has challenged this hegemony, but its heavy impact on latency motivated the introduction of many ConvNet-ViT hybrid models, as well as lighter designs of the underlying self-attention mechanism.

2.1 The ConvNets era

Convolutional Neural Networks (CNNs or ConvNets) (LeCun et al., 1998) were introduced as a more parameter efficient and sparse alternative to multi-layer perceptrons, with an interesting property which made them very powerful for image-based learning: translation equivariance. Since (Krizhevsky et al., 2012) and until recently, ConvNet-based networks have been ubiquitous in deep learning in general, and held a dominant position in computer vision in particular. Successful ConvNet-based solutions can be found in every vision task: classification, upsampling, semantic segmentation, object detection, human and object pose estimation, 3D information processing...

At the heart of the ConvNet architecture is the convolution operator. It receives a feature map $\mathrm{x}_{\text{in}} \in \mathbb{R}^{h \times w \times c_{\text{in}}}$ with height h, width w and c_{in} channels. The convolution layer encloses c_{out} kernels $\mathrm{K}_i \in \mathbb{R}^{k_0 \times k_1 \times c_{\text{in}}}$ of learnable weights. Usually $k_0 = k_1 = k$ is the kernel size. During a forward pass, each kernel is "slid" along the horizontal and vertical dimensions of the input map and its dot product with that section of the feature map is computed, producing one channel of the output feature map. Stacking the outputs of the c_{out} kernels and adding a learnable bias produces an output of size $\mathrm{x}_{\text{out}} \in \mathbb{R}^{h \times w \times c_{\text{out}}}$.

This description is applicable for layers which preserve the resolution, which depends on how the edge pixels are handled (e.g. zero-padding the missing pixels), and how the sliding mechanism works (e.g. with a stride of 2 the resolution is halved).

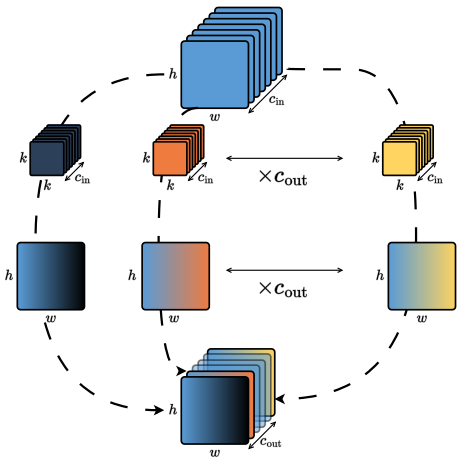


Figure 2.1: Convolution layer

Progressively, new insights pushed the envelope further in terms of performance with ConvNets. VGG (Simonyan, 2014) explored reducing the kernel size while increasing network depth, which demonstrated further accuracy gains. ResNet (He et al., 2016a) addressed the vanishing gradient problem with residual connections, where the output and the input are summed before passing to the next layer. This design allows the creation of much deeper

networks with more accuracy gains. DenseNet (Huang et al., 2017) goes further by connecting (skip connections) each block with all the previous blocks. Group convolutions, first introduced in AlexNet (Krizhevsky et al., 2012) to simplify distributed training because of memory limitations, splits the feature map into groups along the channel dimension and processes each group by its own convolution kernel. ResNeXt (Xie et al., 2017) leverages grouped convolutions for better performance. More recently, ConvNeXt (Liu et al., 2022) introduced a family of ConvNets with competitive, and in some cases superior, performance to vision transformer (ViT) networks, by methodically analyzing the learnings from these modern designs and applying them to pure ConvNets.

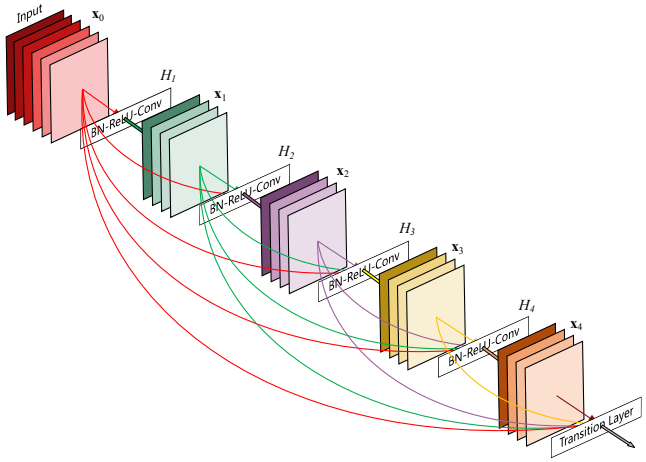


Figure 2.2: DenseNet block. Source: (Huang et al., 2017)

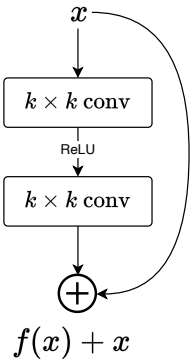


Figure 2.3: Residual block

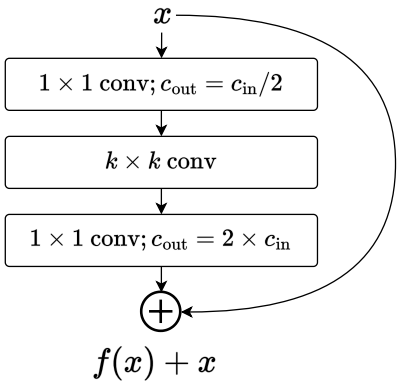


Figure 2.4: Residual bottleneck block

2.2 Efficient ConvNets

When it comes to ConvNets made for edge devices, design modifications were introduced to the convolution operator to make it more efficient. Some of these designs managed to achieve good accuracy figures at a fraction of the computational cost of bigger models, making them easier to deploy on edge devices as backbones useful for a multitude of downstream tasks.

One of the most successful redesigns of the convolution layer for edge devices is the depthwise-separable convolution used in MobileNet (Howard et al., 2017). The idea is to separate the spacial convolution (height and width) from the depth convolution (channels). A first step is a "depthwise convolution", where a series of c_{in} kernels $\mathrm{K}_{\mathrm{d}\,i} \in \mathbb{R}^{k_0 \times k_1 \times 1}$ perform the convolution on a one kernel per channel basis. With the processing along the spatial dimensions done, a pointwise convolution using c_{out} kernels $\mathrm{K}_{\mathrm{p}\,i} \in \mathbb{R}^{1 \times 1 \times c_{\text{in}}}$ produces the output feature map. This represents a significant reduction in the number of parameters, by an order of k^2 where k is the kernel size, so with a 3×3 kernel the computational cost is reduced by a factor of ~ 9 (Howard et al., 2017; Sandler et al., 2018). As a result, this leads to a notable acceleration in the inference latency, while the accuracy hit stays acceptable for edge device use cases. In effect, depthwise-separable convolutions have served as a drop-in replacement for the classic convolution layer when reducing model size or latency is key.

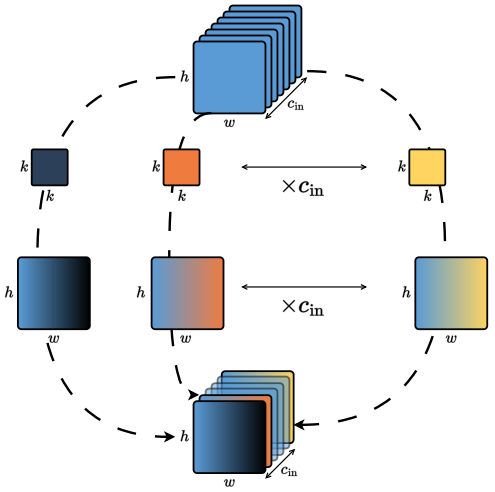


Figure 2.5: Depthwise convolution

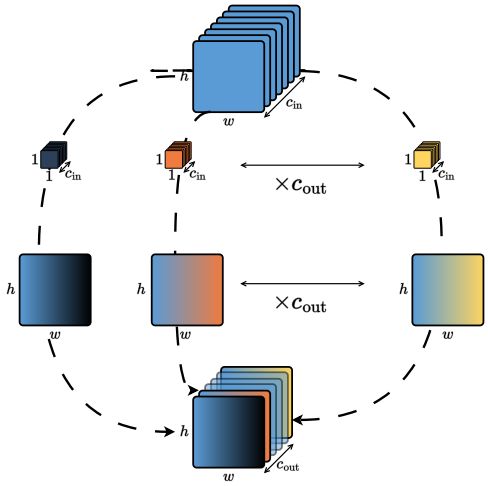


Figure 2.6: Pointwise convolution

MobileNetV2 (Sandler et al., 2018) proposes the inverted residual bottleneck. The residual bottleneck (He et al., 2016a,b; Xie et al., 2017) reduces the channel dimension with pointwise (1×1) convolution and expand it after the main convolution operation. The inverse residual bottleneck pattern expands the input channel dimension with a pointwise convolution, followed by the main depthwise convolution, and a final pointwise operation to reduce the channel dimension again. It also has a residual connection.

MobileNetV3 (Howard et al., 2019) builds on the MobileNetV2 block, adds modifications to some aspects like the nonlinearity used, and augments it with Squeeze-and-excitation (Hu et al., 2018). Using an automatic architecture search scheme, it finds more effective networks.

Many other efficient ConvNet designs have been proposed, for instance ShuffleNetV2 (Ma et al., 2018). EfficientNet (Tan and Le, 2019) studies model scaling using both the ResNet and MobileNet v1 and v2 architectures, demonstrating that depthwise-separable convolutions performance could be pushed even further, among other insights.

2.3 Structural re-parameterization

Structural re-parameterization is based on the idea of distinguishing between the train-time architecture and the inference-time architecture: an over-parameterized train-time model is converted to a functionally equivalent but smaller model for inference. As a result, this allows the train-time model to achieve a higher performance, with no impact on the efficiency of the deployed model.

ACNet (Ding et al., 2019) introduced the Asymmetric Convolution Block (ACB), which can be used as a drop-in replacement for the traditional CNN block. For instance, a 3×3 square convolution kernel is replaced with three parallel branches containing the 3×3 kernel, a 3×1 kernel and a 1×3 kernel. This over-parameterized train-time model achieves a better accuracy. For deployment, the inference-time model is obtained by fusing the three branches into a single 3×3 kernel (Figure 2.7), which performs a mathematically equivalent computation to the parallel branches. The overall model is therefore at equivalent latency and number of parameters as the original model based on CNN blocks, but with further performance gains.

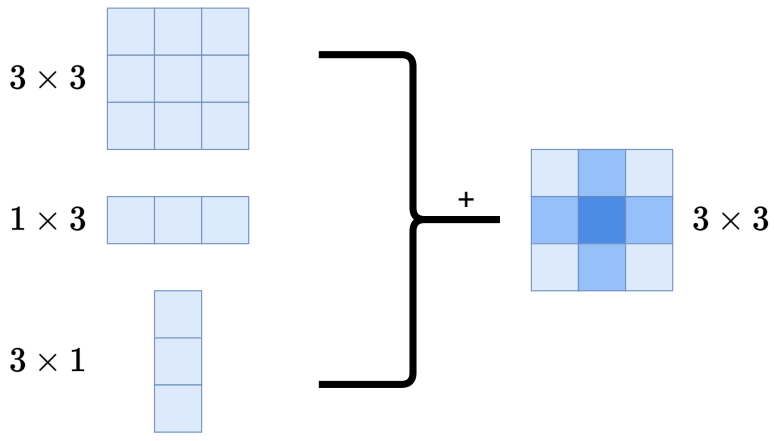


Figure 2.7: Fusing of convolutions for structural re-parameterization in the ACB block from ACNet (Ding et al., 2019)

The benefits of structural re-parameterization goes beyond simply providing further capacity

using a bigger model. ACNet (Ding et al., 2019) demonstrates that the over-parameterized train-time model is more robust to rotational distortions. Similarly, ExpandNet (Guo et al., 2020a) introduces a number of expansion strategies and analyzes their impact on the network's generalization capability. In particular, there is a reduction in gradient confusion (Sankararaman et al., 2020), which measures how correlated the gradients are between different data batches, leading to a faster and more stable training procedure and better results.

Diverse Branch Block (DBB) (Ding et al., 2021a) and RepVGG (Ding et al., 2021b) are other iterations in the structural re-parameterization family of methods, offering further re-parameterization strategies and performance improvements.

When it comes to efficient vision models, structural re-parameterization was used most notably in MobileOne (Vasu et al., 2023b), which focuses further on how this technique is best deployed in the low-parameter regime. This leads to the introduction of trivial over-parameterization branches, which are more suitable for small mobile-focused models. The advantages of effective structural re-parameterization are evident in this scenario, as this technique pushes the performance further while maintaining minimal latency for mobile device use cases.

2.4 Vision transformers

The transformer (Vaswani et al., 2017) architecture changed the NLP landscape since their introduction, with breakthrough performances achieved by many transformer-based large language models (LLMs) (Devlin, 2018; Wolf et al., 2020; Touvron et al., 2023).

Based on the self-attention mechanism, they were first introduced as high performance vision models in (Dosovitskiy et al., 2020) with the Vision Transformer (ViT). ViT was the first non-ConvNet architecture to beat the top performing ConvNets on ImageNet. Since, they have represented a paradigm shift, with the top vision benchmarks no longer exclusively dominated by ConvNets. They serve as very capable vision backbones enabling high performance in downstream vision tasks as well.

The self-attention mechanism takes as input a sequence of tokens. For vision, image patches are used, sometimes implemented as a convolution layer with stride equal to the kernel. The n patches are transformed using matrix multiplication into d-dimensional vectors q, k and v. The result for the whole input is three matrices $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$, respectively the query, key and value matrices.

The quantity $\mathbf{Q} \cdot \mathbf{K}^\intercal$ computes scores for each pair of tokens, representing the degree of attention to be paid to each pairwise interaction. Note that this quantity is not symmetric, meaning the score for $(\text{token}_i, \text{token}_j)$ and the score for $(\text{token}_j, \text{token}_i)$ are not necessarily close. After scaling (for gradient stability), a softmax operation transforms the scaled scores into probabilities. The value matrix is then modulated by these probabilities. This operation is

expressed as follows:

$$\text{Attn}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\cdot\mathbf{K}^{\mathsf{T}}}{\sqrt{d}}\right)\cdot\mathbf{V} \tag{2.1}$$

This process is the operation in one attention head, and the output of multiple heads is concatenated for the multi-head attention layer.

In the context of vision models, the general differences between ConvNets and ViTs are summarized in broad terms in (Khan et al., 2022) table 2.1. Figure 2.8 compares the receptive fields of both families of architectures, where multi-colored connections denote dynamic weights which vary depending on the input.

ConvNet receptive field: local, with fixed weights after training

ViT receptive field: global, with "weights" depending on inputs

Figure 2.8: Receptive fields in convolution and attention. Adapted from: (Adaloglou and Michels, 2023)

2.5 Efficient ConvNet-ViT hybrid models

To leverage the complementarities between ConvNets and ViTs, hybrid ConvNet-ViT models have been introduced. Examples include CoAtNets (Dai et al., 2021b), CvT (Wu et al., 2021), LocalViT (Li et al., 2021c), LeViT (Graham et al., 2021), BossNAS (Li et al., 2021a), and MaxViT (Tu et al., 2022b).

For edge AI models, the ConvNet-ViT hybridization has greater significance: in practice, efficient models still require a number of convolutional blocks to respect hardware constraints and to hit latency targets. As a result, efficient ConvNet-ViT hybrids have become an active area of architecture design, with different approaches to the hybridization such as MobileViT

Attribute	ConvNets	ViTs
Weights	Fixed kernel weights once trained	Weights are dynamic and vary based on input
Receptive Field	Local, grows larger as the resolution decreases along the depth of the network	Global
Inductive Bias	ConvNets rely on image-specific inductive biases (e.g., locality and translation equivariance)	Fewer assumptions, more general; usually more data-hungry to compensate for lack of inductive biases; higher capacity, can benefit from larger datasets
Edge Device Performance	The long-standing prevalence of ConvNets has made the convolution operator enjoy acceleration and wide support in many cases	Pure ViTs can require more memory (more parameters) and take more time to process, impacting latency

Table 2.1: Comparison of ConvNets and ViTs

(Mehta and Rastegari, 2021), MobileFormer (Chen et al., 2022), EdgeViT (Pan et al., 2022), MobileViTv2 (Mehta and Rastegari, 2022), and EfficientFormer (Li et al., 2022). A pattern has emerged, consisting of reserving the expensive self-attention layers to deeper parts of the network, where the resolution and consequently the number of tokens is lowest. Recently, some lighter re-designs of the self-attention mechanism have made it possible to have self-attention at earlier sections of the network (Shaker et al., 2023).

3 Neural Architecture Search

This chapter focuses on the core concepts of Neural Architecture Search, detailing its main components such as search spaces and search strategies.

The efficiency of the search procedure is of primary importance in NAS. We enumerate some prominent techniques to achieve higher efficiency.

Finally, we provide details about practical considerations in NAS, including benchmarks used to compare search methods and frameworks to perform NAS on user-defined search spaces.

Neural Architecture Search (NAS)

Neural Architecture Search (NAS) is the automatic design of neural network architectures. As a special case of AutoML (Hutter et al., 2019), it has two distinctive features in comparison with other AutoML subfields: the design spaces for architectures can be very large, and evaluating a solution is expensive and time-intensive.

3.1 Motivation

Deep learning has achieved spectacular success in numerous use cases and applications. The availability of training data and computing resources are among the most important contributing factors. Neural network (NN) architectures are also at the heart of this revolution: different components and architectural patterns heavily influence their performance.

In the past decade, there has been a steady stream of improvements made to neural network architectures in search for performance and efficiency gains. Some of these improvements were made to aspects of the macro-architecture, such as model scaling in EfficientNet (Tan and Le, 2019). Others instead focused on enhancing existing layers and operators, or designing new ones. For instance, self-attention (Vaswani et al., 2017) proved its impressive capabilities in Natural Language Processing (NLP) and gradually made its way to vision models (Dosovitskiy et al., 2020).

The majority of these architectural innovations have been the result of manual design by domain experts. New papers in the literature incrementally add to the general collection of architectural patterns, best design practices, base components and ways to arrange and compose them. Experimentation by the community of researchers and practitioners have pushed the performance level of these models year after year. This process of manually handcrafting architectures requires time, computing resources, and domain expertise. The impressive results are, as (Lu et al., 2019) states, “fruits of years of painstaking efforts and human ingenuity.”

With these challenges in mind, one of the earliest motivations for developing NAS methodologies is to automate the design step: using NAS as a tool to find completely new architectural innovations. NASNet (Zoph et al., 2018) and EfficientNet (Tan and Le, 2019) are examples of computer vision architectures found, in part or in whole, using NAS.

Gradually, additional applications for NAS have emerged, going beyond open-ended NN architectural research. Specifically, NAS is also instrumental in finding the best architectures for optimal performance in specific use cases. Automatic optimization is very useful if the application imposes hardware constraints, or if the task calls for optimizing accuracy as well as other objectives like energy efficiency or robustness.

In the FBNetV3 paper (Dai et al., 2021a), the authors point out that with the need for small and accurate models in a large number of real-time applications, there is an exponentially large number of ways to combine hardware constraints and architectures. Manual exploration is inherently limited in these situations.

One important research direction addresses the disconnect between designing state-of-the-art architectures trained on high-performance server GPUs, and edge devices where inference occurs in many cases. As TEA-DNN (Cai et al., 2019) showcases by comparing Pareto fronts, this is highly suboptimal. The paper goes further, and shows that due to differences between edge devices, it is advised to adapt the architecture to suit each hardware platform. NAS provides the methodology and the tools to perform this platform-aware optimization. Hardware-aware NAS (Benmeziane et al., 2021) has emerged as a prime example of the practical value of NAS for creating the best architectures for specific hardware.

In section 3.2, we will provide an overview of the main NAS components and the most prevalent methods used in the literature. Due to the costs associated with NAS, search efficiency is of paramount importance, and has been one of the key research questions. Methods to make the search more efficient are detailed in section 3.3. Section 3.4 is focused on practical aspects of NAS research and practice.

3.2 Core concepts and methods

Designing a NAS method necessitates addressing the following questions:

- *Which set of potential architectures are we interested in ?*

  This delimits the search space: the set of feasible architectures. The quality and size of the search space are crucial aspects.

- *How do we explore this space ?*

  Answering this question involves choosing a search algorithm. The No Free Lunch theorem (Wolpert and Macready, 1997) applied to NAS indicates that there is no universally superior algorithm for all NAS problems.

So far, these two questions are commonplace in black-box optimization problems. However, NAS belongs to a more specific subset of these problems, where the evaluation of one candidate solution is especially expensive: an instance of the considered architecture has to be trained and evaluated. With a substantial number of evaluations needed for most search algorithms to produce good results, the costs can add up and make the NAS process impractical and ineffective.

As a result, reducing the cost of architecture evaluations and/or the total number of evaluations is a central research question in NAS. It has implications for the search space definition and

the search algorithm selection. However, it also raises an additional question which receives a lot of attention in NAS literature:

- *How can the quality of a solution be estimated ?* A priori, the quality of the solution is the objective function we are maximizing. However, designing effective NAS methods requires finding an efficient way to make this quality assessment, which we will discuss in more detail in section 3.3.

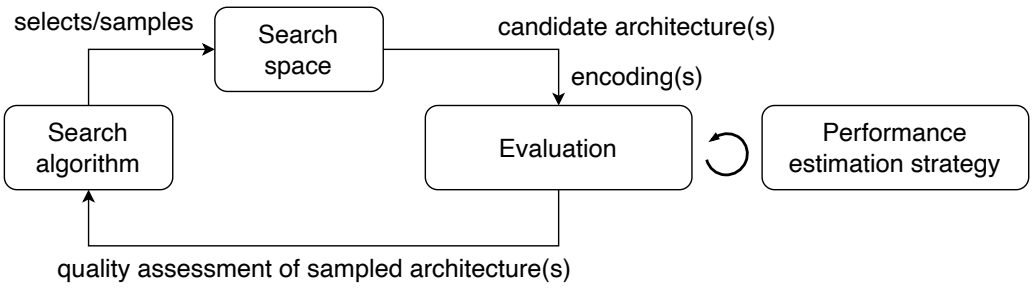


Figure 3.1: NAS components and optimization loop

3.2.1 Search space

The search space spans the set of feasible networks among which we aim to identify the superior network(s). It is generally described by a set of elementary components (e.g. layers, basic blocks of layers and operators), in addition to the ruleset which defines how these components connect to form the computation graph of a network.

Some search spaces focus on tuning architectural parameters of existing widely-used components, such as kernel size, number of channels and stride in convolutional layers (Zoph and Le, 2016; Baker et al., 2016). Other approaches diverge more significantly from the conventional designs. The search spaces found in NAS literature are very varied, and can be difficult to describe in a uniform way. (Talbi, 2021) is a survey which includes a taxonomy of search spaces.

To avoid the pitfalls of the curse of dimensionality and facing unreasonable search times, search space design often has to involve efficiently selecting the design variables and dimensions. Many methods have opted for more measured designs of the search space, striking a balance between its expressivity and the efficiency of the search.

As a result, a pattern has emerged in NAS search spaces, where the objective is to find the best cell or block. This low-level design is then stacked in a macro-architecture, usually a simple sequential layout. Since a succession of the same block design is also the layout followed by most manually designed models, this choice makes sense for designing tractable NAS

problems with the possibility of finding high quality architectures using available resources. Examples of this cell-based search space pattern include:

- FBNet (Wu et al., 2019), where the authors fix the high-level architecture and some blocks of the lower level, and allow the search algorithm to configure the internal blocks.
- NASNet (Zoph et al., 2018) searches for 2 cells: a normal cell (stride = 1) and a reduction cell (stride = 2). A key insight is that the low-level search space is independent of the depth of the network and the image resolution. As a result, the search is performed on a small dataset (CIFAR10), and the two micro-architectures are placed in a deeper fixed macro-architecture for the bigger dataset (ImageNet). The macro-architecture is a chain-like design where a reduction cells is placed before a succession of normal cells.
- DARTS (Liu et al., 2018b) is most widely recognized for introducing a successful method for one-shot or differentiable NAS, which we present in more detail in subsection 3.3.4. This is achieved using a continuous relaxation of the cell-based search space, resulting in a building block used to build the full architecture.

However, stacking the same building block has certain shortcomings, especially in the case of CNN-based vision models: the input resolution is gradually reduced using downsampling layers. The MNasNet paper (Tan et al., 2019) argues that, especially in scenarios where latency is important such as edge devices, stacking the same cell throughout the entire network is suboptimal. Instead, the authors subdivide the networks into sections, where input resolutions are fixed per section (i.e. between downsampling layers), and search for a layer design and the number of layers in each section. Effectively, this is a combined search in the micro-architecture and the macro-architecture, in a factorized or hierarchical search space. MobileNetV3 (Howard et al., 2019) uses the same search space as MNasNet with extra improvements and a way to tune it for mobile phone CPUs.

While the vast majority of NAS methods have traditionally focused on CNNs for vision applications, and especially image classification, there is no shortage of examples of other search spaces focused on other types of architectures and tasks (Tsai et al., 2020; So et al., 2019; Iqbal et al., 2023).

Encodings

Encodings are representations of the architectures in the search space, and their design has a bearing on how effective the search is (White et al., 2020). They depend on and influence every aspect of NAS, including the search space structure and the search algorithm. As design variables for NNs can be of different natures, including continuous, discrete and categorical variables, the encodings can be simple or mixed.

Another pivotal aspect is the overall adequacy of the encodings with the search space they

represent. It is important to make sure they represent the entire search space, and to avoid duplicate architectures represented by different encodings. Ideally, it is preferable to use a one-to-one mapping. There are many ways to create these representations (Talbi, 2021), ranging from fixed-length encodings (Zoph and Le, 2016; Loni et al., 2020), to variable-length encodings (Assunção et al., 2019). In widely-used benchmarks like NAS-Bench-101 (Ying et al., 2019) and NAS-Bench-201 (Dong and Yang, 2020; Dong et al., 2021), a fixed-length encoding is used to represent the adjacency matrix for the direct acyclic graph (DAG) which describes the architecture. The BANANAS paper (White et al., 2021a) compared a number of encoding schemes, and suggested path encoding as a superior way to encode solutions for NAS methods based on neural predictors (Wei et al., 2022).

For evolutionary algorithms, designing the encodings goes hand in hand with the design of the variation operators (crossover, mutation), and the effectiveness of the search depends on the quality of the encodings and operators. With search algorithms such as local search, it is important to ensure connexity: to guarantee that the optimal solution can be attained from any given starting point.

Scope of the search space

The search space can also include design elements not strictly related to the architecture of the networks. In certain instances, the search goes beyond architectural patterns. In that case, the problem is no longer strictly NAS, but joint optimization problems like hardware/architecture co-design, and NAS coupled with hyperparameter optimization (HPO).

NAS+HPO: Although at times treated as separate problems or subdomains, Hyperparameter Optimization (HPO) and NAS have much in common. Many black-box search algorithms, most notably Bayesian Optimization, can be applied to both. Many methods in the literature address both the architecture and the hyperparameters (Talbi, 2021), for instance through nested optimization where each architecture evaluation includes a hyperparameter optimization step, and sequential optimization where the best architecture found by the NAS algorithm goes through a hyperparameter optimization step to further improve quality (Rawat and Wang, 2019).

However, (Zela et al., 2018) points out that due to the high degree of interaction the architecture and the hyperparameters in terms of the model's performance, the most optimal performance is obtained with joint optimization:

- (Zela et al., 2018) casts the NAS problem as an HPO problem, with a number of categorical hyperparameters fully specifying the architecture. Then, both the architecture and hyperparameters are jointly optimized using a combination (Falkner et al., 2018) of Bayesian Optimization (BO) and Hyperband (Li et al., 2017).
- FBNetV3 (Dai et al., 2021a) uses an accuracy predictor, which takes as input a learned

embedding representing the network architecture, and the training recipe, i.e. a set of hyperparameters.

- CoDeepNEAT (Miikkulainen et al., 2024) uses neuroevolution to optimize both the architecture and the hyperparameters jointly.

Hardware/architecture co-design: In some use cases, it is possible to go even further than hardware-aware NAS, where hardware aspects are integrated as constraints or objectives in the optimization process. Widening the scope of the search space to include the hardware configuration is a further opportunity to extract more performance from the same hardware. Configurable hardware platforms with high degrees of design freedom, such as FPGAs, can enable the joint optimization, or co-exploration, of both the hardware and architecture spaces (Jiang et al., 2020). ASICs such as EdgeTPU (Seshadri et al., 2022) have also enabled this mode of joint co-design in NAHAS (Zhou et al., 2021).

3.2.2 NAS as an optimization problem

Let us consider:

- $\mathscr{S}$ is the search space, i.e. the set of feasible architectures considered.
- $a \in S$ is an architecture from the search space, which sets the blueprint of components and connections defining how a network is defined.
- m_a is a model instance of a: it follows architecture a, and it has a number of weights which can be trained using backpropagation and gradient descent.
- f is the objective function. It associates each architecture with a value which denotes its quality, and is the quantity the NAS process maximizes.
- p_{valid} is the performance of a model on the validation data. For instance, for a classification model, this could be the accuracy on the validation dataset
- D_{train} and D_{valid} are respectively the training and validation datasets

In the single objective case, the NAS problem can be formulated as follows:

$$a^* = \underset{a \in \mathscr{S}}{\arg\max}\, f(a) \tag{3.1}$$

In most NAS problems, there are in fact two nested optimization loops. The evaluation of f at architecture a involves training the model m_a on D_{train} and testing it on D_{valid}. This optimization loop is based on backpropagation gradient-descent.

$$f(a) = p_{\text{valid}}(\hat{m}_a) \quad \text{s.t.} \quad \hat{m}_a = \underset{m_a}{\arg\min}\, \mathscr{L}_{D_{\text{train}}}(m_a) \tag{3.2}$$

This inner optimization loop is not applicable to all NAS approaches, for instance zero-shot NAS methods are specifically designed to avoid it (subsection 3.3.3). One-shot NAS methods combine both loops in a bi-level optimization problem solved with gradient-descent, after relaxing the outer loop in continuous space (Liu et al., 2018b).

In the multi-objective setting, the optimization pertains to a vector-valued objective function:

$$\mathbf{f}(a) = (f_1(a), ..., f_m(a)) \quad \text{s.t.} \quad m \geq 2 \tag{3.3}$$

This setting is most often associated with hardware-aware NAS, where additional objectives represent hardware metrics: inference latency, FLOPs, the number of trainable parameters, energy consumption, memory footprint...

With conflicting objectives, no single solution is the best. Instead, the aim is to find the Pareto-optimal set of solutions, which contains all non-dominated solutions. Intuitively, a non-dominated solution cannot be improved with respect to one objective without negatively affecting one or many others.

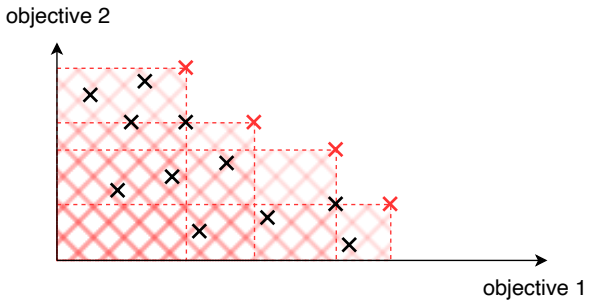


Figure 3.2: Representation of the Pareto optimal set, containing the non-dominated solutions (red), supposing a maximization problem. The cross-dashed area is the area dominated by the Pareto optimal solutions. Each of the black points are dominated by at least one solution from the Pareto-optimal set

3.2.3 Major Search algorithms

In NAS, the objective function does not have a closed-form expression which computes its value from the architecture encoding. It can only be evaluated at points, with no access to first or second order derivatives. Blackbox optimization, or derivative-free optimization, is the most suitable way to tackle similar problems (e.g. certain NP-hard problems, or simulation-based

problems).

Reinforcement learning (RL)

An RL agent, or controller, takes actions and receives rewards which affect its future policy. In this instance, the agent samples an architecture from the search space, and its reward is derived from the quality of this architecture after it is evaluated. The reward guides the controller towards better policies, gradually leading it to generating high-quality architectures. The model used in many RL-based NAS methods is a recurrent neural network (RNN), which generates the architecture sequentially.

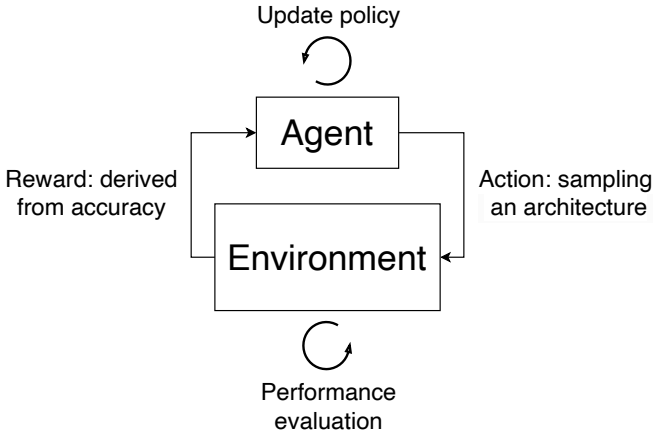


Figure 3.3: The reinforcement learning (RL) process in the context of NAS. The agent samples an architecture based on its policy, the architecture is evaluated, and the reward is calculated from its quality assessment affects the agent's future policy

RL has been used as the search algorithm in many NAS methods, including some of the very earliest (Zoph and Le, 2016). That trend has continued (Zoph et al., 2018), including when it comes to one-shot methods (Pham et al., 2018; Miao et al., 2022).

RL has also been used in hardware-aware NAS settings, for instance in MnasNet (Tan et al., 2019), which incorporates the inference latency in the objective function. MONAS is a multi-objective RL-based method which incorporates the additional objective through scalarization. The following expression models the reward the RL agent receives, with additional constraints defined using thresholds:

$$R = \alpha * \text{accuracy} + (1 - \alpha) * \text{energy} \tag{3.4}$$

One of the main limiting factors for RL usage in NAS is in terms of sample efficiency: certain methods required a very high number of evaluations needed (Real et al., 2019b).

Evolutionary algorithms (EAs)

Evolutionary algorithms (EAs) are one of the most important classes of blackbox optimization methods, used to solve a large range of problems. The inspiration comes from biological evolution. They are population-based metaheuristics (Talbi, 2009), where a population of solutions is evolved with the aim of improving each generation's performance as measured by the objective function. This evolution is accomplished using variation operators, namely crossover and mutation, which produce new children for the next generation, propagating good genetic traits and adding genetic diversity.

NAS literature features many EA-based approaches, with a heavy focus on CNN architectures (Real et al., 2017, 2019b). (So et al., 2019) is an example of EA-based NAS applied for natural language processing (NLP) evolving a transformer-based architecture, showcasing the versatility of this class of methods.

Among the most notable examples of EAs in NAS is LEMONADE (Elsken et al., 2018). A cleverly-designed mutation operator is used to benefit from approximate network morphisms reducing evaluation costs for the networks in the new generation.

NARS is the search method which produced the FBNetV3 architecture (Dai et al., 2021a). It performs predictor-based evolutionary search in cpu minutes. The predictor is pretrained using architecture statistics and used to predict the accuracy of both the architecture and some hyperparameters. Since inference using the predictor is a fast and cheap alternative for the evaluation of networks, the evolutionary algorithm found satisfactory results in CPU minutes.

However, custom or complex search spaces can require careful implementation of the crossover and mutation operators. These operators can have a big impact on the success of the search. The required number of evaluations for many EA implementations can also hinder practical usability for NAS in particular, since the cost of evaluations can be exceptionally high.

Random search (RS)

Random search (RS) coupled with early-stopping has been shown to be a competitive search algorithm for hyperparameter optimization, with the additional benefit of supporting massive parallelization (Li et al., 2020a). In the NAS context, there have been comparisons between random search and other algorithms such as RL and EAs (Liu et al., 2017; Real et al., 2019a). While there was an advantage for RL and EA in these instances, RS was still competitive, with low differences in terms of accuracy on CIFAR-10 after searching for a certain budget. For one-shot NAS in particular (subsection 3.3.4), many methods rely on variants of random search (Bender et al., 2018; Brock et al., 2017; Li and Talwalkar, 2020; Guo et al., 2020b).

Bayesian optimization (BO)

Bayesian optimization (BO) (Mockus, 1974; Garnett, 2023) is especially well-suited for blackbox optimization problems with expensive evaluations. With its sample efficiency, it can achieve good search results with fewer evaluations than competing methods. It is a very prominent method for hyperparameter optimization (Yu and Zhu, 2020). With the close proximity of NAS and HPO, it is no surprise that many works have approached the NAS problem using variants of BO. We introduce BO with more details details about BO in chapter 4, but to illustrate its potential and place in the NAS landscape, we mention a few examples of its usage.

- BO is most often associated with Gaussian Processes (GPs), which rely on kernels and distance metrics. In NASBOT (Kandasamy et al., 2018), the authors define a distance metric in the architecture space based on optimal transport.
- BANANAS (White et al., 2021a) thoroughly studies the bayesian optimization design choices for application in NAS settings. More specifically, many predictive models (feed-forward network (FFN), graph convolutional network (GCN), variational auto-encoder (VAE)), acquisition functions and acquisition function optimization methods are compared. This leads to the design of a fast BO-based NAS method, based on a feed-forward predictor coupled with a path encoding scheme.
- TEA-DNN (Cai et al., 2019) is a multi-objective BO-based NAS approach, targeting the Time, Enegy and Accuracy objectives and measuring the metrics on the target hardware (edge devices) directly. It uses Gaussian Processes (GPs) and searches for a Pareto optimal set.
- FlexiBO (Iqbal et al., 2023) makes an important observation: not all objectives for hardware-aware NAS cost the same, especially in terms of evaluation time. For instance, measuring the inference latency would require averaging the time it takes for a few hundred forward passes with no need for prior training. This makes the cost for this objective orders of magnitude lower than the accuracy objective. FlexiBO takes this into consideration at the level of the acquisition function, making decisions about the next evaluations in light of information acquired using evaluations of the cheap objectives.
- RA-AutoML (Yang et al., 2020) proposes a NAS platform geared towards integrating hardware limitations as constraints for the BO-based optimization process, or constraint-aware BO.
- SpArSe (Fedorov et al., 2019) integrates various concepts we can find in NAS, including multi-objective bayesian optimization and network morphisms (see 3.3.2). It is designed to find CNN architectures designed specifically for microcontroller units (MCUs).

This overview of blackbox optimization methods used for NAS gives a partial view of the search strategies employed in the literature. In fact, due to the high cost associated with solution

evaluation, a lot of efforts have targeted the search efficiency matter in particular. This has led to the emergence of a number of search acceleration solutions. The next section 3.3 details the approaches used in conjunction with blackbox optimization algorithms, as well as zero-shot and one-shot NAS which have emerged as answers to this efficiency bottleneck.

3.3 Search efficiency

Pioneering NAS methods demonstrated its potential for discovering high quality architectures. However, these early methods, whether based on reinforcement learning (Zoph and Le, 2016; Zoph et al., 2018) or evolutionary algorithms (Real et al., 2017, 2019b) were very resource-intensive. Many of them required upwards of 2000 and sometimes 3000 GPU days (Chen et al., 2021).

In practice, this was achieved with hundreds of GPUs concurrently training and evaluating a large number of potential architectures. This clearly limits the accessibility of NAS, and increases the potential losses incurred during unsuccessful search runs (e.g. when the designed search space does not contain high quality architectures). The high costs are far from the only motivation for focusing efforts on optimizing the search efficiency, as the environmental impacts on energy and water resources is a key concern for responsible AI (Ren and Wierman, 2024).

As this efficiency problem is the main challenge in NAS, it is a research question which sparked numerous ideas and efforts to address it. We categorize these ideas into four main classes:

- Optimising the search space design to be better-suited for quick search times
- Estimating the expensive objective function using a predictive model
- Replacing the original objective function with a cost-effective proxy
- Gradient-based methods

3.3.1 Search space efficiency

The design of the search space is of paramount importance, and has to balance two important aspects:

- **Expressiveness:** the search space has to contain enough interesting architectures to justify performing the search.
- **Efficiency:** a big search space has an impact on the search time needed to achieve good results. In practice, with a hard limit on the search time and training resources, this in

turns means the search ends up with a relatively low-quality solution by the time the resources are exhausted.

This is analogous to the exploration-exploitation dilemma: on one hand, we can inject more of our prior domain knowledge, such as knowledge of promising architectures and patterns. This can help reduce the size of the search space and make NAS more practically feasible. On the other hand, this raises the requirement for human expertise, and might introduce bias: there could be unconventional designs which lead to better-performing solutions.

This context explains certain ideas and trends to increase the chances of success using the search space, among which:

- Using domain knowledge or previous NAS experience to streamline the search:
  - Making sure certain well-performing hand-crafted architectures are included in the search space, and giving them special importance to influence the search. For instance, they can be seeded directly in the initial population in an evolutionary algorithm NAS method (Lu et al., 2019; So et al., 2019)
  - Removing ineffective designs and operators. For instance, (Liu et al., 2018a) partly reuses the search space in (Zoph et al., 2018), but it culls design elements which were found unproductive.
- Limiting the scope of the search space only to cells as discussed in subsection 3.2.1 and stacking them sequentially, since many successful hand-crafted architectures similarly follow a sequential layout.
- Removing redundancies: if the mapping between encodings and architectures is not bijective, search resources might be wasted on genotypically distinct but identical architectures.
- Using a progressive search scheme: (Liu et al., 2018a) gradually increases the complexity of the search domain. As an SMBO method, the surrogate model (reward predictor) begins learning during the initial low-complexity stages and subsequently guides the search in later stages, all while continuing to learn.

3.3.2 Objective function estimation

The most common idea to improve search efficiency is to replace the expensive objective function with a lower-cost estimation. Multiple ways have been proposed with varying degrees of complexity. Each offers a different tradeoff between the speedup and the quality of the estimation, which in turn affects the quality of the search results.

Low-fidelity training

The most obvious way to estimate the performance evaluation is to replace it with a lower-quality version. For instance, truncating the training data will significantly lower the training costs. Alternatively, capping the number of training epochs is also an easy way to achieve this.

However, the impact on the quality of the approximation is significant. An experiment was conducted in (Zela et al., 2018), where the Spearman rank correlation coefficients are measured between accuracy scores at different points in the training process. The results show that the rank correlations degrade heavily when the training time budgets are very disparate. If we consider the 3-hour validation accuracy as the baseline for the fully trained models, the rank correlations are reported in table 3.1.

Budget	400s	20m	1h
Correlation with final ranking	0.05	0.64	0.86

Table 3.1: Rank correlations with the 3-hour ranking per training budget. Source: (Zela et al., 2018)

The same phenomenon is reported in the FBNetV3 paper (Dai et al., 2021a). According to the reported metrics, taking the ranking at 150 training epochs as baseline, the rank correlations are at first inconsistent and low (less than 0.25), and begin gradually increasing at around the 25 epoch point, reaching 0.90 after 100 epochs, i.e. 66% of the full training time.

Consequently, (Zela et al., 2018; Dai et al., 2021a) clearly demonstrate that a very low training epoch cap results in a heavily altered architecture ranking with a low correlation with the true ranking. This also adds another complication: the point at which the correlations are good enough depends on many factors, including the search space and the considered objective function. For instance, altering the training dataset might influence them.

An interesting solution was used in NARS to find FBNetV3 (Dai et al., 2021a): dynamically determining the appropriate early-stopping point. It uses the first iteration of the search to determine the epoch at which the correlation between the true rankings and those produced by the lower-fidelity training is above a threshold (e.g. 0.92). All subsequent evaluations of solutions in the NAS process are terminated when that point is reached.

Favorable weight initialization

The main idea is to avoid training candidate networks entirely from scratch.

Reusing weights from trained networks:

Network morphisms (Chen et al., 2015; Wei et al., 2016) are function-preserving operators in the space of neural architectures. Using these operators yields a child network which can be

wider or deeper and remain functionally equivalent to its parent network. The child network then needs far fewer epochs to be trained.

This idea inspires the introduction of approximate network morphisms (Elsken et al., 2018), which is a relaxed definition of network morphisms allowing to decrease the model size (e.g. removing or pruning layers, substituting convolutions with depthwise-separable convolutions, etc...). These operators serve as mutations in the context of an evolutionary algorithm, and allow the children network to inherit weights from parent networks, reducing their training cost significantly.

Such an approach can also be found in other works, such as SpArSe (Fedorov et al., 2019), which uses them in conjunction with multi-objective bayesian optimization to adapt CNN networks for resource-limited microcontrollers. The morphisms are only applied starting from a reference configuration, as opposed to a population of networks in LEMONADE's EA (Elsken et al., 2018).

Generating weights:

The idea is to assign weights for a candidate network directly, making it possible to cheaply assign weights to new candidate networks with little to no subsequent training needed.

Using a neural network trained to generate weights for another is the purpose of hypernetworks (Ha et al., 2016). An interesting example of this idea for NAS is SMASH (Brock et al., 2017). The authors create a separate network, HyperNet, trained to predict the weights of any untrained network in the search space. As a result, the cost of training a new network is reduced to performing an inference with HyperNet, which is orders of magnitude cheaper.

Depending on the structure of architectures in the search space, it is possible to specialize the hypernetwork for better quality weights. For instance, GHNs (Zhang et al., 2018) use graph neural networks.

Accuracy prediction

For hyperparameter optimization, the learning curve can be modelled to enable techniques like early stoppping by providing future performance predictions (Domhan et al., 2015; Klein et al., 2022). Similarly, it is also possible to use a predictive model as a surrogate for the expensive objective function for NAS.

An early example in NAS can be found in (Baker et al., 2017), which uses regression models to predict the final performance of candidate networks, leading to significant speedups in total search time.

FBNetv3 (Dai et al., 2021a) uses a fully-connected neural network which receives an architecture encoding and a number of hyperparameters (a "training recipe") and predicts the accuracy. A pretraining scheme is used to improve its predictive capabilities.

DPP-Net (Dong et al., 2018) uses a recurrent neural network (RNN) to predict the classification accuracy. This choice notably can handle variable-size inputs.

3.3.3 Zero-shot NAS

The most aggressive way to accelerate the search is to sidestep the evaluation altogether. This approach has resulted in a family of methods collectively referred to as zero-shot NAS.

In contrast to accuracy prediction methods, zero-shot NAS methods do not rely on a model to predict the objective function. Instead, they propose specially designed scores which correlate with well-performing networks. Crucially, these scores can be quickly and easily calculated on untrained networks.

Many zero-shot NAS methods have been proposed in recent years. Many of them showcase good comparative performance on NAS benchmarks with walltimes orders of magnitude quicker than most NAS methods. They also offer interoperability with existing NAS methods, providing a low-cost proxy for the quality of solutions without requiring other changes, as long as the scores are suitable for the search space.

At the core of these methods are the previously mentioned scores:

- NAS Without Training, or NASWOT (Mellor et al., 2021), computes a kernel matrix based on the overlap between the patterns of ReLU activations. Well-performing architectures can be identified as the kernel matrices are more distinct for different inputs.
- TE-NAS (Chen et al., 2021) take a theoretical approach to derive two metrics that can be measured on untrained networks, using characteristics of their input space and their neural tangent kernel. The authors demonstrate that these metrics are good indicators of future performance for a number of vision models.
- More rudimentary zero-shot metrics have been shown to sometimes match or even outperform early sophisticated scores, e.g. the number of trainable parameters (White et al., 2022).
- A more recent method dubbed ZiCo (Li et al., 2023) uses a proxy based on statistics of gradients using many samples. Its authors show that this proxy outperforms simple and more complex earlier proxies.

3.3.4 Differentiable NAS

The objective behind differentiable NAS is finding a way to apply gradient-based optimization methods to optimize neural architectures. Using a differentiable representation of the search space, gradient information can guide the exploration and the search for the best solution.

This approach unlocks a significant speedup in comparison with black-box NAS methods (RL, EAs, SMBO...). As a result, differentiable NAS methods stand apart from the rest of NAS methods in terms of required search time, reducing it from thousands of GPU hours for some methods to GPU days (Liu et al., 2018b). This speed comes at the price of certain limitations and challenges detailed further in this section.

NAO (Luo et al., 2018) uses an RNN-based encoder-decoder to move from the discrete architecture space (discrete string tokens) to a differentiable representation space. This enables gradient-based optimization in the representation space, at the conclusion of which the best solution is decoded back to the discrete architecture space.

The most ubiquitous class of differentiable NAS methods goes a step further by unifying the architecture search process and the training of individual solutions. This is achieved with a supernetwork, also referred to as a one-shot model. It is represented by a supergraph whose subgraphs are the networks of the search space.

The supernetwork idea enables weight-sharing: the subnets which share common edges also in effect share the same weights. As a result, training the supernetwork assigns weights to all the subnets. Evaluating each subnet therefore becomes an easy endeavour, since it can inherit the weights from the supernetwork and the training step for it is either reduced very significantly or skipped entirely. Only needing to train the supernetwork with the shared weights is the explanation for the significant speedups demonstrated by this class of NAS methods.

DARTS (Liu et al., 2018b) is widely credited with pioneering this NAS strategy, using gradient descent and a particular form of weight-sharing to search for cells. DARTS introduced a continuous relaxation of certain design choices required to specify an architecture, for instance choosing one operator among many. Instead of choosing a particular operator, DARTS considers them at the same time, assigning each an architectural weight. The optimization is based on gradient-descent and is used to optimize both the operator weights and the architectural weights. At the end of the optimization process, the operator with the highest score is retained, and its internal weights will have already been set. (Figure 3.4)

DARTS has kickstarted the proposition of many methods to search for architectures in similar ways. There are differences most notably in the method for training the supernetwork. Examples include ENAS (Pham et al., 2018) and SNAS (Xie et al., 2018).

Depending on the implementation, hardware can be a limiting factor on how large or complex the supernetwork is, in turn constraining the size and complexity of the search space that can be used. This is the case when all the supernetwork weights need to be in memory during training. ProxylessNAS (Cai et al., 2018) avoids saturating the GPU VRAM as the authors devise a strategy to avoid loading the entire supernetwork weights: an edge is either used (therefore loaded) or ignored following the value of a binary variable, whose probabilities are learned in the process. As a single path is traced through the supernetwork at each time, and as a result

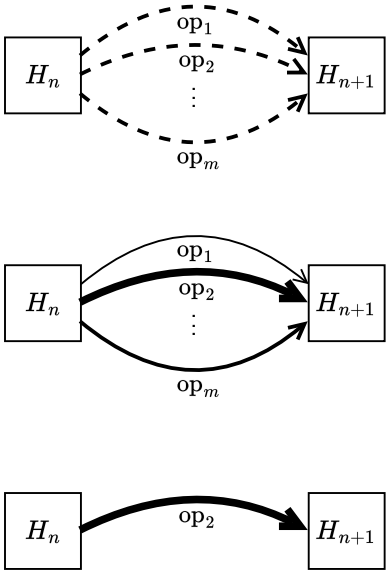


Figure 3.4: Illustration of DARTS with the continuous relaxation of the choice of operator between two latent representations H_n and H_{n+1} **Top:** The starting point is a continuous relaxation of the candidate operations. **Center:** The training procedure affects the architectural weights or mixing probabilities. Here, op_2 emerges as the most probable operation. **Bottom:** The final architecture is deduced.

the memory requirements are reduced significantly.

While these methods have demonstrated their success in multiple scenarios, especially considering their relatively cheap costs, there are a number of drawbacks associated with them.

An obvious limitation is the inflexibility stemming from limiting the searchable space to the subnetworks. In recent years, there have been efforts to alleviate these concerns, for instance with an automatic generation of supernetworks as described in BootstrapNAS (Muñoz et al., 2021).

Another major problem is the co-adaptation, or weight-coupling, which manifests itself when training the supernetwork (Xie et al., 2018; He et al., 2021). This can skew the relative ranking of the subnets (Yu et al., 2019). Path dropout (Bender et al., 2018) and random sampling (Brock et al., 2017) are possible answers to alleviate this problem.

3.4 NAS benchmarks and frameworks

3.4.1 NAS benchmarks

As (Lindauer and Hutter, 2020) explains, using the accuracy scores on datasets like ImageNet (Deng et al., 2009) and CIFAR-10/CIFAR-100 (Krizhevsky et al., 2009) does not provide an accurate understanding of how NAS methods compare. While these scores might indicate which resulting architecture outperforms the others, they give little insight about the search strategies which produced them.

This is due to potential differences in search spaces, in training techniques and most importantly in available resources. Other aspects can also introduce noise in the results, including training pipelines and hyperparameters. Consequently, it is difficult to separate which results can be attributed to the search method itself, and which are due to secondary factors.

To advance NAS research and impact, NAS benchmarks were introduced as a way to enable comparisons in a principled way. They provide a consistent framework in which approaches can be compared fairly, and eliminate the previously mentioned elements where noise and differences can distort the results. Using a benchmark does not guarantee that all the necessary criteria for reproducibility is met (Lindauer and Hutter, 2020), but their presence has enabled fast iteration and steadily ongoing progress in the NAS field.

NAS benchmarks have other notable advantages which explain how central and important they are. One very important contribution is enabling to test and iterate on NAS methods without performing the costly architecture evaluations: tabular benchmarks can be queried to get the results of the evaluation data instantly.

This unlocks the possibility of focusing on search strategies without worrying about prohibitively large training costs. It also abstracts away certain practical aspects which can heavily impact results:

- Orchestrating the training and evaluation of networks on computation clusters
- Ensuring consistency of the training process between different candidate networks (for instance training on a 6-GPU or an 8-GPU node with different batch sizes)
- Handling software dependencies and accelerator hardware compatibility
- Performing multiple evaluation runs to obtain statistically significant comparisons between different candidate networks

These aspects are important when NAS is performed on new search spaces, for new tasks or for new optimization objectives. However, before launching costly and unpredictable experiments, it is useful to confirm that the search method is effective using the benchmarks.

Over the years, multiple NAS benchmarks have been introduced. The most popular ones are focused on models for image classification:

- **NAS-Bench-101** (Ying et al., 2019) was the first publicly available benchmark made for NAS research. It contains 423k CNN architectures trained and tested on CIFAR-10. Its search space is cell-based, with each solution described by a direct acyclic graph (DAG). It also reports multiple details about the evaluation procedure, including training, testing and validation accuracies, training walltime, model size... Multiple training runs were performed, and the accuracy metrics were evaluated at multiple stages of the training process. With its large size and its incompatibility with weight sharing methods, other benchmarks have been subsequently proposed.
- **NAS-Bench-1shot1** (Zela et al., 2020a) leverages NAS-Bench-101 to propose a benchmarking framework with support for supernetwork based NAS (one-shot NAS). It proposes small (6k), medium (29k) and large (363k) search space variants.
- **NAS-Bench-201** (Dong and Yang, 2020) has a cell-based search space of 15.6k architectures tested on CIFAR-10, CIFAR-100 and ImageNet16-120 (16x16 images and 120 categories). Cells are also represented by DAGs with operations from a predefined operation set. Subsequently, NATS-Bench (Dong et al., 2021) includes two search spaces: NAS-Bench-201 dubbed the architecture topology space, and a new search space where the number of channels is the focus, dubbed the architecture size space. Along with a unified interface to both search spaces, NATS-Bench expands the accessible information about the architectures, including FLOPs, as well as training details like the loss values. It also features snapshots at different points of the training process.
- **NAS-Bench-301** (Zela et al., 2020b): The authors argue that relying on tabular benchmarks like NAS-Bench-101 and NAS-Bench-201 has limited the scope of NAS research to small search spaces. This is a fundamental limitation for tabular benchmarks, the creation of which requires training and testing all their architectures multiple times, under the same conditions. Instead, they introduce surrogate benchmarks, and argue they can effectively replace tabular ones for large search spaces: a regression model can be used as a surrogate for the quality evaluation of the networks in the search space, opening the door for arbitrarily large search spaces, provided that the regression models are of sufficient quality. They test a number of regression models, and find that it is possible to model the performance of networks in a large search space, possibly better than a tabular benchmark. They introduce NAS-Bench-301, containing 10^{21} possible architectures.

Other benchmarks have targeted other tasks, both vision tasks beyond image classification, and other deep learning application domains.

- **NAS-Bench-ASR** (Mehrotra et al., 2021) is a speech recognition-focused NAS benchmark. It includes 8.2k models trained on the TIMIT Acoustic-Phonetic Continuous

Speech Corpus (Garofolo et al., 1993).

- **NAS-Bench-NLP** (Klyuchnikov et al., 2022) is focused on RNNs for Natural Language Processing (NLP). It provides 14k architectures evaluated on language modeling tasks.

Some benchmark are focused on cross-task NAS: evaluating models on a diverse set of tasks.

- **TransNAS-Bench-101** (Duan et al., 2021) focuses on NAS for multiple vision tasks, by providing performance evaluations of vision models in image classification, pixel-level predictions like semantic segmentation, regression tasks and self-supervised learning tasks. For vision models especially, transfer learning (Tan et al., 2018) is very relevant, as many solutions start with backbone models trained for image classification on ImageNet.
- **NAS-Bench-360** (Tu et al., 2022a) focuses on vision and non-vision tasks, ranging from predictions related to DNA sequences, to classifying sound events or predicting the final state of a fluid. It provides the tabular performances of NAS-Bench-201 architectures on the NinaPro (Atzori et al., 2012) hand movement classification dataset, and the Darcy Flow (Li et al., 2020b) for PDEs.

With the emergence of hardware-aware NAS, there is a need for a specialized benchmark. **HW-NAS-Bench** (Li et al., 2021b) extends NAS-Bench-201 (Dong and Yang, 2020) and FBNet (Wu et al., 2019) search spaces by providing measured or estimated hardware metrics related to their architectures on different edge devices.

- *Metrics:* latency and energy (not both metrics are available for every device)
- *Devices:* NVIDIA EdgeGPU Jetson TX2, Raspberry Pi 4, EdgeTPU, Pixel 3, ASIC-Eyeriss

3.4.2 NAS frameworks

Benchmarks are highly important for NAS research, especially for the purposes of developing efficient search methods. However, just as automatic differentiation-based frameworks (Paszke et al., 2019; Abadi et al., 2016; Bradbury et al., 2018) have enabled unprecedented progress in deep learning adoption, software frameworks are also needed to democratize NAS usage. The objective is to enable users to find high quality model architectures for real-world deep learning use cases. NAS software needs to be practical, easy-to-use, efficient in its resource usage, and of course publicly available.

There are many choices for software tools and frameworks to perform NAS. General purpose blackbox optimization libraries can be useful, including evolutionary algorithm libraries (Blank and Deb, 2020; Gad, 2023; Cahon et al., 2004) or bayesian optimization libraries (Balandat

et al., 2020). Depending on the choice of the search strategy, the search space and the software tool, additional elements might be required, such as custom operators for EAs or distance measures for BO with Gaussian Processes. Many tools allow this level of customization, for instance PyMoo (Blank and Deb, 2020) enables creating custom variation operators.

NAS is particular case of AutoML (Hutter et al., 2019) focused on the architectures of neural networks. Many AutoML tools have been released over the years. Auto-sklearn (Feurer et al., 2015, 2020) can be used to automate the selection and tuning of sklearn (Pedregosa et al., 2011) estimators. AutoGluon (Erickson et al., 2020) has specific APIs to perform AutoML for tabular data, time series and multimodal data (using a deep learning model zoo).

For NAS in particular, some of the more general AutoML tools can still be used, for instance Google Vizier (Golovin et al., 2017) is designed as general-purpose blackbox optimization tool, useful for tuning hyperparameters but also for NAS. This flexibility can come at the price of more requirements which need to be implemented by the user, including handling the search space, the encodings, and the evaluation procedure.

More specific NAS libraries provide tools to design new search spaces, mainly using predefined blocks which can be chained together or combined to delimit the domain of the search space. For instance, Auto-Keras (Jin et al., 2019) uses a similar API to the Keras (Chollet et al., 2015) functional API to specify architecture search spaces. Another example is NNI (Microsoft, 2021), which includes a powerful set of tools enabling users to perform NAS, hyperparameter tuning and model compression. NNI features specific high-level and low-levels APIs to enable the definition of *model spaces* (i.e. search spaces). Once model spaces and evaluators are defined, the user can select from a list of search strategies, including:

- *multi-trial algorithms:* grid search, random search, regularized evolution (Real et al., 2019b), reinforcement learning (Zoph and Le, 2016)

- *one-shot algorithms:* DARTS (Liu et al., 2018b), ENAS (Pham et al., 2018), ProxylessNAS (Cai et al., 2018)

A number of NAS frameworks are more specific to particular spaces and use cases, such as searching for capsule neural networks (Marchisio et al., 2020, 2022).

For interoperability with general deep learning architectures implemented using PyTorch or other frameworks, PyGlove (Peng et al., 2020) provides a way to make these implementations mutable, which makes it easier to construct search spaces with existing code as a starting point. It is not a NAS tool, but it can be a very useful addition for a NAS pipeline based on a general-purpose blackbox optimization engine.

3.4.3 Lightweight vision datasets in NAS

As discussed in subsection 3.4.1, vision models have received the greatest deal of attention from NAS benchmark designers. This is also the case in NAS methods in general, as most evaluations are performed on vision datasets.

ImageNet-1k (Russakovsky et al., 2015) occupies a central place in this context, but due to its size, with 1.2M 224x224 training images across 1000 classes, it can only be used as a training set by one-shot NAS methods.

For other NAS approaches where a large number training processes are performed, smaller datasets need to be used during the search. Afterwards, the best performing architectures are trained from scratch on ImageNet to evaluate their performance against other image classification models in general. This can be considered a performance estimation strategy as well, since the performance on CIFAR-10 or CIFAR-100 (Krizhevsky et al., 2009) is being used as a proxy for performance on ImageNet.

CIFAR10 and CIFAR100 are used by many approaches, as evidenced also by their ubiquitousness in NAS benchmarks. Their size is much more manageable, as they contain 50k 32x32 images across respectively 10 and 100 classes.

In (Chrabaszcz et al., 2017), the authors show that it is possible to construct smaller versions of ImageNet with downsampled images, which maintain its characteristics with regard to hyperparameter optimization.

This line of reasoning naturally extends to NAS. For instance, NAS-Bench-201 goes beyond evaluation metrics for CIFAR10 and CIFAR100. It also provides them for ImageNet16-120, a subset of ImageNet16 (Chrabaszcz et al., 2017) with 120 classes.

We calculated the Spearman rank correlations between the accuracies in NAS-Bench-201 for the three datasets.

- CIFAR10 and CIFAR100 showcase a high degree of similarity in terms of network rankings, with a Spearman rank correlation of 0.98.
- Correlations with the ImageNet16-120 ranking are respectively 0.95 and 0.96 for CIFAR-10 and CIFAR-100. While these are high correlations, the differences are large enough to matter for many search algorithms. These differences are also presumably larger for the full ImageNet dataset with higher resolution images and many more classes.

Consequently, estimating performance on ImageNet-1k using CIFAR-10 and CIFAR-100 as proxy datasets introduces unnecessary noise to the objective function estimation. This is especially important because there are subsets of ImageNet-1k of similar sizes to CIFAR-10 and CIFAR-100, which provide the opportunity to train and evaluate on a dataset more similar to ImageNet without requiring a very long and costly training process.

As previously mentioned, ImageNet16-120 can be a good alternative. The NAS-Bench-201 repository (Dong et al., 2020, 2021) provides ways to obtain the data.

Imagenette (Howard, 2019a) and Imagewoof (Howard, 2019b) are two subsets of ImageNet with 10 classes and multiple resolution options. They are readily accessible on platforms like Huggingface datasets (Lhoest et al., 2021):

- Imagenette features easily distinguishable classes. The scores on Imagenette are therefore higher than on the general ImageNet set.
- Imagewoof selects 10 very similar classes of dog breeds. As a result, the accuracy scores are more conservative than those for Imagenette.

4 Bayesian Optimization

In this chapter, we start with a general introduction to blackbox optimization methods. We then present Bayesian Optimization, a sample-efficient method especially useful with expensive objective functions as is the case in NAS.

Finally, we introduce an example of using Bayesian Optimization principles to efficiently diagnose and identify the failure conditions of an image classifier using a text-to-image model.

4.1 Introduction

4.1.1 Blackbox optimization

Blackbox optimization methods address the problem of optimizing an objective function f with the following characteristics:

- Blackbox: very few assumptions made about f, specifically: no available functional form or closed-form expression, no useful features (convexity, linearity...). f is also potentially multi-modal.
- Derivative-free: no first or second order derivatives. This precludes from applying first or second order optimization methods (gradient descent, Newton or quasi Newton methods).
- Point-wise observation: The only assumption about f is that it can be evaluated at points in its domain. The evaluation can be expensive.
- Noisy evaluation: the observation of the values of f at point x is potentially affected by noise.

As (Talbi, 2009) notes, a standout benefit of metaheuristics is that their assumptions about the problem or model are minimal. This makes them suitable for optimizing objective functions with the aforementioned attributes.

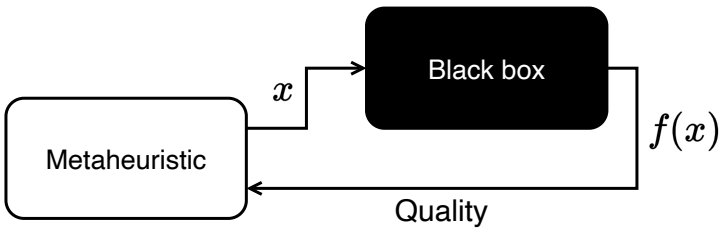


Figure 4.1: Optimization using metaheuristics for a blackbox optimization function. Adapted from (Talbi, 2009)

In particular, blackbox optimization methods are the go-to class of methods when almost no information about the objective function are available. This in turn makes them applicable to a wide range of difficult optimization problems, including NP-hard problems in moderate size spaces, or easy problems (P-hard) in very large search spaces. The only conditions are a known search space and an observation mechanism for evaluating the objective function at points in the search space. The versatility of these methods explains their ubiquity for solving engineering problems which require simulations, for instance optimizing neural network architectures, computational biology or transportation networks.

4.1.2 Expensive objective functions

This is a subclass of blackbox optimization problems with a high cost of evaluating f. In this instance, f is still only possible to evaluate at points, but a key concern becomes making as few evaluations as possible.

There are many instances where f is too costly to evaluate. (Frazier, 2018) defines "expensive to evaluate" as a function whose evaluations must be limited to a few hundreds at most. Exhaustive evaluation is of course out of the question, but the paper cites the different types of costs typically associated with f:

- *Time costs*, if evaluations take hours each
- *Financial costs*, such as cloud compute, energy costs...
- *Human involvement costs*, for instance in human-in-the-loop situations where the evaluation is partly or wholly performed by one or many human subjects, who are usually evaluating aesthetic, artistic or sensory qualities (Herdy et al., 1997; Kim et al., 2017; Zhang et al., 2017; Zhou et al., 2020; Biswas et al., 2024).

This ***expensive objective function*** makes certain types of the blackbox optimization method more pertinent than others.

Bayesian Optimization (BO) is in particular renowned for being very sample-efficient (Shahriari et al., 2015). It fully leverages the history of the previous observations to make educated guesses about the next evaluations to make, and as a result is much more efficient in using as few evaluations as possible.

First devised to aid in optimizing complex engineering systems (Jones et al., 1998), BO has become an important asset used to optimize design choices in a wide range of science and engineering application domains. Whenever a design problem includes multiple choices which interact with each other, and where evaluating one set of choices is expensive, BO can be of help. Examples can be found in environmental modeling and monitoring (Shoemaker et al., 2007; Marchant and Ramos, 2012; Morere et al., 2017), reinforcement learning (Brochu et al., 2010; Lizotte et al., 2007; Balakrishnan et al., 2020; Wilson et al., 2014), robotics (Martinez-Cantin et al., 2007; Berkenkamp et al., 2023; Martinez-Cantin, 2017), sensor networks (Garnett et al., 2010; Srinivas et al., 2009; Sajedi and Liang, 2022), or simulations (Sha et al., 2020; Morita et al., 2022; Park et al., 2018).

4.2 Bayesian Optimization

> Bayesian optimization does not refer to one particular algorithm but rather to a philosophical approach to optimization grounded in Bayesian inference from

> which an extensive family of algorithms have been derived.
>
> (Bayesian Optimization, Garnett (2023))

Bayesian Optimization (BO) lies at the crossroads of these two concepts:

- **Surrogate Model Based Optimization (SMBO):** At the core of a BO method, a probabilistic model is used as a stand-in for the expensive objective function.
- **Sequential Optimization:** The sequential optimization (Garnett, 2023) loop contains three steps:
  - selecting the next evaluation point x following a certain policy
  - evaluating the objective function at the selected point $f(x)$
  - updating the dataset containing the history of previous point-evaluation pairs $D = (x_i, f(x_i))$, which influence the selection policy

Algorithm 1 Sequential optimization (adapted from (Garnett, 2023))

input: $D_0 \leftarrow$ empty set or random point-observation pairs $\{x_i, f(x_i)\}$
repeat
  $x_{n+1} \leftarrow \text{ObservationPolicy}(D_n)$
  $f(x_{n+1}) \leftarrow \text{Evaluate}(x_{n+1})$
  $\text{best} \leftarrow \max(f(x_{n+1}), \text{best})$
  $D_{n+1} \leftarrow D_n \cup \{(x_{n+1}, f(x_{n+1}))\}$
until termination
return best

The combination of these two ideas is how BO operates. Specifically, the probabilistic model contains the prior beliefs about the values of the objective function in the search space. The policy in sequential optimization is determined using the model's estimation of the value and uncertainty at any point of the search space. With the addition of new observations to the dataset D, the model is updated, with the Bayesian posterior representing the updated beliefs.

The definition of the policy, often consisting of maximizing an acquisition function, is what makes BO sample-efficient: each decision to evaluate a point takes into account the history of previous observations, and the decision is made with a sophisticated model which properly models the uncertainty associated with an unobserved point. The exploration-exploitation trade-off is determined by the choice of policy or acquisition function.

4.2.1 Acquisition functions

The policy for selecting the next observation is almost always defined as a maximisation of an intermediate function, called acquisition function or infill criteria (Garnett, 2023).

Algorithm 2 Bayesian Optimization

1: **input:** $D \leftarrow$ empty set or random point-observation pairs $\{x_i, f(x_i)\}$
2: **repeat**
3: $\quad x_{n+1} \leftarrow \arg\max_x \left(\text{Acq.Fn.}(x, \text{model}(D_n))\right)$
4: $\quad f(x_{n+1}) \leftarrow \text{Evaluate}(x_{n+1})$
5: $\quad \text{best} \leftarrow \max(f(x_{n+1}), \text{best})$
6: $\quad D_{n+1} \leftarrow D_n \cup \{(x_{n+1}, f(x_{n+1}))\}$
7: **until** termination
8: **return** best

$$\mathrm{x}_{\text{next}} = \arg\max_{\mathrm{x} \in S} \alpha(\mathrm{x}, D) \tag{4.1}$$

This function depends on the prior observations (contained in D). This dependance is in practice derived from using the probabilistic model's beliefs about the values and uncertainty associated to the objective function at each point of the domain. The acquisition function maps every point in the domain to a score which denotes its usefulness or utility in the search process.

Acquisition functions are much cheaper to evaluate, which makes their optimization significantly simpler than the overall objective function. This aspect, as pointed out in (Garnett, chapter 5), is what makes it a reasonable idea to replace the global difficult optimization with a process which includes its own optimization loop at each iteration.

An important aspect of the acquisition function is that it decides on choices stemming from the **exploration-exploitation dilemma** faced by the search algorithm. Since the scores assigned by the acquisition function are based on the expected value and the uncertainty associated with it, the acquisition function has the task of arbitrating between different situations: capitalizing on a good area of the search space with low uncertainty, or taking more risks and evaluating a potentially higher-value point with higher uncertainty.

There are many acquisition functions used throughout the BO literature and applications. Two of the most notable examples are:

- **Expected Improvement (EI):** how much can the best value found so far $f_{\max}$ be improved on ?

$$\mathbf{EI}(\mathrm{x}) = \mathbb{E}[(f(\mathrm{x}) - f_{\max})^+] \quad \text{s.t.} \quad z^+ = \max(z, 0) \text{ is the positive part}$$

- **Probability of Improvement (PI):** measures the probability of improving on the best value found thus far

$$\mathbf{PI}(\mathrm{x}) = P(f(\mathrm{x}) > f_{\max})$$

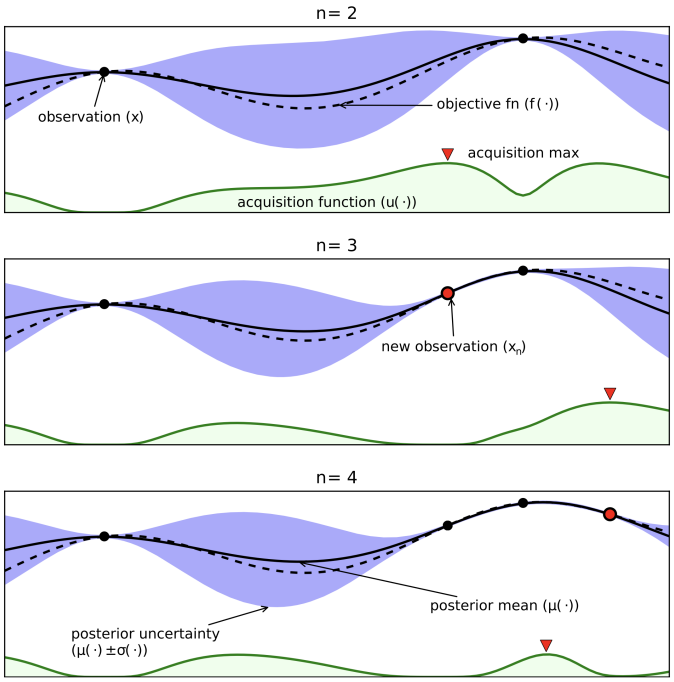


Figure 4.2: Three iterations of the Bayesian Optimization procedure, showing the mean and confidence intervals as estimated by the probabilistic model, and the acquisition function values. Source: (Shahriari et al., 2015)

There are a number of other acquisition functions, including Upper Confidence Bound (Srinivas et al., 2009), knowledge gradient (Frazier et al., 2008; Wu and Frazier, 2016), entropy search (Hennig and Schuler, 2012), and predictive entropy search (Hernández-Lobato et al., 2015).

4.2.2 BO with Gaussian Processes (GPs)

Gaussian Processes (GPs) are to a large extent the canonical probabilistic model for BO, used in the majority of BO instances. Their advantages include simplicity, proven quality in modeling uncertainty, and enabling the expression of relevant quantities, for instance acquisition functions, in a closed form.

Gaussian Processes (GPs) are stochastic processes which extend the multivariate normal distribution (Garnett, 2023). For any finite collection of points $\mathbf{x} = (\mathrm{x}_1, .., \mathrm{x}_n)$, $\mathbf{f} = (f(\mathrm{x}_1), .., f(\mathrm{x}_n)$ is multivariate gaussian:

$$\mathbf{f} \,|\, \mathbf{x} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}) \tag{4.2}$$

$$\mathbf{m} = (\mu_0(\mathrm{x}_1), .., \mu_0(\mathrm{x}_n)) \tag{4.3}$$

$$\mathbf{K} = k_0(\mathrm{x}_{1..n}, \mathrm{x}_{1..n}) \quad \text{i.e.} \quad (\mathbf{K})_{i,j} = k_0(\mathrm{x}_i, \mathrm{x}_j) \tag{4.4}$$

s.t. $\mu_0 : X \mapsto \mathbb{R}$ is the prior mean function and $k_0 : X \times X \mapsto \mathbb{R}$ is the kernel or covariance function (positive semidefinite).

The Gaussian Process is fully described by defining μ_0 and k. The kernel ensures that close points are highly correlated, making their function values more similar than with distant points. GPs have the flexibility to model a wide array of functions, which also explains their suitability in the context of BO.

Let us assume that k noise-free observations have been made $D_k = \{(x_i, f(x_i)) \quad 1 \leq i \leq k\}$. The distribution of value $f(x)$, conditioned on the observations, of a new unobserved point x can be inferred from the mean function and the kernel, where the kernel encodes the proximity of the new point to the previous observations.

$$f(\mathrm{x}) \mid D_n \sim \mathcal{N}(\mu_{k+1}(\mathrm{x}), \sigma^2_{k+1}(\mathrm{x})) \tag{4.5}$$

$$\mu_{k+1}(\mathrm{x}) = \mu_0(\mathrm{x}) + k_0(\mathrm{x}, \mathrm{x}_{1..k})\ k_0(\mathrm{x}_{1..\mathrm{k}}, \mathrm{x}_{1..k})^{-1}\ (\mathbf{f}_{1..k} - \mu_0(\mathrm{x}_1, .., \mathrm{x}_\mathrm{k})) \tag{4.6}$$

$$\sigma^2_{k+1} = k_0(\mathrm{x}, \mathrm{x}) - k_0(\mathrm{x}_{1..k}, \mathrm{x})\ k_0(\mathrm{x}_{1..k}, \mathrm{x}_{1..k})^{-1}\ k_0(\mathrm{x}, \mathrm{x}_{1..k}) \tag{4.7}$$

We present these details to show precisely where the fundamental limitation faced with GPs lies: the dense matrix inversion, whether using the previous expressions directly or via a Cholesky decomposition as a faster and more numerically stable alternative (Frazier, 2018). This operation's complexity increases cubically ($O(k^3)$) with the number of observations k, making it a real bottleneck as the search progresses in the context of BO. As a result, it becomes difficult to handle situations where many evaluations are needed (Snoek et al., 2015) (e.g. very large search spaces and/or very high dimensional problems), as well as complicating parallelization.

4.2.3 BO with alternative models

While GPs have very good reasons to have a privileged position in BO as the most prevalent regression model, the cubic scalability issue arises in certain scenarios. As a result, approaches based on sparse GP regression have been proposed to alleviate this inefficiency (Seeger et al., 2003; Snelson and Ghahramani, 2005; Lázaro-Gredilla et al., 2010).

Alternatively, other models have been successfully used in BO, which alleviate the cubic complexity but offer their own set of tradeoffs. Examples include:

Random Forests (RFs):

Random forests for regression average the prediction of an ensemble of decision trees trained on different subsets of the training data.

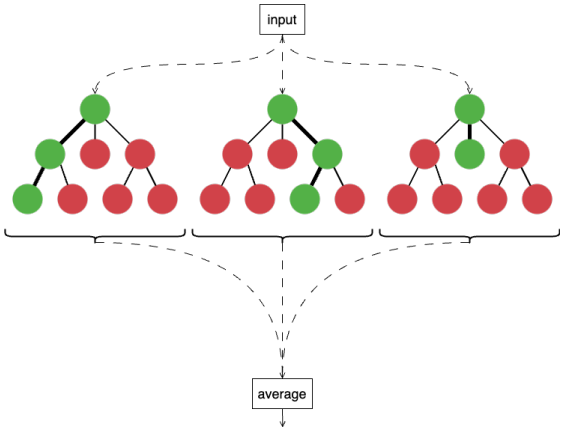


Figure 4.3: Schematic representation of a random forest

For Bayesian Optimization, they offer the following benefits:

- Since they are based on decision trees, incorporating categorical or conditional input data is possible.
- Straightforward parallelization
- Scalability: support for high dimensions and for a big number of datapoints as compared to GPs

The non-differentiability of the output precludes the use of gradient-based or quasi-Newton methods. Following SMAC (Hutter et al., 2011), blackbox search methods (especially random search and local search) can be used to optimize the acquisition function. Additionally, (Hutter et al., 2011) uses an empirical estimate of the uncertainty via the variance of the outputs of the different decision trees.

Neural networks (NNs) and other deep models:

In (Snoek et al., 2015), the authors use a neural network as a model for BO. The main objective is to improve scalability, by reducing the cubic complexity of GPs to a linear complexity. Special attention was paid to retaining the advantages of GPs especially in the estimation of uncertainty.

Instead of using a full Bayesian Neural Network (BNN) (Jospin et al., 2022), which are theoretically linked to GPs, the authors propose a different approach to avoid the high computational cost associated with BNNs. The DNGO method developed in (Snoek et al., 2015) starts with a normal (deterministic) NN trained using the usual gradient-descent based empirical loss minimization. The final layer made a bayesian linear regressor, which turns the model into a suitable alternative to GPs with good properties for uncertainty estimation. The method is then applied for multiple optimization problems, including finding high quality CNNs on image classification datasets, and image description using deep language models.

Other deep models are cited in (Garnett, 2023), including Deep Gaussian Processes (Dai et al., 2015) and probabilistic transformers (Maraval et al., 2022).

4.3 BO application example: efficient exploration of image classifier failures

4.3.1 Introduction

It is widely acknowledged that vision models have made significant breakthroughs and achieved impressive results in recent years. While they can achieve excellent average classification accuracy, particularly when tested on data with a high degree of similarity to the training data, one of the issues is a potentially significant drop in performance in certain conditions. This can be the case in conditions not encountered frequently in the training data, when there is a shift in the data distribution, with the presence of biases, or with out-of-distribution (OOD) inputs.

To be able to use them in critical systems, such as in autonomous driving, aviation, disaster response or medical applications, trustworthy predictions are key. Therefore, it is important to have practical and effective ways to diagnose and identify failure points in their models. As a result, quantitative and qualitative assessments of vision models' points of failure is a very important prerequisite for safety, adoption and trust.

Multi-modal generative models have recently made very big strides in the quality of their outputs. Text-to-image models in particular offer the capability to generate high quality images from textual prompts. This opens up the possibility of methodically evaluating the performance of vision classifiers on synthetic images obtained by chaining together a number of descriptive text tokens, thereby analysing how often and under which specific conditions failures occur (Metzen et al., 2023; Vendrow et al., 2023; Wiles et al., 2022)

Diffusion models (Ho et al., 2020) are a leading class of generative models, which have outperformed GANs and VAEs in generating high quality and diverse images. They are trained by adding noise to data and learning de denoise it, acquiring the ability to generate high-quality samples in the process. Conditioning the reverse step on text embeddings allows finer-grained, text-based control of the outputs. They can have an inference time, owing to the multiple denoising steps.

Using diffusion models to identify the conditions under which a vision classifier underperforms is computationally expensive (Wiles et al., 2022). Additionally, the space of possible combinations for composing the text prompts, as with other spaces representing multi-dimensional design choices, can be combinatorially large.

While (Metzen et al., 2023) uses combinatorial testing, these circumstances favour using a

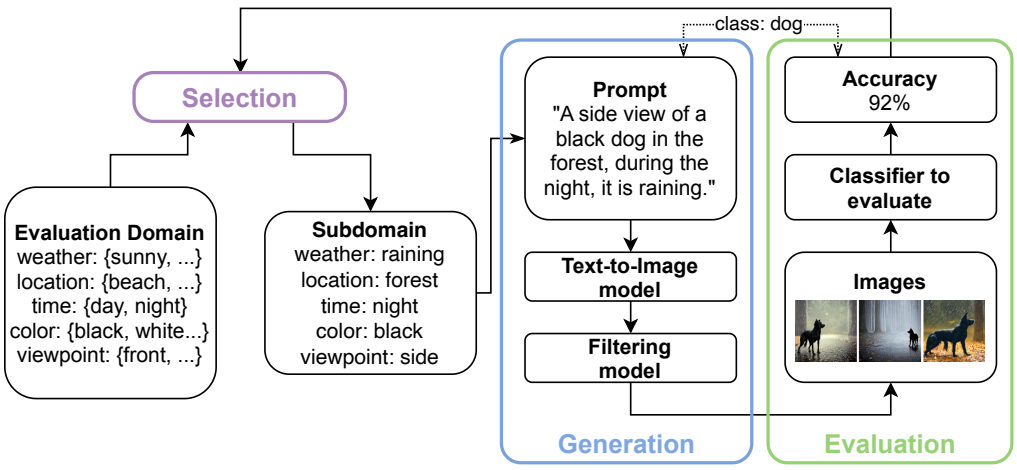


Figure 4.4: The exploration method, which involves **selecting** the next subdomain to evaluate leveraging evaluations of the previously selected subdomains, **generating** the synthetic images, and **evaluating** the classifier's performance on the selected subdomain.

blackbox optimization algorithm. With expensive evaluations, BO and similar sample-efficient methods are a good pick to evaluate vision models all while keeping time and compute costs reasonable. The goal is to find efficient ways to explore the data attributes and conditions which most heavily degrade classification performance.

4.3.2 Method

The text prompts are generated using a template: *"a (viewpoint) view of a (color) dog (location), during (time), it is (weather)"*. The categorical attributes inserted in the template form a space of 1032 combinations, the result of combining multiple types of weather conditions, locations, time settings, colours and perspectives, and after the elimination of unfeasible combinations.

The text-to-image generative model used is an implementation of Stable Diffusion 2.1 by Stability AI, based on the architecture of Latent Diffusion Models (Rombach et al., 2022). A filtering model, a pretrained CLIP ViT-L/14, is used to discard images inconsistent with the text prompt generated from a particular combination.

Figure 4.5 showcases a few examples of prompt and image pairs. The images in the first row are classified correctly by the image classifier, while those in the second row are examples where the classification fails.

The classifier used is a vision transformer, specifically ViT-B/16 (Dosovitskiy et al., 2020), with pretrained weights on ImageNet. It is repurposed as a binary classifier to classify dog images by summing the probabilities for dog breeds (119 classes out of ImageNet-1k's 1000 classes).

(a) "A side view of a brown dog in the city, during the day, it is sunny."

(b) "A front view of a white dog at the beach, during the day, it is sunny."

(c) "A front view of a red dog in the city, during the day, it is snowing."

(d) "A front view of a white dog at the beach, during the day, it is sunny."

(e) "A rear view of a blue dog in the mountains, during the night, it is foggy."

(f) "A front view of a beige dog inside a house, during the day, it is sunny."

Figure 4.5: Examples of the synthetic images with the prompts used to generate them

Search method:

Using the same intuitions behind BO, a predictive model is used to guide the search towards the critical subdomains where failures lie.

1. The selection policy used to pick the next subdomain (i.e. combination of descriptors) to evaluate is a simple version of Expected Improvement, the most widely used acquisition function in BO approaches. Leveraging the model's estimation of each subdomain's quality, the subdomain with the greatest estimated improvement over the current best subdomain is selected.

2. Afterwards, the selected subdomain is evaluated. This evaluation loop includes:

   (a) composing of the text prompt

   (b) generating the images using the generative model

   (c) resizing them to a 256x256 resolution

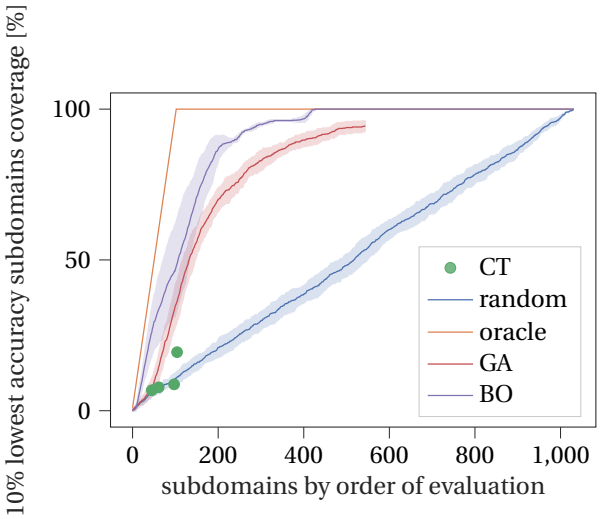


Figure 4.6: Comparing how fast the 10% lowest accuracy subdomains are covered by different search strategies as the search progresses

(d) filtering them to ensure alignment with the original prompt

(e) evaluating the binary classifier on them

3. With the newly acquired observation data consisting of a new pair of (subdomain, classifier performance), the model is updated by adding this new point to its training dataset, in preparation for future iterations. Future selections are therefore made in light of the information about subdomain failures acquired during this loop and past loops.

4.3.3 Experiments and results

In order to evaluate different search algorithms, the classifier's performance is evaluated on all 1032 subdomains and the results are stored in a lookup table, similarly to NAS benchmarks. With the costs of generating enough images to obtain 50 valid ones, evaluating one subdomain takes on average 12 minutes on an NVIDIA V100 GPU, putting the total cost of evaluating the entire space at over 200 GPU hours. While this is done for benchmarking purposes in this experiment, it further illustrates the need for efficient exploration methods.

In the following results, Oracle denotes the hypothetical optimal policy where the accuracies are known beforehand, to illustrate the highest performance theoretically possible in this experiment.

The goal is to measure how quickly each search method identifies the subdomains where performance degrades the most. Using the 10% subdomains with the lowest accuracy, figure 4.6 illustrates how the coverage of these subdomains evolves as the search progresses, reaching

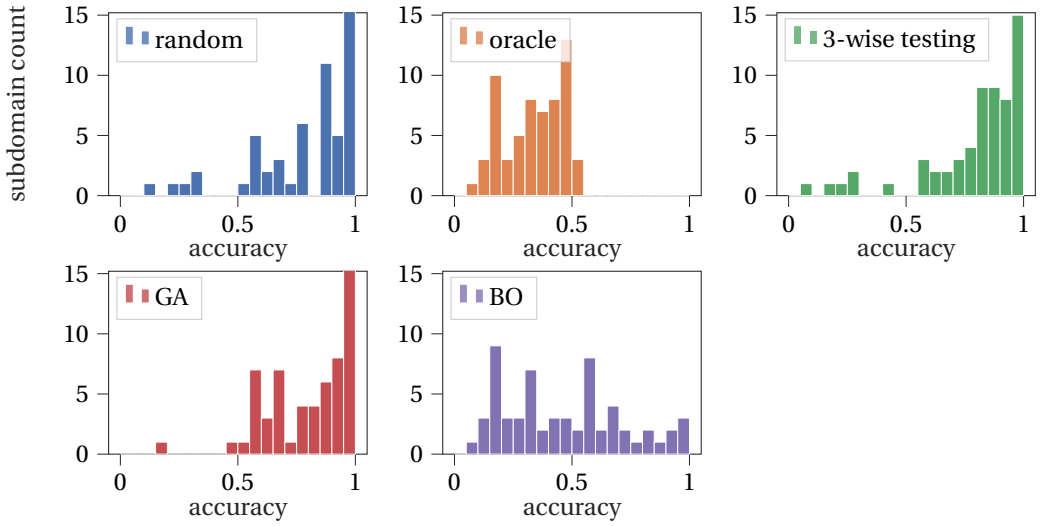


Figure 4.7: Distribution of the evaluated subdomains at a fixed budget of 61 evaluations. BO identifies the biggest proportion of low-accuracy subdomains.

100% when they are all found. The described search method identifies all the subdomains after ≈ 300 evaluations, outpacing a genetic algorithm-based solution, as well as random search and combinatorial testing.

Comparing the results of the set of evaluated subdomains with a fixed budget of 61 evaluations, the genetic algorithm (GA) implementation and BO demonstrated better capabilities in identifying subdomains with lower classification accuracy, and therefore are better at detecting the binary classifier's failure points.

Part II

Pretrained deep ensembles and multi-fidelity BO for NAS

5 Fast NAS using pretrained deep-ensembles and multi-fidelity BO

In this chapter, we present an efficient NAS method based on Bayesian Optimization and deep ensembles.

We present the motivations behind choosing deep ensembles as the BO model, chief among which the potential to significantly accelerate the search procedure.

We illustrate this search acceleration using two complementary ideas: simultaneous pretraining and multi-fidelity search.

Using multiple experiments on a NAS benchmark, we analyze the impact of each of these ideas and compare the search efficiency to other methods in the literature.

5.1 Introduction

5.1.1 Motivation

As discussed in chapter 4, Bayesian Optimization (BO) is a sample-efficient method to perform blackbox optimization, with Gaussian Processes (GPs) underpinning most BO approaches.

GPs have many advantages as well as a few limitations, chiefly the scalability bottleneck. Nevertheless, is possible to effectively use alternative probabilistic models to guide the BO search. This is also the case in BO applied to NAS, where graph convolutional networks and Bayesian graph NNs have been used with good results (Ma et al., 2019; Shi et al., 2020).

Many blackbox optimization problems with expensive objective functions are often solved using BO. However, the NAS problem has some attributes which set it apart:

GPs and NAS search spaces:

Applying GPs to NN architecture spaces, which can be complex with mixed variables and conditionals, is non-trivial. In fact, there is a need to define a distance function to use in conjunction with a kernel, and to select an appropriate kernel.

For search spaces consisting of neural network architectures, the question of quantifying the distance between two architectures is a design choice with no immediately obvious answer, and one with substantial implications on the effectiveness of the search.

This also potentially makes the applicability of the BO method dependent on the search space, where certain distance functions or kernels might not support certain operations. Finally, this aspect introduces an additional need for domain knowledge in order to define a meaningful distance function.

A number of kernels and distance measures have been suggested, including optimal transport metrics for architectures of neural networks (OTMANN) (Kandasamy et al., 2018), Weisfeiler-Lehman kernels (Ru et al., 2020), edit-distance neural network kernels (Jin et al., 2019), and phenotypic distance (Hagg et al., 2019).

Availability of extra information:

Additional information about architectures in the search space can be obtained with very little cost, as certain metrics can be reliably measured without the need for training.

This provides an opportunity to potentially improve efficiency and reduce the search time, which is a key concern in NAS and a prime reason for choosing sample-efficient search strategies like BO.

5.1.2 Contributions summary

In this chapter, we present a fast NAS method which combines the inherent sample-efficiency of BO with the previously mentioned elements which are specific to the NAS context. Our NAS method is built on the idea of deep-ensembles as a predictor, leveraging its inherent properties to accelerate the search time and minimize the number of evaluations needed to achieve good results on benchmarks as well as on custom search spaces.

A first important advantage of this design choice is eliminating the need to explicitly define a meaningful distance function between architectures in the search space. This has immediate implications, as the application of the method becomes much easier on new or particularly complex search spaces.

The second key advantage is the possibility of leveraging a maximum of the available inexpensive information to accelerate the search. We achieve this using **weight sharing** at the predictor level: since DEs are sets of neural networks, we can design architectures where unified embeddings are used for multiple prediction targets. As a result, during training, multiple sources of information can be leveraged concurrently to improve the unified embeddings' quality. Instead of only relying on the expensive evaluations of the objective function, fewer evaluations are combined with useful information from cheaper-to-evaluate sources and encoded in the internal representations of the ensemble. We illustrate this with two complementary ideas: **simultaneous pretraining** on zero-cost metrics, and **multi-fidelity search**.

Pretraining is of course a widely-used technique in machine learning. It gives the model a head-start, by allowing it to learn general patterns in the data before the training starts, boosting its predictive performance when it comes to the real prediction target. By using BO, a key objective is to minimize the number of evaluations, which in turn means the model gets fewer datapoints. As a result, pretraining becomes especially crucial, as we want the model to perform well as early in the search process as possible.

While not very common for GPs in particular, methods such as HyperBO (Wang et al., 2024) use pretrained GPs for BO-based hyperparameter optimization. By contrast, pretraining a deep neural network, and by extension a deep ensemble, is more straightforward, commonly performed using conventional tools, and readily supports parallelization.

Going beyond simple pretraining schemes, our main insight in the **simultaneous pretraining** section is how to best combine multiple pretraining targets. Instead of pretraining the ensemble on one metric at a time, we use a shared embedding to predict all the metrics at once. This ensures the internal representation is more generic, and results in a significant speedup of the BO search using the pretrained ensemble, with an improved predictive performance especially during the first iterations of the BO loop.

Additionally, this notion of shared embeddings also allows us to share information between

different fidelity levels, allowing us to trade a few high quality evaluations for many lower quality evaluations. The unified internal representation is improved during the search by evaluations using information from both fidelity levels, allowing an even more robust performance. This **multi-fidelity search** approach also results in clear speedups, allowing the search strategy to reach the top performing architectures without incurring a big number of costly evaluations.

Combined, these additions significantly accelerate the search time, finding the optimum on the NAS-Bench-201 benchmarks in an equivalent time and cost to performing as few as 50 to 80 evaluations.

5.2 Method description

5.2.1 Deep ensembles and NAS

Ensembling methods (Ganaie et al., 2022) involve aggregating the predictions of a set of diverse models. Using an aggregate prediction is more robust and reliable than using an individual prediction model. The models can compensate mutual weaknesses and offer complementary strengths. A deep ensemble (DE) is the ensemble of a number of independently-trained neural networks.

DEs can reliably quantify uncertainty in many scenarios (Lakshminarayanan et al., 2017), and are effective in out-of-distribution cases. With a large number of parameters in neural networks, there are many low-loss regions in which a network can approximate the training data. Deep ensembles can produce a set of diverse networks representing different low-loss regions. In practice, compared to other models such as Bayesian NNs, DEs are also easier to implement and to parallelize.

The Trieste BO-based search framework (Secondmind Labs, 2023) presents an example of using a deep ensemble as a probabilistic model for Bayesian optimization. In the NAS context specifically, the most prominent example is BANANAS (White et al., 2021a), where the authors thoroughly analyzed many components of BO applied to architecture search: predictor choice, acquisition functions, etc.. This led to the design of a successful NAS method based on an ensemble of networks, coupled with a novel architecture encoding scheme: path encoding. This work clearly demonstrated the potential of deep ensemble-based NAS.

Compared to GPs, the previously mentioned methods eliminate the need for explicit kernel and distance functions in the space of neural network architectures. Flexibility in the choice of architecture for the ensemble networks is another advantage. For example, a GNN-based predictor can be used for graph-based search spaces. The ensembling allows an ordinarily non-probabilistic model, such as a feedforward network or a GNN, to be used as an effective probabilistic predictor for BO.

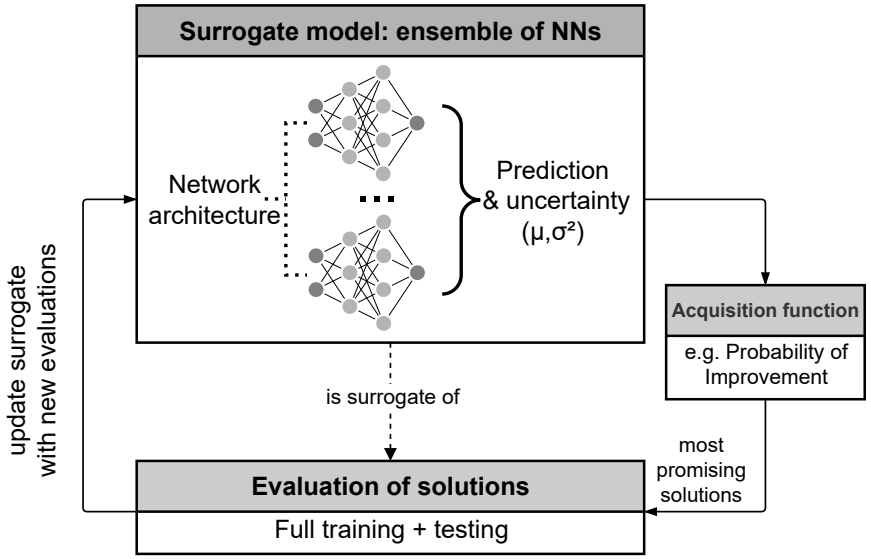


Figure 5.1: Overview of Bayesian optimization with deep ensembles

5.2.2 Bayesian optimization with DEs

At a glance, our Bayesian optimization (BO) procedure with deep ensembles is similar to BO with Gaussian processes (Figure 5.1). At each iteration, the probabilistic model is used to select the best points to be evaluated, and the evaluations are in turn used to update the model and improve it in preparation for the next iterations.

Our surrogate model is a deep ensemble, composed of a number of neural networks which take an architecture (described by an encoding) as input, and predict its performance (the objective function). Owing to the different initializations, the networks in the ensemble end up occupying different low-loss regions in the loss landscape after training. As a result, their predictions for a particular candidate architecture are different, and combining them yields a more robust prediction (average) and a reliable indication about the uncertainty (variance).

The acquisition function is the criteria used to select the most interesting points to evaluate. It relies on the predictions and uncertainty measures provided by the probabilistic model, and different acquisition functions strike various balances between exploration and exploitation. We used the probability of improvement (PI) as acquisition function. It selects the points with the highest probability of improving over the current best observed value. Although in GP-based BO other acquisition functions (e.g. Expected Improvement) usually outperform PI, our tests showed PI to result in slightly faster search times. We use the predictions computed by the ensemble networks as MC samples to approximate the acquisition function PI.

At each iteration, the newly acquired observations are added to the training set of the deep ensemble, and the networks of the DE are trained to incorporate the new information.

5.2.3 Simultaneous pretraining

For Neural Architecture Search, FBNetV3 (Dai et al., 2021a) is a notable example of successfully using a pretrained predictor in the search process. In a first step, architecture statistics are used to pretrain a predictor, which is then used as a proxy for the objective function in a predictor-based evolutionary search. The pretraining procedure led to an important boost in the sample efficiency of the predictor.

In the context of Bayesian optimization, the use of deep ensembles significantly simplifies this pretraining step. As opposed to GPs and other probabilistic models, pretraining a deep ensemble is simple: it trivially consists of pretraining its component networks. The impact of pretraining in general, and the way to pretrain the ensemble, is well established. Instead, our main insight is related to the pretraining data we use, and especially to the manner in which the pretraining on many metrics is performed.

For neural networks in the search space, we can quickly calculate a number of metrics to use as pretraining data. We focus on zero-cost metrics, i.e. metrics whose computation or estimation doesn't require prior training of the neural network in question. These metrics depend only on the architecture, or hardware, but not on whether it is trained or not. Among such zero-cost metrics, we use the number of trainable parameters, the average inference latency, the number of floating point operations (FLOPs) and the memory footprint.

With access to multiple pretraining metrics, the main question we focused on is **which pretraining method is the best way to fully leverage these multiple metrics**.

Our idea is to force the common representation generated by the networks to remain as generic as possible, while retaining relevant information. Since these metrics are not our real prediction target, simultaneous pretraining is used to keep the embedding from specializing on any one metric to the detriment of our real prediction target. Practically, this is achieved by using a common embedding to predict all the metrics simultaneously.

In practice, each ensemble network is composed of a section made of n layers $(L_1, .., L_n)$, followed by prediction heads for the different target metrics. This first section is mutualized to all the pretraining metrics. In the forward step, this shared section produces a common embedding $\mathbf{y}$ of the input data $\mathbf{x}$.

$$\mathbf{y} = L_n(L_{n-1}(....(L_1(\mathbf{x})))) \tag{5.1}$$

The embedding $\mathbf{y}$ is subsequently used to predict each of the m pretraining metrics, using a separate prediction head f_i for each metric.

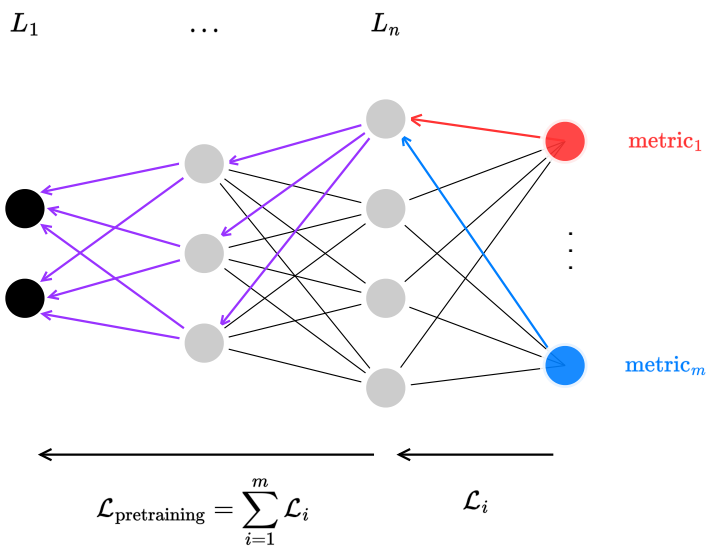


Figure 5.2: Backpropagation step of the simultaneous pretraining scheme, with a separate prediction head for each zero-cost metric

$$\begin{aligned} \text{metric}_1 &= f_1(\mathbf{y}) \\ \text{metric}_2 &= f_2(\mathbf{y}) \\ &\vdots \\ \text{metric}_m &= f_m(\mathbf{y}) \end{aligned} \tag{5.2}$$

In other words, all the metrics are predicted using a common vector $\mathbf{y}$ representing the input architecture. The loss is the sum of regression losses of the pretraining metrics, as illustrated in Figure 5.2. As a result of this setup, the backpropagation step updates the weights of the shared section using combined loss information from all the zero-cost metrics.

In our experiments (subsection 5.3.1), we measure the impact of correlations between the metrics, and the impact of the number of metrics used. We also compare this pretraining scheme to no pretraining and to sequential pretraining.

5.2.4 Multi-fidelity search

The motivation for multi-fidelity search is to replace a low number of high quality evaluations, with a high number of lower quality evaluations (Figure 5.3). For example, we can use the validation score after fully training the model (N epochs) as the high-fidelity evaluation. The low-fidelity evaluation is then the validation score after a smaller number n of training epochs ($n < N$).

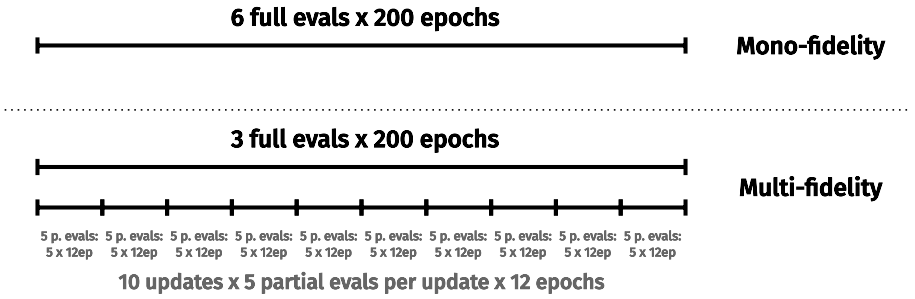


Figure 5.3: Mono-fidelity and multi-fidelity training at equivalent costs

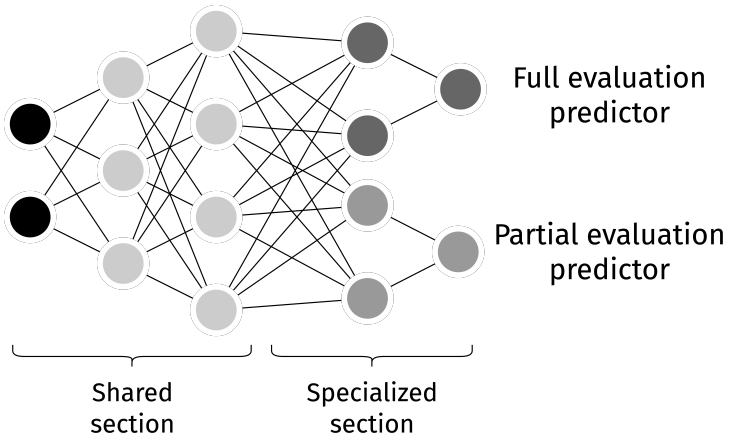


Figure 5.4: Ensemble network architecture for multi-fidelity search

The weight sharing is implemented in a similar way to simultaneous pretraining: a common embedding is used to estimate both fidelity levels of the evaluation. As a result, in a similar way to the multi-head pretraining method, our ensemble networks have a shared section, followed by a specialized section (a dense layer and prediction head) for each level of fidelity. Figure 5.4 illustrates this architecture.

The training procedure is as follows:

- At each iteration, perform p low-fidelity evaluations.
- Every f iterations, perform q high-fidelity evaluations.

Both of these steps update the shared sections of the ensemble networks, which results in this section being updated more frequently than in the mono-fidelity case. The high-fidelity evaluations are given more importance than the low-fidelity evaluations, by training the ensemble on their data for more epochs.

In subsequent experiments, we compare this training procedure to the mono-fidelity approach, and illustrate whether its impact is additive or contradictory with the impact of

simultaneous pretraining.

5.3 Experiments and results

In this section, we use a feedforward network architecture for the ensemble networks, and perform multiple tests using the NAS-Bench-201 benchmark (Dong and Yang, 2020; Dong et al., 2021). Appendix A contains additional details about the implementation, including the ensemble architecture, hyperparameters and parallelization.

More predictive performance could be expected if we adapt the ensemble architectures to the graph-based structure of the benchmark (e.g. using GNNs), but we elected to demonstrate the applicability of the method to a larger spectrum of possible search spaces. In chapters 7 and 8, the same ensemble configuration is used on cell-based search spaces, as well as more complicated spaces with conditional variables.

Spearman rank correlation

We use the Spearman rank correlation in many of the following experiments to quantify how well the model ranks the architectures in the benchmark: the correlation between the true ranking, and the ranking based on the model's prediction.

The Spearman rank correlation is defined as the Pearson correlation coefficient calculated between the rankings of the data series. For n pairs of datapoints $(x^i)_{1\leq i\leq n}$ and $(y^i)_{1\leq i\leq n}$, the correlation is calculated on the rankings $(r_x^i)_{1\leq i\leq n}$ and $(r_y^i)_{1\leq i\leq n}$:

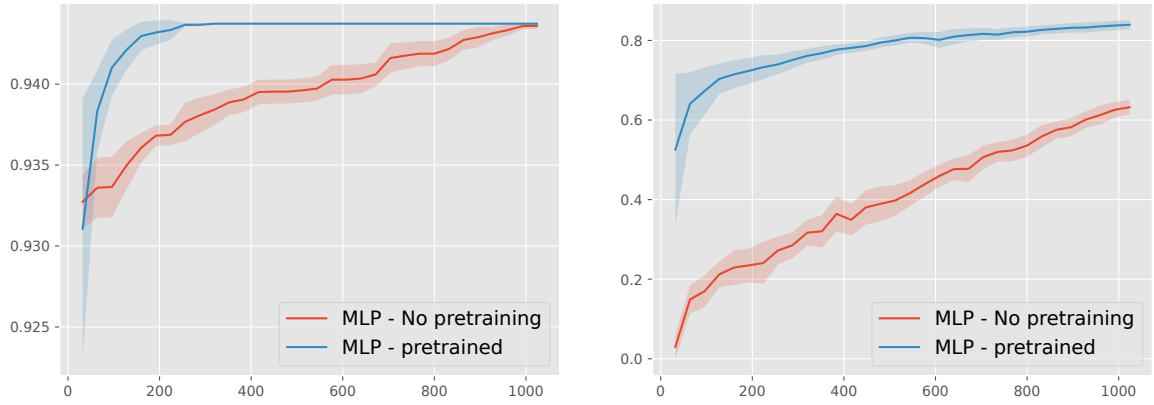
$$\rho(x,y) = \frac{\operatorname{cov}(r_x,\ r_y)}{\sigma_{r_x}\,\sigma_{r_y}} \tag{5.3}$$

5.3.1 Simultaneous pretraining experiments

In figure 5.5, we compare the evolution of the best value and the Spearman rank correlation with and without the simultaneous pretraining, as we advance along the search procedure. The x-axis tracks the number of evaluations performed.

In figure 5.6, we report the average search time, i.e. the number of evaluations needed to reach the optimum, for CIFAR10 and ImageNet16-120. We also report the Spearman rank correlation values ρ after 512 evaluations.

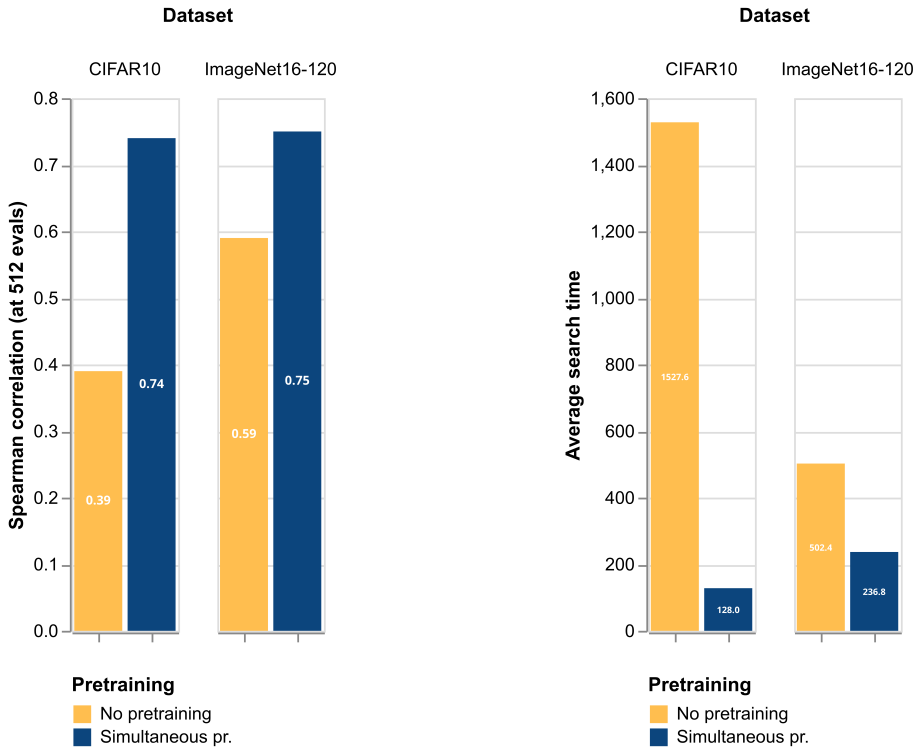
The results show that pretraining has a significant impact on the predictive performance of the ensemble: the increase in rank correlations means the model suggests better points to be evaluated, and as a result the best value plot shows a significant advantage for the pretrained



Best accuracy found

Spearman rank correlation ρ

Figure 5.5: Best accuracy and rank correlation evolution during the search (CIFAR10 - 20 run averages)



Spearman rank correlation ρ

Average search time

Figure 5.6: Impact of the pretraining step on rank correlations and average search time after 512 evaluations (CIFAR10 and ImageNet16-120)

DE.

Impact of the pretraining method

To specifically evaluate the impact of **simultaneous pretraining**, we compare it to **sequential pretraining**, where pretraining is performed on metric_1, then metric_2, etc..

We varied the number of pretraining epochs to account for the different hyperparameters each pretraining mode might require. We reported the average search time in figure 5.7.

The results show a clear trend where the search is significantly accelerated by simultaneous pretraining. In our experiments, the same trend does not hold for the rank correlations, which remain largely the same for both modes of pretraining. This indicates that models with similar rank correlations can still have different search times.

Pairwise correlations and number of pretraining metrics

During the pretraining step, we used three zero-cost metrics: number of trainable parameters, FLOPs, average inference time. The NAS-Bench-201 (Dong and Yang, 2020; Dong et al., 2021) benchmark allows us to calculate some pairwise correlations between the different pretraining metrics, as well as between evaluations at high and low fidelities (Table 5.1).

We also measured the impact of the number of the metrics used on the search time (Table 5.2). The search time corresponds here to the average number of evaluations needed to find the optimal architecture in the benchmark.

While testing with 2 metrics, we found that the differences in pairwise correlations have a small impact on the search time. The number of metrics has a much more significant impact. Therefore, using more metrics, even if that includes highly correlated metrics (e.g. number of parameters and FLOPs), yields the fastest search times.

5.3.2 Multi-fidelity search experiments

NAS-Bench-201 includes validation scores after 12 epochs and after 200 epochs of training, which we use respectively as the low-fidelity and the high-fidelity evaluations.

We use values of $p = 5$, $q = 3$, $f = 10$, ie. 3 full evaluations are performed for every 50 partial evaluations.

In figures 5.8 and 5.9, we plot the average best values found and Spearman rank correlation with and without multi-fidelity training. For CIFAR10, we also include the values without any pretraining or multi-fidelity training, to illustrate the combined impact of these two procedures. On the x-axis, we track the current cost in terms of "equivalent full evaluations",

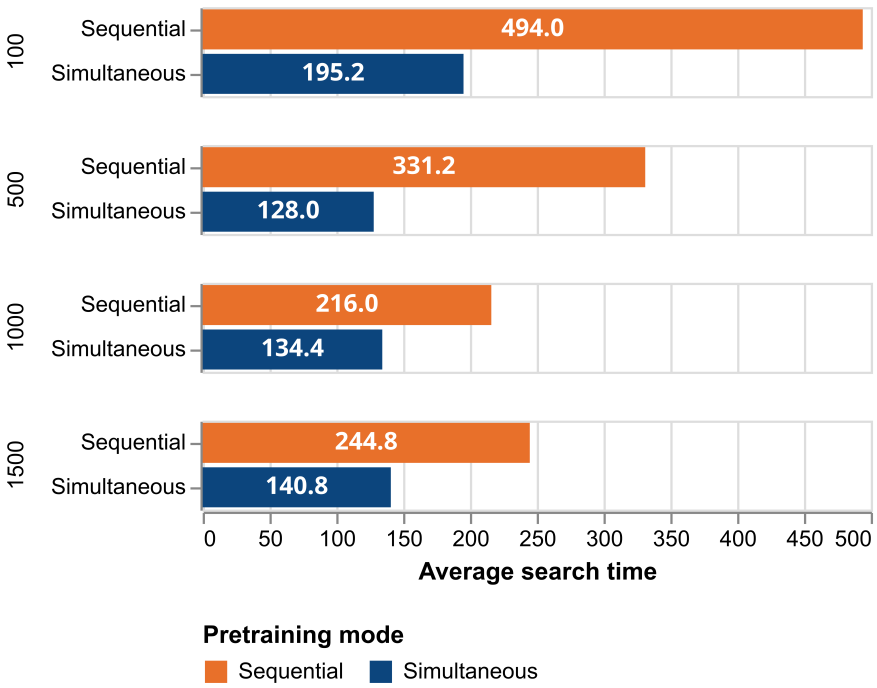


Figure 5.7: Average search times using sequential and simultaneous pretraining. The rows indicate the number of pretraining epochs.

Table 5.1: Pairwise correlations of pretraining metrics

	N.params	Latency	Hi-Fi acc.	Lo-Fi acc.
N.params	1	0.68	0.32	0.38
Latency	0.68	1	0.38	0.48

Table 5.2: Effect of the number of metrics on search time

N.metrics	1	2	3
Search time (in evaluations)	113.4	96	78.9

to be able to compare results with the mono-fidelity approach. We calculate this as the total number of epochs at that point, divided by the number of epochs for a full evaluation.

Our experiments show that multi-fidelity search with shared weights improves the search speed with and without pretraining, and that combining both produces much faster search speeds. The next section provides an overview of the method, and its main results on NAS-Bench-201.

5.3.3 NAS Method overview and results

Figure 5.10 sums up the NAS approach in its two steps: the simultaneous pretraining, and the multi-fidelity BO loop.

We present our method in pseudo-code form in Algorithm 5.12.

We provide an overview of the impact on search time of the two main ideas introduced in this paper in figure 5.11: simultaneous pretraining and multi-fidelity search.

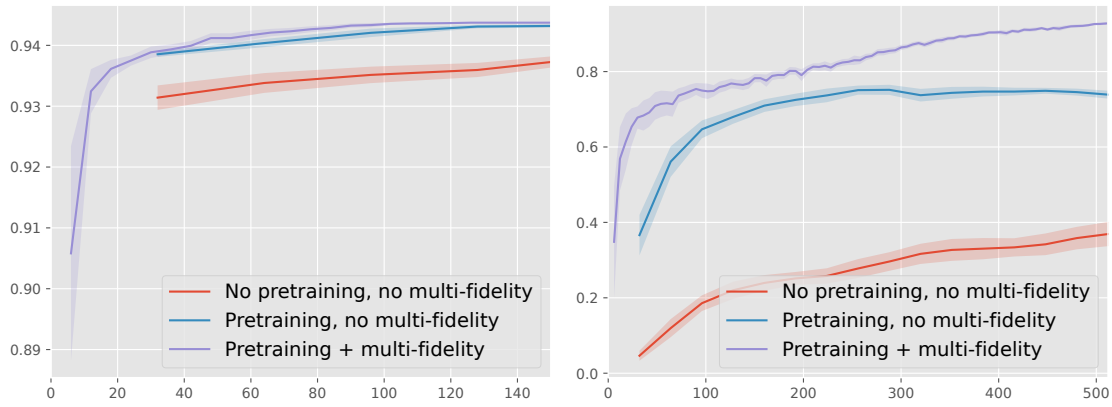
In table 5.3, we report statistics about the number of evaluations needed to reach the optimum on NAS-Bench-201 datasets. We compare to other NAS methods in the literature where it was possible to access this data, most notably using the NASZilla library (White et al., 2020), which includes BANANAS (White et al., 2021a) and the local search approach (White et al., 2021b).

In table 5.4, we report the best accuracy value found by the search method, after respectively 100 and 200 evaluations (or queries) on NAS-Bench-201. We compare this with other NAS methods in the literature, either using our own tests or reported results from their respective papers.

On the 100 evaluation budget, our method finds architectures with the top value in all 3 benchmark datasets, finding the optimum for C100. By the 200 evaluations mark, our method find all 3 optima for the NAS-Bench-201 dataset.

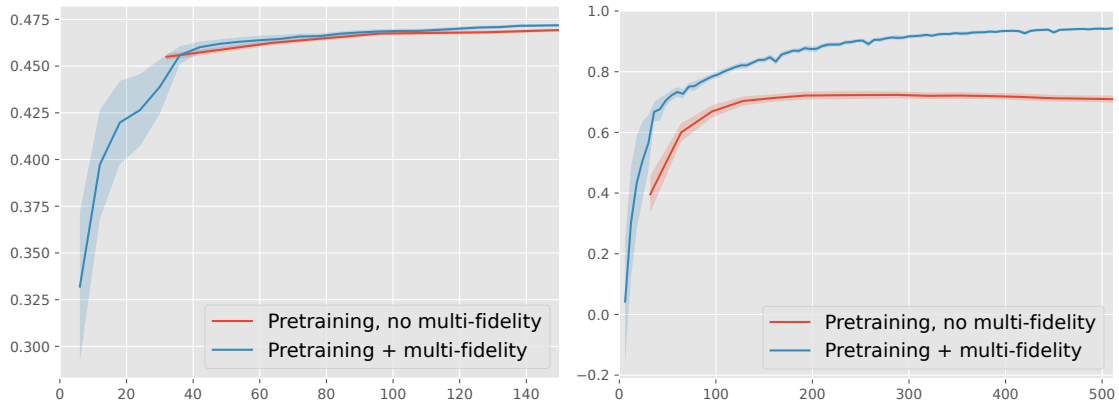
5.4 Conclusion

Deep ensembles are an effective probabilistic model for Bayesian optimization applied to NAS. They present a number of upsides, especially the flexibility to leverage additional sources of information to improve the predictive performance. We showcase this in practice with simultaneous pretraining on zero-cost metrics and multi-fidelity search. These additions lead to significant speedups of the search procedure, respectively accelerating the search 2.5 and 1.6 times on NAS-Bench-201 (CIFAR10).



Best value Spearman ρ

Figure 5.8: Impact of multi-fidelity training coupled with pretraining (CIFAR10, 20-run average)



Best value Spearman ρ

Figure 5.9: Impact of multi-fidelity training coupled with pretraining (ImageNet120-16, 20-run average)

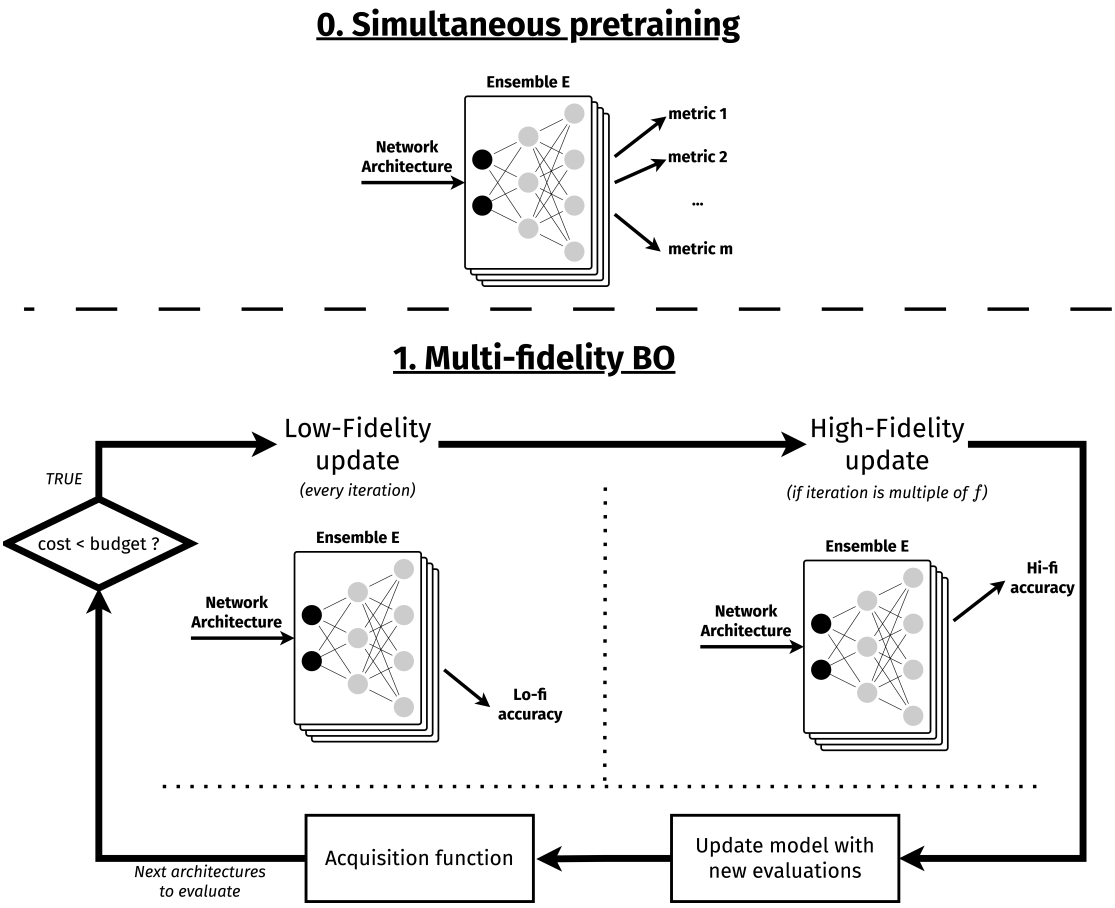


Figure 5.10: General overview of the NAS method. Ensemble E's weights are pretrained in **step 0**, then updated both by high fidelity and low fidelity evaluation data. It is used at each iteration to suggest the next points to evaluate.

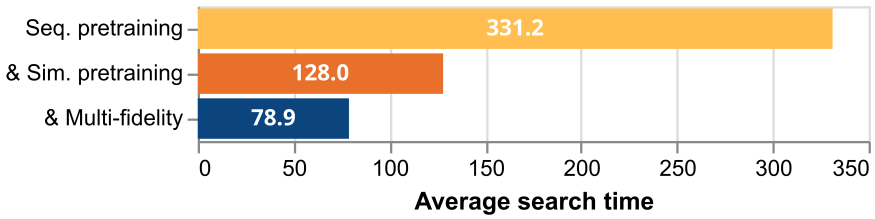


Figure 5.11: Combined effect of simultaneous pretraining and multi-fidelity search on the search time (CIFAR10, 20 run average)

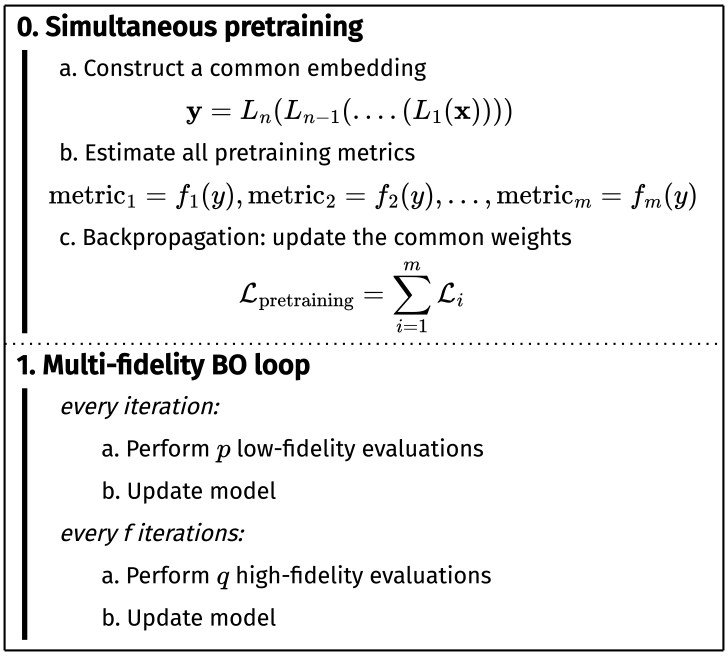


Figure 5.12: Overview of the search procedure

Table 5.3: Statistics about the number of evaluations to reach optimum (20-run averages)

		Min	Max	Mean	Std
CIFAR10	Random search	280	13690	7717.0	4498.8
	Evolution	20	1000	250.0	247.41
	BANANAS	20	470	128.5	118.59
	Local search	30	190	99.0	56.82
	Ours	36	132	**78.9**	25.45
CIFAR100	Random search	520	14690	7141.5	4137.08
	Evolution	40	650	247.5	154.2
	BANANAS	40	310	101.0	54.49
	Local search	20	490	118.5	96.4
	Ours	18	96	**53.7**	24.03

Table 5.4: Best value after 100 and 200 epochs (20-run averages)

NAS Method	CIFAR10	CIFAR100	ImageNet16-120	Queries
Optimum*	**94.37**	**73.51**	**47.31**	
Arch2vec & BO (Yan et al., 2020)	94.18	73.37	46.27	100
AG-Net (Lukasik et al., 2022)	94.24	73.12	46.20	100
Random search	93.70	70.94	45.44	100
Local search	94.26	73.15	46.09	100
Evolution	94.20	72.70	46.03	100
BANANAS	94.15	73.28	46.46	100
Ours	**94.36**	**73.51***	**47.19**	100
Random search	93.86	71.78	45.70	200
Local search	**94.37***	73.48	46.21	200
Evolution	94.24	73.07	46.30	200
BANANAS	94.20	73.43	46.51	200
AG-Net	**94.37***	**73.51***	46.43	192
Ours	**94.37***	**73.51***	**47.31***	200

NB: There are of course better-performing networks on these datasets in the literature in general, but here we are interested in how fast the different methods perform the search, and are restricted to the architectures contained in the benchmark. As a result, the accuracies have an upper bound (first line).

6 Neural Architecture Tuning: A BO-Powered NAS Tool

In this chapter, we present a NAS tool to enable the efficient exploration of custom search spaces. The tool leverages the efficiency of the search method described in the previous chapter, and provides a simplified way to apply it to new search spaces built using standard PyTorch syntax.

A special emphasis is placed on flexibility and ease-of-use. This is partly due to the inherent advantages provided by deep ensembles. Specifically, applying the search procedure does not require a custom search-space dependent distance function like in BO with Gaussian Processes, or custom operators like crossover and mutation in evolutionary algorithms. The search space can be constructed in a simple way starting from vanilla PyTorch model code, and the components of the search procedure, such as pretraining and multi-fidelity search, are handled automatically.

6.1 Motivation

Neural Architecture Search (NAS) has been a driver for many NN architectural innovations, helping to find many state-of-the-art models especially in computer vision, e.g. NASNet (Zoph et al., 2018) and EfficientNet (Tan and Le, 2019). However, the potential of NAS extends beyond finding reference architectures, as its usefulness can extend to a broader set use cases and a reach a wider audience.

In fact, directly applying reference architectures from the literature is often suboptimal for particular use cases. An example of this is the usual practice of providing a set number of variants of various sizes (number of parameters) for a given model family. These variants often differ not only in depth (number of blocks or layers), or width (number of channels), but also certain architectural choices at times (for instance, the number of ViT blocks in a ConvNet-ViT hybrid). For a particular use case, we might be seeking different compromises than the ones struck by the provided variants, for instance accepting a slightly bigger model than one particular variant for more performance, or vice versa. Use case circumstances sometimes impose hard limits on memory consumption or latency, and adjusting the architecture then becomes a manual tuning procedure.

Automatically tuning architectures is a useful idea in this context. Using architectures in the literature as reference points, general deep learning practitioners could automatically find the best models to suit their specific application and constraints. As a result, specificities about data, available resources, and specific quality metrics can be addressed in a more principled way by automatically tuning available architectures to these characteristics, ensuring that the model being used is the best one for the task at hand, and avoiding suboptimal models produced by manually tuning reference architectures.

However, in constrast to quantization and hyperparameter optimization, NAS is not usually a common step in model deployment pipelines. The practical implementation of NAS currently carries certain limitations:

- **General-purpose blackbox optimization tools:** Using a general-purpose solution often requires specialized knowledge of the search strategy, and difficult design decisions which also depend on the search space and thus might have to be redefined for each new search space. For instance, the user would need to define mutation and crossover operators suitable for how their search space is structured for EAs, or to select a kernel and a distance function for Gaussian Process-based BO. The expensive costs are an additional concern, as certain search strategies (RL, EAs) can require a big number of evaluations, which is inconvenient for NAS problems.
- **Search spaces in NAS frameworks:** NAS-specific frameworks such as Vertex AI (Google Cloud, 2024) or NNI (Microsoft, 2021) use a specialized design language or API for search space specification, while AutoKeras (Jin et al., 2023) uses similar syntax to the Keras functional API. While they offer advanced tools including access to weight-sharing

methods for some, the lack of flexibility in search space definition has the following limitations:

  – Restricting the expressiveness of the search space, where new and custom layer designs are impossible or very difficult to integrate. Often, the search space definition has to select from a limited set of possible design components.
  – New architectural innovations are published very frequently in the literature, often tested on large reference benchmarks. It is not a straightforward process to build search spaces from publicly available architecture code in a seamless way. Existing architecture code often written using deep learning frameworks such as PyTorch (Paszke et al., 2019), needs to be converted to the NAS tool's specialized design language in order to be incorporated in a search space.

Summary of contributions

We propose a simplified search tool, offering efficiency in the resources needed to achieve good results, and flexibility by design, allowing for an easy and open definition of the search space and objective function. Interoperability with existing code or newly released architectures from the literature allows the user to quickly and easily tune architectures to produce well-performing solutions tailor-made for particular use cases.

Our NAS tool is built with the following design objectives in mind:

- Fast search: the search method needs to find the best performing architecture using the fewest number of evaluations
- Few assumptions: the search space and objective function can be defined in a non-constrained way. The user can define any search space and any objective function.
- Simple incorporation of existing code: it is easy to include newer architectural advancements in the literature, custom-made architectural elements, specific and custom training and testing routines, etc...
- Sharing search spaces which can be directly applied to new objective functions (e.g. on a different dataset, using a different training recipe, or optimizing a different quality metric).

This is achieved by only requiring the user to provide the list of encodings their search space is made of, and a function to measure the quality of a solution represented by an encoding. Two optional additions can be made to significantly speed up the search:

- Providing an additional, lower-fidelity but faster objective function. For instance, if the full objective function is the validation accuracy after the model was trained for 300

epochs, the low-fidelity objective function could be the validation accuracy after only 15 epochs of training.

- Pretraining data: zero-cost metrics like number of parameters, FLOPs, latency or memory footprint, for a subset of the search space networks. If the user is optimising the architecture of PyTorch models, they can instead provide a function which takes an encoding and returns a PyTorch `nn.Module` object, and a sample input for it (e.g. a random image-shaped tensor for a vision model). The framework automatically generates the pretraining data.

This flexibility is intended to facilitate usage in the simplest use case, when a reference architecture has architectural parameters (e.g. width, depth, type of block or layer, or other more specific aspects) to be tuned to adapt to a use case. The search space definition is not limited to a set of existing components, allowing for even more expansive search spaces, incorporating multiple design patterns, hybridisation of many model families, or searching for entirely new blocks and layers.

6.2 Search method summary

A more detailed overview of the search strategy used is in chapter 5. We list the key points here:

- The search strategy is based on Bayesian Optimization with a deep ensemble as a predictive model. This allows a greater flexibility and ease-of-use, with no search-space dependent operators. The choice also stems from the desire to exploit different data sources to accelerate the search.

- More specifically, we use a number of zero-cost metrics, as well as combining low and high quality evaluations of the objective function. This additional data improves the shared internal representation, improving the model's predictive performance in a cost-effective manner and by extension accelerating the search.

- The combination of the deep-ensemble based BO search, and these two additional improvements, creates a fast NAS method with accelerated search times, reaching the optimum on NAS-Bench-201 in fewer evaluations than competing methods: under 80 evaluations for CIFAR10, and under 60 evaluations for CIFAR100.

- It has also been tested successfully on very different search spaces, like cell-based vision search spaces, and graph-based search spaces such as NAS benchmarks with reasonable computing resources (Chapters 7 and 8).

6.3 NAS framework usage

Search space definition

A `SearchSpace` object is created for each new search space. To define it, the user only needs to provide the encodings, and optionally a function to convert from an encoding to a PyTorch `nn.Module`.

```
ss = create_search_space(name = 'search-space',
                         save_filename = 'ss.dill',
                         encodings = encodings,
                         encoding_to_net = encoding_to_net)
```

The `preprocess` function automatically generates the pretraining data, using the `PyTorch Profiler` tools, then creates and pretrains the deep ensemble associated with this search space. It requires a sample input suitable for the networks in the search space, e.g. here we generate a random image-shaped tensor for a vision search space, for the profiling step. The user can instead provide custom pretraining data if for instance a different deep learning framework is in use.

```
ss.preprocess(sample_input = torch.rand(16, 3, 224, 224),
              threads = 16)
```

Reusability

The `SearchSpace` object is saved to a file which can be shared, providing a pretrained deep ensemble ready to be used for the search step immediately. It enables the launch of a new search on a user-defined objective function, e.g. on a new dataset, or using new data augmentation or training techniques.

Launching the search

To initiate a new search, the user defines the objective function (the high-fidelity evaluation), as well as an optional but recommended low-fidelity function. This is defined in a `SearchInstance` object, which encapsulates the current search progress, logs, and data. It is saved in a file, which can be loaded using the `dill` package to resume a previous search.

```
s = SearchInstance(name = 'search-inst',
                   save_filename = 'search.dill',
                   search_space_filename = 'ss.dill',
                   hi_fi_eval = hi_fi_eval,
                   hi_fi_cost = 240,
                   lo_fi_eval = lo_fi_eval,
                   lo_fi_cost = 12)
```

To run the search for an evaluation budget n:

```
s.run_search(eval_budget = n)
```

For image classification, a helper function provides code to train in distributed mode (using `torchrun`) for a user-specified number of epochs and then test the networks. Any dataset from the `datasets` package can be specified.

```
evaluator = create_img_class_evaluator(dataset = dataset,
                                       n_classes = num_classes,
                                       n_gpus = n_gpus,
                                       config_to_model_file = filename,
                                       dataset_config = dataset_config,
                                       eval_split = eval_split,
                                       reparam = True)
```

6.4 Conclusion

In this chapter, we described a simplified framework to perform NAS quickly and with maximum flexibility for incorporating custom architectural components. It mainly relies on Bayesian Optimization with deep ensembles, a pretraining scheme and multiple fidelities to accelerate the search. We argue that this method can be used effectively to improve existing architectures or find new ones, with the simple tuning and adaptation of reference architectures from the literature as a target use case. In chapters 7 and 8, starting from baseline architectures from the literature, we construct search spaces of varying types and with different complexity levels, and launch search instances using this tool to find the top-performing architectures on different datasets.

Part III
Efficient vision model search spaces

7 Cell-based CNN-ViT hybrid architecture search spaces

In this chapter, we illustrate the application of the previously described NAS method and tool on custom-built cell-based search spaces. We first start by constructing a hybrid search space generalizing two reference ConvNet and ConvNet-ViT architectures. Then, we showcase how this search strategy can be used in the context of one family of architectures by tuning its architectural parameters, such as depth, width, the use of Squeeze-and-Excitation, and the structural re-parameterization scheme. Finally, another hybrid CNN-ViT search space is constructed using cells from different architectures. This time, the experiment focuses on exploring the search space using different datasets. This demonstrates that the top-performing architecture can vary depending on the dataset, and makes the argument for using NAS to refine and tune architectures for custom use cases.

7.1 Introduction

In this chapter, we use the NAS approach and the tool presented in chapters 5 and 6 to search for efficient vision models using cell-based architecture designs. For each of these search spaces, our approach is as follows:

- Select a number of high-perfoming base design elements, including ConvNet variants and ViT designs. The elements are usually blocks, but can also be certain patterns and techniques such as including Squeeze-and-excitation (Hu et al., 2018), or structural reparameterization (Ding et al., 2019, 2021a,b).
- Create a search space generalizing the selected designs, where the set of possible architectures includes the base design patterns, as well as all the hybrids in-between them. In this chapter, the search space design is relatively simple but sufficiently expressive to contain high-performing architectures
- Use the tool to apply the BO NAS method on these custom search spaces. A subset of ImageNet (Howard, 2019a,b) is used to reduce search costs, using the accuracy on this smaller dataset as a less expensive alternative.
- Test the resulting architecture on the full ImageNet-1k set (Russakovsky et al., 2015), and note the improvement over the original designs which were detrimental to the performance

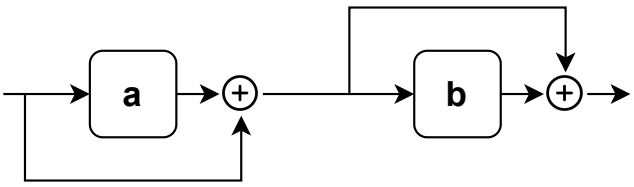
The goal is to test the NAS method and tool on a diverse set of real custom search spaces, and to perform the search using reasonable resources. Since the architecture of the ensemble networks used are generic feedforward networks and not tailored to the graph-based benchmarks, no tuning of the hyperparameters or architectures was performed.

7.2 MOAT and MobileNetV2 hybrid ConvNet-ViT search space

7.2.1 Search space design

MOAT (Yang et al., 2022) is a family of hybrid ConvNet-ViT networks for vision tasks. They are built on a hybrid block which effectively merges mobile convolution with transformer blocks: the MOAT block. The design of the MOAT block involves replacing the MLP in the transformer block with a mobile convolution, and rearranging the components.

Our choice for the MOAT architecture is justified by its design: it is close enough to both the MBConv and ViT architectures, therefore it is possible to find a common pattern generalizing all three.



Block	(a) miniblock	(b) miniblock
MBConv (Sandler et al., 2018)	MBConv & SE	Zero
ViT (Dosovitskiy et al., 2020)	Self-attention	MLP
MOAT (Yang et al., 2022)	MBConv, no SE	Self-attention

Figure 7.1: Meta-architecture of a cell in the search space

Starting with the architecture of the MOAT block, we built a search space of hybrid ConvNet-ViT networks. It contains purely mobile convolution-based networks including MobileNetV2 (Sandler et al., 2018), purely transformer-based networks including ViT (Dosovitskiy et al., 2020), and various hybrid configurations including MOAT.

MOAT has a cell-based design, made up of 4 stages, where the image resolution is divided by 2 at each new stage. An architecture is described by which cell design is selected for each section of the network. Our optimization objective is to find the optimal sequence of cell types associated with the stages. This way, we allow sufficient flexibility, but avoid having an unreasonably large space potentially full of ineffective or unfeasible architectures. This strategy keeps the search space at a similar size to benchmark search spaces like NAS-Bench-201.

The possible cell designs were constructed by leveraging an architectural pattern common to the MOAT block, the ViT block and the MBConv block. They can all be described using a succession of two mini-blocks, each with a skip connection. Figure 7.1 illustrates this pattern.

The cell types are described by specifying the miniblocks **(a)** and **(b)** from a list of possible miniblocks:

{MBConv with Squeeze-Excitation (SE), MBConv with no SE, Self-attention, MLP, Zero}.

We eliminate unfeasible designs, resulting in a total of 7 possible cell designs. We add two experimental cell types, with miniblocks **(a)** and **(b)** in parallel.

7.2.2 Experiments and results

The objective function is the validation score on Imagewoof (Howard, 2019b), a challenging 10-class subset of ImageNet-1k. As explained in 3.4.3, using ImageNet-1k directly significantly increases costs, and using an ImageNet subset is a better alternative to using other small datasets like CIFAR-10. The high-fidelity evaluation is computed after 200 epochs of training

Table 7.1: Performance comparison of the searched architecture and the baseline MOAT architecture

Experiment	Baseline	Ours
Imagewoof - 200 epochs (Objective function)	81.13	83.76
Imagewoof - 300 epochs	84.34	86.54
Imagewoof - 300 epochs (MOAT-0 channels)	89.82	90.86
ImageNet-1k - 90 epochs	61.84	67.74

epochs, and the low-fidelity evaluation is after 15 training epochs. We compute 4 pretraining metrics using PyTorch Profiler: FLOPs, inference time, GPU memory footprint and number of trainable parameters.

We use the `tiny-moat-0` configuration and number of channels as a baseline. It is a small network of around 2.7M parameters, and the ImageNet results should be interpreted in comparison to this baseline network and not performances of much bigger networks.

The baseline architecture **tiny-moat-0** is as follows:

```
(MBConv) | (MBConv) | (MOAT block)| (MOAT block)
```

The search procedure yielded the following architecture after approximately 70 evaluations:

```
(MBConv no SE) | (MOAT block) | (MBConv & SE + MLP) | (MOAT block & SE)
```

We report the validation results on Imagewoof and ImageNet-1k (with no pretraining) in table 7.1. We also test a bigger version of the baseline and the searched architecture, using the `MOAT-0` channel sizes. The search procedure has produced a better-performing architecture than the baseline we started with.

7.3 MobileOne-based ConvNet search space

7.3.1 MobileOne architecture

MobileOne (Vasu et al., 2023b) is a family of efficient vision backbones targeted towards mobile devices. They are purely ConvNet-based, with numerous improvements to traditional efficient ConvNet designs aimed at minimizing the latency on a mobile device.

One key aspect of the MobileOne architecture is re-parameterization (section 2.3), building on advances in (Ding et al., 2019, 2021a,b). Different architectures are used during training and inference. At train time, parallel branches have convolution and batch normalization operations with diverse kernel sizes. At inference time, these branches are fused into an equivalent block with a much simpler architecture. This design leads to performance improvements across many vision tasks (image classification, object detection, semantic segmentation) without sacrificing low latency.

7.3.2 Search space description

Setting the MobileOne architecture as a baseline, we start with the `MobileOne-s0` variant, we vary the depth and width in each of its four stages. We also set whether Squeeze-and-Excitation (SE) is used, and the number of additional convolutional branches (i.e. the number of parallel branches added to the train-time model). These design choices define a search space around the MobileOne architecture. (Table 7.2).

We aim to search for a better configuration of the `MobileOne-s0` architecture on a different dataset, with a similar number of total parameters. The total depth, or total number of blocks, for `M1-s0` is 21 distributed over 4 stages. The number of parameters of the inference-time model is 1.06M.

Therefore, we filter the resulting search space as follows:

$$19 \leq \text{Total depth} \leq 22$$
$$0.86\text{M} \leq \text{n. params} \leq 1.26\text{M}$$

This yields a total of 55870 architectures. We perform the search using as objective function the validation accuracy on the Imagenette (Howard, 2019a) image classification dataset, a 10-class subset of ImageNet. The high quality evaluation has 240 epochs during training, while the low quality evaluation has 12 epochs.

Table 7.2: MobileOne-based ConvNet search space

	Stage 1		Stage 2	
Base width	**64**		**128**	
	Depth	**Width multiplier**	**Depth**	**Width multiplier**
MobileOne-s0	2	0.75	8	1.0
Value ranges	{0,1,2,5,8,10}	[0.25..4] in steps of 0.25	{0,1,2,5,8,10}	[0.25..4] in steps of 0.25 (≥ Stage 1)

	Stage 3		Stage 4	
Base width	**256**		**512**	
	Depth	**Width multiplier**	**Depth**	**Width multiplier**
MobileOne-s0	10	1.0	1	2.0
Value ranges	{0,1,2,5,8,10}	[0.25..4] in steps of 0.25 (≥ Stage 2)	{0,1,2,5,8,10}	[0.25..4] in steps of 0.25 (≥ Stage 3)

	Use SE	**N. conv. branches**
MobileOne-s0	No	4
Value ranges	{Yes, No}	{1, 4}

7.3.3 Search results

We run the search for the equivalent of 60 evaluations. We compare the resulting architecture and the baseline `M1-s0` architecture's results in table 7.3, where we report the value of the high quality evaluation (equivalent to the objective function during the search) i.e. 240 epochs, as well as the evaluation at 300 epochs.

The search process successfully found a smaller and better-performing depth and width configuration for the M1-s0 model, as it applies to this dataset.

Table 7.3: Search results for the M1-based search space on Imagenette

	N. params (train-time)	N.params (test-time)	240 epochs	300 epochs
MobileOne-s0	4.28M	1.06M	89.34	90.09
Search result	0.95M	0.93M	**90.37**	**90.77**

7.4 SwiftFormer and MobileOne hybrid ConvNet-ViT search space

7.4.1 SwiftFormer architecture

SwiftFormer (Shaker et al., 2023) introduces a novel attention mechanism designed with low latency in mind: efficient additive attention. Instead of an expensive operation with quadratic complexity w.r.t the input resolution, self-attention is implemented as a fast linear operation.

In the SwiftFormer family of models, the architecture is a succession of stages separated by downsampling steps, each stage being a succession of convolution-based blocks followed by one attention-based block. This is an original setup only possible because of the efficiency of the self-attention mechanism: where efficient ViT-CNN models generally reserve attention modules to the latter stages where the resolution is lowest, SwiftFormer can apply self-attention anywhere in the network, with minimal impact to efficiency and latency.

7.4.2 Search space description

Starting from the SwiftFormer-XS variant, we built a search space incorporating and generalising both MobileOne-s0 and SwiftFormer-XS architectures. For each stage, 2 variables have to be set: the type of convolution block to use, and the number of attention blocks at the end of the stage. In the attention block(s), the attention mechanism is preceded by a mini-convolution block. This is of the same type as the other convolution blocks in the stage.

- `conv-enc`: SwiftFormer's convolution-based block design
- `mo`: MobileOne block
- `mo-se`: MobileOne block with SE (Squeeze-and-Excitation)

For each stage, 12 possible combinations exist, and with 4 stages the search space spans $12^4 = 20736$ architectures, including SwiftFormer-XS and MobileOne-s0.

Table 7.4: MobileOne-SwiftFormer hybrid search space

	Stage 1		Stage 2	
	Conv. type	**Attn. blocks**	**Conv. type**	**Attn. blocks**
SwiftFormer	`conv-enc`	`last-1`	`conv-enc`	`last-1`
MobileOne (s0-s3)	`mo`	`none`	`mo`	`none`
Value ranges	{`conv-enc`, `mo`, `mo-se`}	{`none`, `last-1`, `last-2`, `all`}	{`conv-enc`, `mo`, `mo-se`}	{`none`, `last-1`, `last-2`, `all`}

	Stage 3		Stage 4	
	Conv. type	**Attn. blocks**	**Conv. type**	**Attn. blocks**
SwiftFormer	`conv-enc`	`last-1`	`conv-enc`	`last-1`
MobileOne (s0-s3)	`mo`	`none`	`mo`	`none`
Value ranges	{`conv-enc`, `mo`, `mo-se`}	{`none`, `last-1`, `last-2`, `all`}	{`conv-enc`, `mo`, `mo-se`}	{`none`, `last-1`, `last-2`, `all`}

Table 7.5: Search results on the Imagenette dataset

Architecture	Accuracy (240 epochs)
MobileOne-s0	89.94
SwiftFormer-XS	91.08
Search result (4 branches)	91.18
Search result (1 branch)	92.05

7.4.3 Search results

We perform the search using two different datasets. Along with Imagenette, we also use the Alzheimer MRI disease classification dataset (Falah.G.Salieh, 2023), a 4-class dataset which seems to be slightly more challenging for the tested networks. For Imagenette, two search instances were tested, one with the number of branches in the MobileOne blocks set to 4, and a second one with this value set to 1.

Tables 7.5 and 7.6 contain the search results.

Figures 7.2 and 7.3 illustrate the evolution of the best found scores as the search progressed.

Table 7.6: Search results on the Alzheimer-MRI dataset

Architecture	Accuracy (200 epochs)
MobileOne-s0	74.53
SwiftFormer-XS	64.60
Search result	76.95

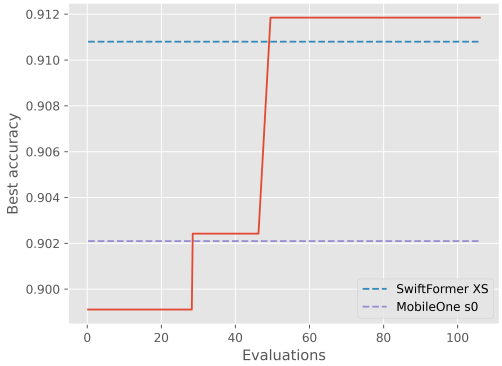


Figure 7.2: Best value evolution during search - Imagenette (4 branches)

7.5 Conclusion

In this chapter, the NAS method (chapter 5) and tool (chapter 6) were tested on a number of cell-based search spaces. In section 7.3, it is used to tune an existing architecture's basic architectural attributes, for instance depth and width. Sections 7.2 and 7.4 showcase how this method can be used to efficiently find the top-performing architectures in search spaces which generalize reference architectures or represent hybridization of different architectural designs.

In the next chapter, we will construct a more complex search space with conditional variables and mixed variables. We will look specifically to optimize the design of the most important operation in vision transformers: the self-attention mechanism.

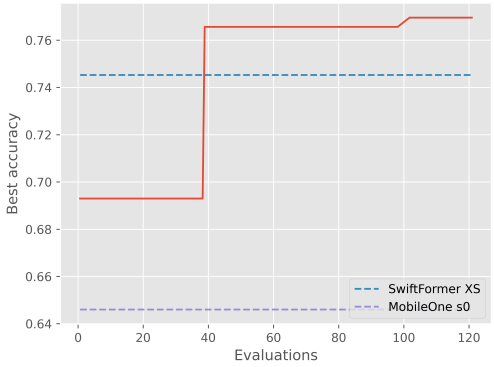


Figure 7.3: Best value evolution during search - Alzheimer-MRI dataset

8 Efficient self-attention mechanism search space

In this chapter, we test the search strategy on a more ambitious and challenging search space: the goal is to search for an efficient re-design of the self-attention mechanism at the center of vision transformers and ConvNet-ViT hybrid models.

We start by presenting the general principles according to which the search space is designed. The resulting search space has discrete and continuous variables, as well as conditional variables.

The search scheme is applied to this search space, and the experiments show that the it yields a better-performing design which boosts the accuracy while simultenously allowing for wider models at the same latency level as the baseline.

8.1 Introduction

As highlighted in chapter 2, vision transformers (ViT) have challenged the dominance of ConvNets in high-performing vision model architectures. An integral component in this architecture is the self-attention (SA) mechanism, which scales in $O(n^2)$ with the number of input tokens.

For mobile vision applications on resource-contrained devices, where latency is a crucial concern, the focus has been on efficient hybrid ConvNet-ViT designs (Mehta and Rastegari, 2021; Li et al., 2022; Mehta and Rastegari, 2022; Vasu et al., 2023a). In these designs, the self-attention mechanism is usually used sparingly because of its big impact on the latency. In fact, the ViT blocks are usually only reserved for the latter stages in the architecture, where the input resolution is lowest. In other words, the latency cost of the quadratic SA mechanism is too high to use in the high-resolution early stages. Instead, the high resolution stages are exclusively occupied by efficient ConvNet blocks based on fast convolution variants like depthwise-separable convolution.

Similarly, fast designs of the self-attention mechanism have been explored in recent architectures (section 2.5), with the aim of reducing the inference latency while preserving the accuracy improvement ViT models and blocks provide. Ideas include reducing the dimension with regard to which the mechanism has quadratic complexity (Maaz et al., 2022), or reducing the quadratic complexity to linear complexity (Mehta and Rastegari, 2022; Shaker et al., 2023). With these more efficient designs, some works such as SwiftFormer (Shaker et al., 2023) have enabled a self-attention block at the tail end of every stage of the network, including the high resolution stages.

With the aim of finding an efficient and high-performing SA mechanism design, specifically targeted for mobile vision applications, we apply the NAS method and framework presented in part II on a novel search space for efficient self-attention mechanisms. Since the different designs heavily affect the latency, we adjust the size of the model (specifically, the width, i.e. number of channels) for each SA design to maintain the same latency throughout the entire search space.

In this chapter, we make the following contributions:

1. We create a meta-design of efficient self-attention (SA) mechanisms, which generalizes many designs in the literature and contains a large number of new designs,
2. We create a search space around this meta-design, containing ConvNet-ViT hybrid networks. Binary search is used to tune the size of each network via its width, ensuring all networks in the search space have comparable latencies,
3. We perform fast automatic search using Bayesian Optimization, powered by a pretrained deep ensemble and multi-fidelity search

We summarize the result of this search as follows:

- Using NAS, we design a low-latency high-performing self-attention mechanism.
- We build a series of small models we call SnapFormer, aimed for mobile use cases
- Our models achieve higher accuracy than competing architectures at the same latency. Our S variant reaches 79.1% top-1 ImageNet-1K accuracy, while being as fast as SwiftFormer-S, and twice as fast as MobileViT-v2.

8.2 Efficient attention mechanisms

The self-attention mechanism is the centerpiece of vision transformers and by extension an important part of hybrid ConvNet-ViTs. It enables the injection of global context to inform the local representations of sections of the image. This is achieved by computing a representation of token **interactions**, and combining it with local representations of the tokens.

In practice, an input $\mathbf{x}$ is an $n \times d$-dimensional tensor, for n tokens represented by d-dimensional embeddings. Trainable weight matrices $\mathbf{W}_Q$, $\mathbf{W}_K$ and $\mathbf{W}_V$ are used to project $\mathbf{x}$ to query ($\mathbf{Q}$), key ($\mathbf{K}$) and value ($\mathbf{V}$) tensors. Token interaction scores are represented by query-key interactions, computed by scaling and applying Softmax to the dot product of $\mathbf{Q}$ and $\mathbf{K}^\intercal$. The value tensor $\mathbf{V}$, which propagates the local representations, is combined with the global context using dot product. Multi-head self attention (MHSA) duplicates this process h times for h attention heads, allowing for more variety and better attention capability.

$$\mathbf{G}_{\text{context}} = \text{softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^\intercal}{\sqrt{d}}\right) \in \mathbb{R}^{n \times n} \tag{8.1}$$

$$\mathbf{x}_{\text{output}} = \mathbf{G}_{\text{context}} \cdot \mathbf{V} \in \mathbb{R}^{n \times d} \tag{8.2}$$

The original self-attention mechanism has a strong representation capacity, but the complexity of its operations is $O(n^2 d)$. This makes it a real bottleneck for latency-critical use cases, especially on resource constrained devices.

Many improvements have been proposed to alleviate this computational bottleneck while retaining the accuracy advantage ViTs brings to the table. We break them down along the two following main ideas:

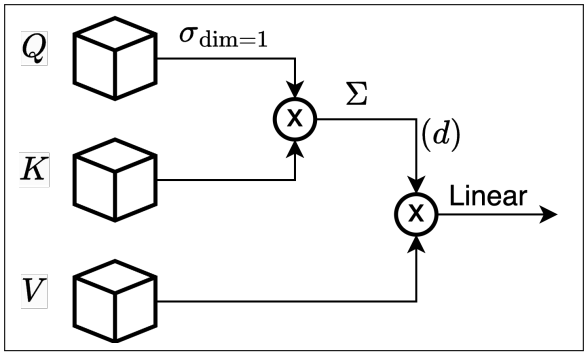


Figure 8.1: Separable self-attention in MobileViTv2 (Mehta and Rastegari, 2022)

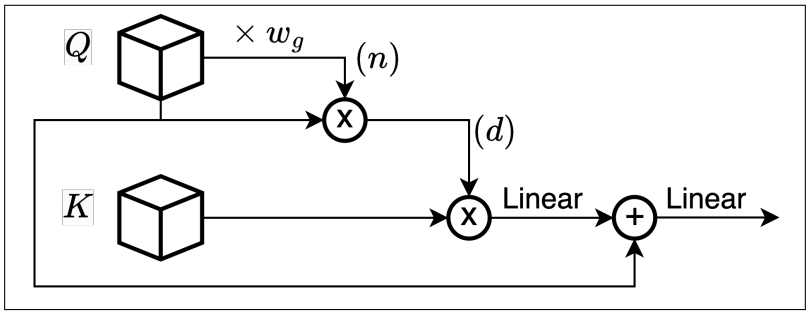


Figure 8.2: Efficient additive attention in SwiftFormer (Shaker et al., 2023)

Reducing the dimensions of the product operations

The idea is to minimize the cost of the product operations by avoiding computations with $O(n^2)$ complexity.

A notable example is transpose self-attention, introduced with the EdgeNeXt architecture (Maaz et al., 2022). In this design, the **Q**-**K** interactions are instead represented by $\mathbf{G}_{\text{context}} \in \mathbb{R}^{d \times d}$. It is obtained by computing $\mathbf{Q}^\intercal \cdot \mathbf{K}$. The values usually taken by n and d, and the complexity of $O(nd^2)$, contribute to a reduction of latency with a relatively minimal impact to the accuracy of the model.

This approach is taken further by separable self-attention from MobileViT-v2 (Mehta and Rastegari, 2022) (Figure 8.1), and efficient additive attention from SwiftFormer (Shaker et al., 2023) (Figure 8.2). Both of these self-attention mechanisms produce a d-dimensional context vector, rendering the final product a simpler and faster multiplication between a d-dimensional vector and an $n \times d$-dimensional local representation tensor.

Using fewer linear projections

The reliance on all three of the **Q**, **K** and **V** linear projections appears to be an area of potential latency savings.

SwiftFormer (Shaker et al., 2023) forgoes the **V** matrix entirely, relying only on **Q** and **K**. It instead uses the **Q** matrix to produce the d-dimensional vector, and propagates the local representations using **K**. A linear layer is also applied to the product output, and is added to **Q** to produce the final output.

8.2.1 Search space of efficient attention mechanisms

We create a search space comprising 10560 possible designs for the self-attention mechanism, based on the two design principles detailed previously. More specifically, we allowed the possibility of operating with reduced dimensions for the inputs of the product operations, and allowed flexible reuse of linear projections. As a result, the space we used in the search generalizes many efficient designs of the self-attention mechanism, such as the Efficient Additive Attention from SwiftFormer (Shaker et al., 2023), and provides room to explore many other alternative designs.

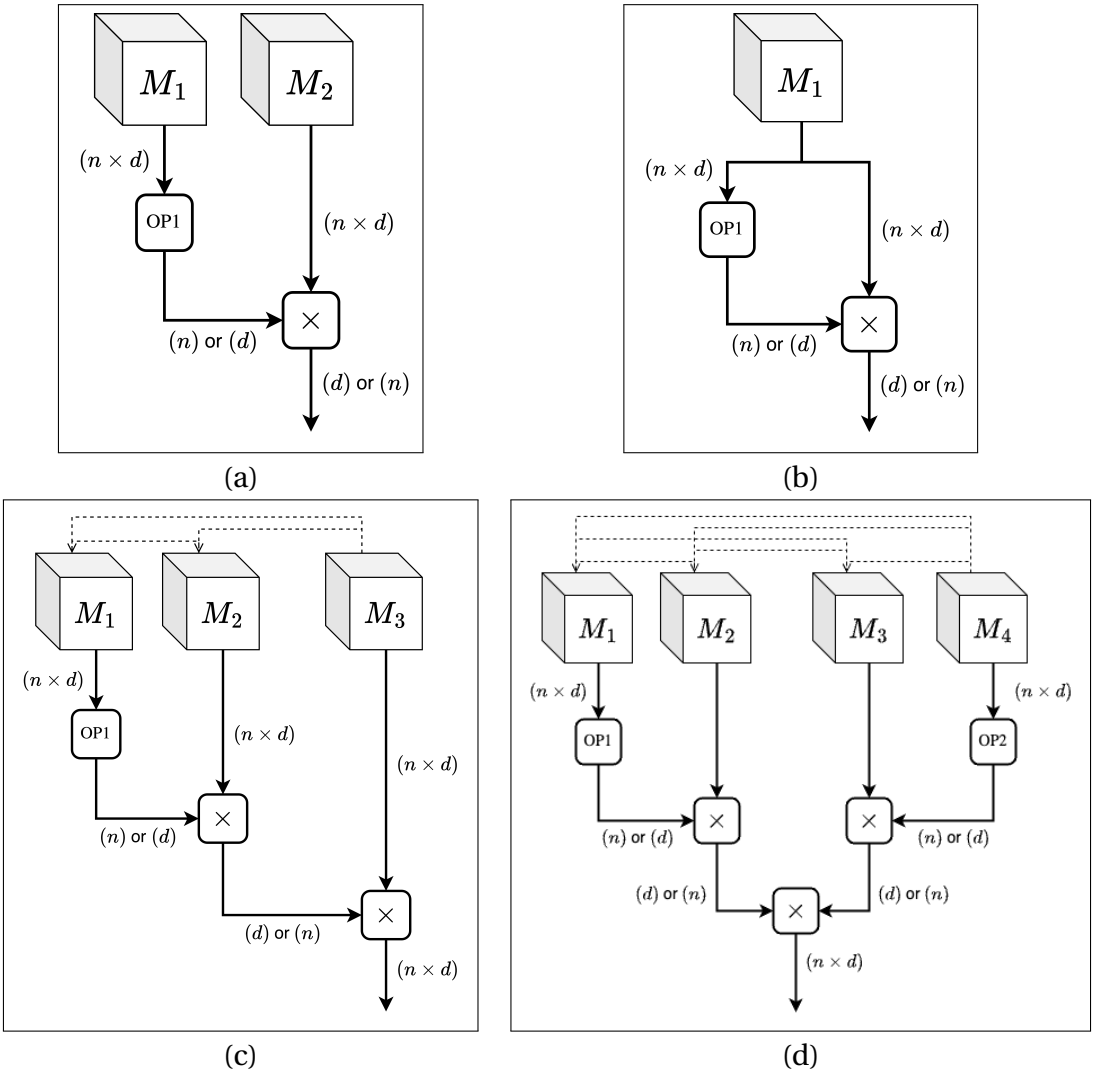


Figure 8.3: Self-attention mechanism search space: (a) A branch with a 1-dimensional output, using two linear projections; (b) A branch with a 1D output, using only one linear projection; (c) Cell design where branch 2 has a 2D output. The dotted lines indicate that some of the linear projections may be reused; (d) Cell design where both branches produce 1D outputs.

Branch 1

In this branch, information is encoded in a 1D, n-dimensional or d-dimensional vector. In the case of a d-dimensional output, a projection of the input $\mathbf{M}_1$ is reduced to an n-dimensional embedding by multiplying it with a learnable weights vector $\mathbf{w} \in \mathbb{R}^d$. In the case of an n-dimensional output, we sum along the d-dimension.

The vector undergoes a operation which preserves its dimensions: a normalization or an activation function from this set Softmax, GeLU, SiLU. The resulting 1D vector is multiplied

with $n \times d$-dimensional projection $\mathbf{M}_2$ to produce the output of the branch (Figure 8.3 - a). $\mathbf{M}_1$ $\mathbf{M}_2$ can be the same projection (Figure 8.3 - b).

Branch 2

This part can output an $n \times d$ projection, or a 1D-vector like branch 1. It can reuse any of the projections in branch 1, or use a new projection.

Combination

The combination step contains the elementwise multiplication of the outputs of branch 1 and 2. It can be summed up as follows:

$$\mathbf{x}_{\text{output}} = proj_2(proj_1(\mathbf{b}_1 \cdot \mathbf{b}_2) + \mathbf{M}_{\text{comb}}) \tag{8.3}$$

$\mathbf{M}_{\text{comb}}$ can be any of the projections used by the two branches. Each layer proj_j can either be identity or a linear layer.

Width scaling coefficient

We use the SwiftFormer-XS width list as a reference point. We scale these widths using a width scaling coefficient, thereby modulating the size of the model. The purpose is to account for the difference in latency between the attention mechanisms contained within the search space. For each mechanism design, we find the appropriate width scaling coefficient to obtain the same inference latency as SwiftFormer-XS. As a result, all networks in our search space have the same latency.

8.2.2 Search procedure overview

We ran the search with a budget of around 80 evaluations, using the accuracy on Imagewoof (Howard, 2019b) as a proxy for the objective function.

Encoding details

The encoding used to represent the self-attention mechanisms in the search space has 10 dimensions, with an additional variable for the width scaling coefficient.

```
(b1_dim) | (b2_dim) | (b1_op) | (b1_mat2) | (b2_mat12)
(b2_op) | (b2_mat3) | (b2_mat4) | (comb_type) | (comb_mat)
```

The first two variables `b1_dim` and `b1_dim` set the dimensions of the two branches, which define the type of cell used. The definition of the `bi_matj` variables depends on the dimensions of the branches, and allow for the reuse of projections. Some of them are conditional, for instance `b2_mat12` is only defined when branch 2 has an $n \times d$ output, which automatically sets `b2_op`, `b2_mat3`, `b2_mat4` to the unused value 0.

8.3 SnapFormer architecture

8.3.1 Low-latency attention mechanism

The search procedure produces a design with a number of differences with the baseline design used by SwiftFormer (Figure 8.2). The best-performing design for the attention mechanism (Figure 8.4) is structured as follows:

- d-dimensional first branch, produced using two projections
- n-dimension second branch, produced using one new projection and reusing a projection from branch 1
- Both $proj_1$ and $proj_2$ are the identity operator

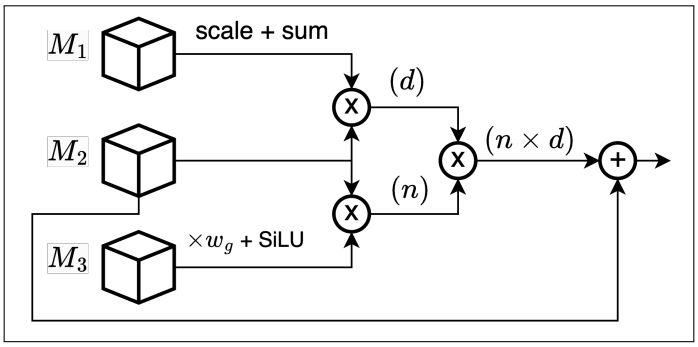


Figure 8.4: The low-latency self-attention mechanism found using NAS and used to build the SnapFormer models

Before the search, the width scaling coefficient for this design was assigned a value of $c_{\text{width}} = 1.1$. This indicates that its inherent latency is lower than SwiftFormer's Efficient Additive Attention mechanism, as the width scaling scheme had to increase the network's width to match the baseline latency.

The faster design is explained by what's contained and omitted from the design: it has fewer final linear projections than the baseline Efficient Additive Attention, and both multiplication operation is performed between two 1D vectors, as opposed to one 1D × 1D and one 2D $times$ 2D multiplications in the baseline.

Meanwhile, the presence of three initial projections M_1, M_2 and M_3 instead of two in the baseline seems to cost less in latency than the aforementioned optimizations, without sacrificing predictive performance.

Since the search space contains the baseline design as well, the mechanism designed using the search method outperforms it. This means that the combination of the lower-latency mechanism coupled with a wider model (using the width coefficient) combine to produce a model with higher performance at the same latency as the baseline design.

We perform further latency comparisons in subsection 8.4.

8.3.2 SnapFormer architecture description

Macro-architecture

We follow the same design as SwiftFormer when it comes to the global architecture of the ConvNet-ViT hybrid, as our focus is the self-attention mechanism itself, and the improvements introduced by SwiftFormer in the ConvNet blocks and the integration of the self-attention mechanism perform very well. The following design principles describe it:

- The networks have 4 stages, separated by downsampling layers which progressively reduce the resolution (and therefore the number of tokens) and increase the channels
- Each stage starts with a succession of convolutional blocks followed by a number of ViT blocks
- The convolutional block performs the following computation:

```
(3x3 Dwise-Conv) + BN → Pwise-Conv + GELU → Pwise-Conv
```

- The ViT block performs the following computation:

```
(3x3 Dwise-Conv) + BN → Pwise-Conv → SA
 → Pwise-Conv + BN + GELU → Pwise-Conv
```

Where Dwise is the depthwise-convolution operation, and Pwise is pointwise-convolution.

Width and number of ViT blocks

In the original SwiftFormer architecture, each stage ends with one ViT block. We experimented with the architecture's configuration considering the lower latency of the attention mechanism as well as the accuracies obtained.

Model variant		XXS	XS	S
Stage 1	*Layers*	3	3	3
	ViT layers	0	0	0
	Width	48	48	48
Stage 2	*Layers*	3	3	3
	ViT layers	2	2	2
	Width	56	64	64
Stage 3	*Layers*	5	6	9
	ViT layers	2	2	2
	Width	96	138	182
Stage 4	*Layers*	4	4	6
	ViT layers	2	2	2
	Width	192	232	256

Table 8.1: SnapFormer variants architecture details

- Number of ViT blocks: we use 0 ViT blocks in the first stage, and two ViT blocks at the end of the other stages. This increases the total number of ViT blocks from 4 to 6.
- Width (ie. number of channels): we match the baseline latencies achieved by SwiftFormer by slightly increasing the number of channels. This of course increases the total number of parameters. Since one of our design objectives was to maintain the same latency as the baseline architecture while obtaining a higher accuracy, we ensure that the depths used yield a model with the same latency as the baseline.

Table 8.1 breaks down the parameters used to construct each of the variants.

8.4 Experiments

Preliminary tests of the self-attention mechanism

We perform the following tests on a 40GB A100 GPU, using a batch of 64 random inputs, 1000 inference repetitions and 20 warmup repetitions.

We first replace the Efficient Additive Attention mechanism in SwiftFormer with the design we described in the previous section. This leads to a reduction in inference time. We then increase the model width to get closer to the baseline latency, with the aim of obtaining a model with better performance at the same latency cost.

Table 8.2 contains the latency information and the accuracy on the Imagewoof set. For both the XS and S variants, the low-latency self-attention mechanism found by the NAS scheme

	Params	Latency (ms)	Top-1 acc.
SwiftFormer-XS	3.47M	12.25	82.99
+ low-latency attention mechanism	3.41M	11.70	84.55
+ more channels: SnapFormer-XS	4.48M	12.13	86.30
SwiftFormer-S	6.09M	14.90	85.84
+ low-latency attention mechanism	6.01M	14.31	86.33
+ more channels: SnapFormer-S	7.86M	14.92	87.01

Table 8.2: Latency and accuracy results on Imagewoof

outperforms the baseline SwiftFormer networks. Scaling the widths further, we obtain models with roughly equivalent latencies and significant performance boosts over the baseline.

Image classification on ImageNet-1K

The training details on ImageNet-1K are the following:

- Epochs: 300
- Optimizer and scheduler: AdamW, cosine LR scheduler, $LR_{initial} = 0.001$
- PyTorch 2.1.0, Python 3.9, CUDA 12.2
- Latency/throughput measurement: batch of 64 random inputs, 1000 repetitions
- Distillation: Regnety160, with the same distillation scheme used by SwiftFormer (Shaker et al., 2023) and EfficientFormer (Li et al., 2022).

We summarize the results in table 8.3, grouping architectures based on their measured latency and size. We also plot the top-1 accuracy vs inference latency in Figure 8.5.

In the smallest models tier, our XXS achieves a good precision with relatively fewer parameters, but pure CNNs are still the best choice in terms of accuracy and latency.

As we move to bigger modules, both SwiftFormer and SnapFormer achieve good performance, often offering the best accuracy values at a particular latency range. Our SnapFormer S model has an accuracy of models with 1.5x its capacity.

8.5 Conclusion

In this chapter, we apply Neural Architecture Search using the deep-ensemble based BO method on a search space of efficient self-attention mechanisms. We designed a complex

	Params	Latency (ms)	Throughput	Top-1 acc.
MobileNetV2	3.50M	9.84	6504.3	71.8
MobileNetV3-large x0.75	3.99M	7.2	8886.1	73.3
MobileNetV3-large	5.48M	8.09	7907.5	75.1
SnapFormer XXS	**2.84M**	**10.22**	**6268.2**	**74.91**
SwiftFormer XS	3.47M	12.23	5230.1	75.7
SnapFormer XS	**4.48M**	**12.13**	**5276.2**	**76.96**
EfficientNet B0	5.29M	14.04	4557.2	77.1
MobileNetV2 x1.4	6.1M	14.27	4484.3	74.7
SwiftFormer S	6.09M	14.90	4295.6	78.5
SnapFormer S	**7.86M**	**14.92**	**4288.5**	**79.10**
PoolFormer S12	11.91M	15.96	4008.2	77.2
EfficientFormer L1	12.28M	15.57	4109.6	79.2

Table 8.3: Top-1 accuracy results on ImageNet-1K. Models are grouped together by latency and size.

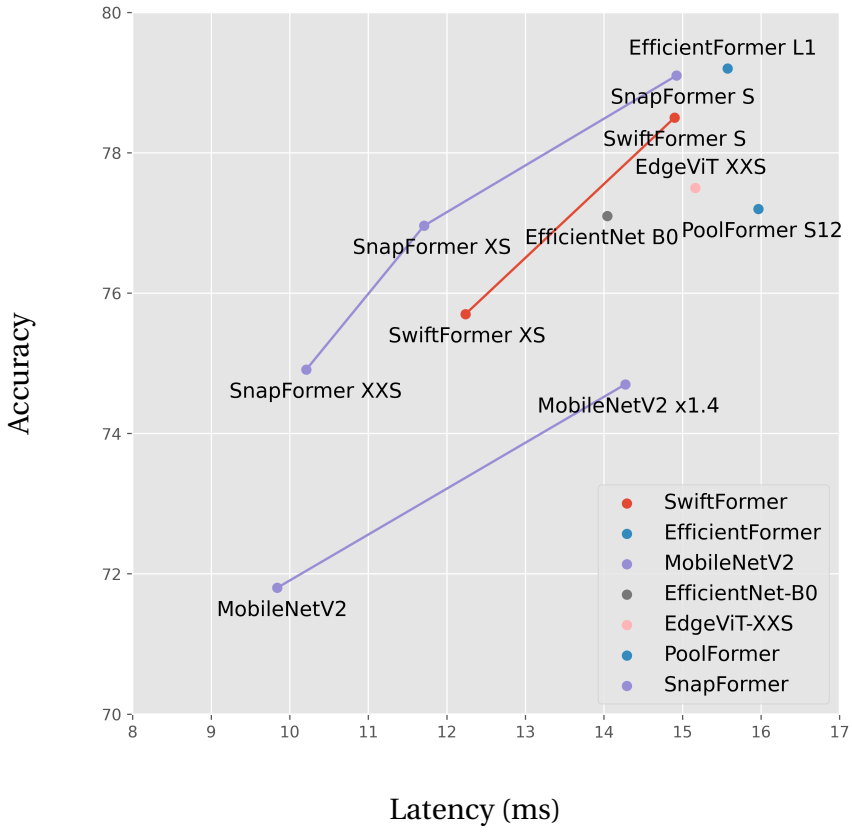


Figure 8.5: Latency vs accuracy plot in the targeted latency range

search space with conditional variables, mixed discrete and continuous variables (e.g. width scaling). In fewer than 80 total evaluations performed during the search, the top performing design exhibits good accuracy and latency properties. Based on this mechanism, we build a series of small ConvNet-ViT hybrid models, especially designed with low-latency in mind. Our SnapFormer XS and S models achieve high accuracies in image classification on ImageNet-1K, and can serve as capable and fast vision backbones for a large number of use cases.

9 Conclusion and future outlook

9.1 Thesis overview and conclusion

In the beginning of this manuscript, we started by highlighting the current rise of neural networks as exceptionally versatile and high-performing solutions to many difficult problems. Then, we presented the case for on-device and lightweight models. In this context, automatically optimizing the architectures of efficient vision models is a highly important and impactful endeavour.

With that in mind, we briefly introduced some of the most ubiquitous architecture families of efficient vision models in chapter 2, starting with ConvNets and reaching ConvNet-Vision Transformer models. These trends are highly important for designing high quality architectures for any downstream vision task.

The architectural patterns allow the construction of search spaces where we can look for higher performance or better efficiency or ideally both, potentially under hardware constraints concerning energy and memory. Exploring these search spaces in an efficient manner calls upon the techniques of Neural Architecture Search (NAS). Beyond the search space and optimization strategy, search efficiency is a central topic. It has motivated many approaches aiming to make NAS more practically viable and to reduce the prohibitive costs associated with the earliest iterations. An overview of NAS literature and a special emphasis on search efficiency techniques was the focus of chapter 3.

One blackbox optimization method in particular was central to our work: Bayesian Optimization (BO) offers great sample efficiency and opportunities to boost it in the context of NAS. We aimed to present a general overview of BO in chapter 4. While Gaussian Processes (GPs) are ubiquitous in BO methods, we also included a brief description of unorthodox BO variants based on alternative models. Drawing from these principles, a method for quickly identifying failure conditions of an image classifier using BO and a text-to-image model was described at the end of the chapter.

After the introductory part which dealt with the general context and motivations of our work, the following chapters are dedicated to our contributions. We introduce a NAS method based on Bayesian Optimization, powered by deep ensembles. Then, we showcase its practical application on a range of efficient vision model search spaces.

First, chapter 5 presents the core ideas of our BO-based NAS method. It motivates some of the design choices made, such as using a deep ensemble, with the design and implementation of improvements, namely simultaneous pretraining and multi-fidelity search. These improvements significantly accelerate the search efficiency, which can be showcased using NAS benchmarks.

In chapter 6, we present a software tool which enables the practical application of the aforementioned NAS method on custom-built search spaces. The design choices are motivated by providing maximum flexibility and ease-of-use to the user, enabling the usage of vanilla PyTorch model code. This removes significant barriers to the practical usage of NAS as it aims to eliminate the separation between search space definition in NAS contexts and architecture code found in literature or implemented using general deep learning frameworks.

Coupled with the efficiency of the NAS method, this tool has enabled us to focus on the second big topic of the thesis: using automatic architecture optimization for the design of efficient vision models.

This is the main objective in chapters 7 and 8. Our focus shifts on search space design using the patterns and elements of efficient vision models we highlighted earlier in the thesis: efficient convolutional blocks, structural re-parameterization, the self-attention mechanism, etc.

In chapter 7, the search spaces are cell-based, using of hybridization and generalization to construct promising search spaces from high-performing architectures existing in the literature. We also highlight how the search strategy consistently finds better-performing architectures using limited resources. Additionally, we showcase how different datasets yield different search results. This further supports the idea that the automatic tuning of architectures for specific use cases is a valid way to improve performance in a principled way, akin to hyperparameter optimization.

In chapter 8, the search space design takes a different approach: mixed and conditional variables are used, and a width modifier ensures that the latencies are uniform across the search space. This enables us to search for an efficient self-attention mechanism which eliminates certain elements from other efficient designs in the literature without sacrificing performance, allowing a wider model with higher performance at no additional latency cost as compared to the baseline.

9.2 Concluding remarks

Our work has focused on automatic architecture optimization for efficient vision models. This goal was approached in a first step by devising a NAS method with sufficient efficiency to go beyond benchmarks and to be practically applied on new search spaces, using limited computing resources. Then, we focused on creating diverse search spaces with the potential for including high quality architectures, basing our work on combining and generalizing well-established architectural patterns to find new models.

The successful application of the search strategy on these search spaces simultaneously highlights its versatility and practical usability, as well as the quality of the search spaces. It goes without saying that there remain many possible extensions and ideas for further improvements. There are also interesting questions to investigate and research directions to pursue. We describe these potential future directions in the next section.

9.3 Future outlook

The most intuitive extension of our work is to deploy it in the multi-objective case. In this case, additional objectives can be hardware-related, which makes it a hardware-aware NAS method. They can also be robustness or explainability metrics.

In fact, our approach in chapter 8 with the fixed latency across the search space can be thought of as an instance of the ϵ-constrained method (Talbi, 2009): one of the objectives, in this case the latency, is fixed to a specific value, and the optimization is performed on the other objective. If it is extended to multiple values of the latency, the search strategy will find different designs each time, along the Pareto front. This will in turn produce a Pareto set of efficient self-attention mechanisms, corresponding to different latency targets.

Other potential extensions and interesting future directions can be focused on either the search strategy or the search spaces described in the chapters of this thesis. On the search strategy level, a possible extension is in further adjusting the architecture of the deep ensemble to fit the search space better, potentially optimizing it using a NAS approach in conjunction with the quality on the pretraining metrics. The goal for this extension is primarily to increase the efficiency of exploring much bigger search spaces.

The joint optimization of hyperparameters and architecture was not explicitly explored in our work, although the search strategy can readily handle mixed variables with no additional modifications needed. As briefly mentioned in the early parts of this thesis, joint optimization is more optimal than performing NAS and hyperparameter optimization sequentially. However, joint optimization implies a higher number of dimensions and by extension a bigger search space. With limited time or computing resources, this presumably has a negative effect on the quality of the search results. Balancing these two aspects is important. Finding a relation between the available budget -in terms of time or hardware resources- and the corresponding

size of the search space to be explored under that budget is also an interesting question: if we have a budget of 200 evaluations, how high can we take the dimensions and size of the search space ?

At the search space architectures level, the potential extensions are only limited by imagination and available computing resources. We have focused on only launching our search instances on well-designed search spaces with a reasonable expectation of finding better results than the baselines we started with. However, with more computing resources available, the design of the search space can be opened to much wider potential patterns. It is then possible to explore further away from established design elements like the efficient convolutions and self-attention mechanisms we presented in previous chapters.

The granularity of the search space can be boosted further, taking the basic components used in architecture design to a lower level. The idea is instead of using well-established blocks or well-known layers, the search space can be built by considering basic mathematical operators offered by the deep learning frameworks or even lower (Benmeziane et al., 2024). However, this type of search space can easily face exponential explosion and require very expensive resources, with the ever-present risk of not finding results worth the effort and cost. The environmental impacts are also a major concern, as well as the fact that improving the performance can be achieved in potentially easier ways: for instance, dataset quality has a big impact and might be a better strategy.

The focus of our efforts in this thesis were centered around efficient vision models. By design, the NAS strategy makes very few assumptions on the explored search space. Therefore, it can be deployed by other researchers and practitioners for their own usage in other areas. As a result, a potential future direction is to explore new architectures and designs in Natural Language Processing (NLP), audio processing, 3D point cloud processing, etc.

Finally, it is clear that the architecture is only partially responsible for the success or failure of deep model deployment. Many other design decisions factor in, from data selection and collection, to the training scheme and hyperparameters, to deployment considerations like quantization and compression. An important example is the usage of different numerical formats: the same model can be trained in single precision (FP32), half precision (FP16), or using a dynamic system like mixed precision which combines both. This accelerates training but has implications on the performance of the model. It stands to reason that the automatic exploration of this generalized search space is a more principled and robust way to find good results than incremental manual tuning. Additionally, these design decisions have high mutual interactions, and are better optimized jointly where possible.

With the impressive achievements of large language models (LLMs), an interesting idea might leverage these general-purpose text models for the automatic generation of search spaces starting from a reference architecture. As a result, it can be a viable way to fully automate the NAS process: the user can select a reference architecture from the literature which roughly corresponds to the size and specifications of the model they require, and then a search space

is generated to tune architectural parameters of the model and fit it for the specific dataset, task and hardware constraints of the considered use case.

A NAS method implementation details

Deep ensemble architecture

The deep ensemble is composed of 6 identical networks with the following architecture:

```
class EncoderModule(nn.Module):
    def __init__(self, n_objectives):
        super().__init__()
        self.shared = nn.Sequential(nn.Linear(6, 64),
                                    nn.LayerNorm(64),
                                    nn.SiLU(),
                                    nn.Linear(64, 128),
                                    nn.LayerNorm(128),
                                    nn.SiLU(),
                                    nn.Linear(128, 256),
                                    nn.LayerNorm(256),
                                    nn.SiLU(),
                                    nn.Linear(256, 256),
                                    nn.LayerNorm(256),
                                    nn.SiLU(),
                                    nn.Linear(256, 256),
                                    nn.LayerNorm(256),
                                    nn.SiLU())
        self.specialized_list = nn.ModuleList(
            [nn.Sequential(nn.Linear(256, 512),
                           nn.LayerNorm(512),
                           nn.SiLU())
             for _ in range(n_objectives)])
        self.embedding_dim = embedding_dim


def network_generator_func(n_objectives):
    return EncoderModule(n_objectives)
```

The variable `n_objectives` is set to `2`, to account for the multi-fidelity setup:

- the `self.shared` section is used for both levels of fidelity

- the `self.specialized_list` is specific to each level of fidelity

The same code is used in the mono-fidelity case, keeping one of the branches of the specialized section inactive.

Details and hyperparameters

These are the details of the pretraining and search procedures. Although performance is altered depending on hyperparameters, the trends shown in the paper are relatively stable. Most of the following hyperparameters were not specifically tuned, or tuned once at first for one scenario then kept at that value (like batch sizes).

Simultaneous pretraining	
Number of pretraining architectures	`2000`

Pretraining metrics	`FLOPs`
	`number of trainable parameters`
	`average latency`

Learning rate	`1e-2`
Batch size	`16`

Mono-fidelity experiments	
Evaluations per iteration	`32`
Pretraining epochs	`500`
Learning rate	`5e-3`
Batch size	`16`

Multi-fidelity experiments	
Full evaluations (200 epochs)	`3` full evals every `10` iterations
Partial evaluations (12 epochs)	`5` part evals every iteration
Pretraining epochs	`300`
Learning rate	`5e-3`
Batch size	`16`

Hardware used

All the benchmarks were run on an Macbook Pro (CPU of the M1 Pro chip), which we found to be faster in this instance than common GPUs. The code provided can be executed on a GPU or the M1's CPU. The network evaluations in section 4 were performed using 8x V100 GPUs, with each solution taking around 15m for a full evaluation, which puts the total time at around 20H.

Parallelization details (CPU)

For pretraining and training the ensemble on the CPU, we implemented a parallelization using the `joblib` library.

Every training epoch's internal steps is performed on the ensemble networks using a separate thread for each network.

```
from joblib import Parallel, delayed

def train_multiple_cpu(self, input_data, target_data, epochs, bs=16,
  obj_lst=None):
    input_data = input_data.to(self.accelerator)
    target_data = [e.to(self.accelerator) for e in target_data]
    train_set = TensorDataset(input_data, *target_data)
    train_loader = DataLoader(train_set, batch_size=bs, shuffle=True,
      num_workers=0)

    def train_epoch(module, optimizer):
        for batch_idx, batch in enumerate(train_loader):
            optimizer.zero_grad()
            preds = module(batch[0])
            loss = 0
            rng = range(1, len(batch)) if obj_lst is None else [1+_ for _
               in obj_lst]
            for i in rng:
                if batch[i] is None:
                    pass
                loss += nn.functional.huber_loss(input=preds[i - 1],
                  target=batch[i].view(preds[i - 1].shape))
            loss.backward()
            optimizer.step()
        return module, optimizer

    with Parallel(n_jobs=self.devices) as parallel:
        for _ in range(epochs):
            res = parallel(
                delayed(train_epoch)(
```

```
                    module,
                    optimizer
                )
                for module, optimizer in zip(self.modules, self.
                  optimizers)
            )
            self.modules, self.optimizers = [], []
            for (module, optimizer) in res:
                self.modules.append(module)
                self.optimizers.append(optimizer)
```

Bibliography

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

N. Adaloglou and F. Michels. Attention mechanism and transformers in deep learning. https://theaisummer.com/attention/, 2023. Accessed: 2024-08-17.

Apple. Introducing apple foundation models, 2024a. URL https://machinelearning.apple.com/research/introducing-apple-foundation-models. Accessed: 2024-08-16.

Apple. Neural engine transformers, 2024b. URL https://machinelearning.apple.com/research/neural-engine-transformers. Accessed: 2024-08-16.

F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro. Denser: deep evolutionary network structured representation. *Genetic Programming and Evolvable Machines*, 20:5–35, 2019.

M. Atzori, A. Gijsberts, S. Heynen, A.-G. M. Hager, O. Deriaz, P. Van Der Smagt, C. Castellini, B. Caputo, and H. Müller. Building the ninapro database: A resource for the biorobotics community. In *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 1258–1265. IEEE, 2012.

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

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

S. Balakrishnan, Q. P. Nguyen, B. K. H. Low, and H. Soh. Efficient exploration of reward functions in inverse reinforcement learning via bayesian optimization. *Advances in Neural Information Processing Systems*, 33:4187–4198, 2020.

M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33:21524–21538, 2020.

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

H. Benmeziane, K. E. Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang. A comprehensive survey on hardware-aware neural architecture search. *arXiv preprint arXiv:2101.09336*, 2021.

H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, and S. Niar. Grassroots operator search for model edge adaptation using mathematical search space. *Future Generation Computer Systems*, 157:29–40, 2024.

F. Berkenkamp, A. Krause, and A. P. Schoellig. Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics. *Machine Learning*, 112(10):3713–3747, 2023.

A. Biswas, Y. Liu, N. Creange, Y.-C. Liu, S. Jesse, J.-C. Yang, S. V. Kalinin, M. A. Ziatdinov, and R. K. Vasudevan. A dynamic bayesian optimized active recommender system for curiosity-driven partially human-in-the-loop automated experiments. *npj Computational Materials*, 10(1): 29, 2024.

J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

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

S. Cahon, N. Melab, and E.-G. Talbi. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of heuristics*, 10(3):357–380, 2004.

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

L. Cai, A.-M. Barneche, A. Herbout, C. S. Foo, J. Lin, V. R. Chandrasekhar, and M. M. S. Aly. Tea-dnn: the quest for time-energy-accuracy co-optimized deep neural networks. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2019.

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

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

Y. Chen, X. Dai, D. Chen, M. Liu, X. Dong, L. Yuan, and Z. Liu. Mobile-former: Bridging mobilenet and transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5270–5279, 2022.

F. Chollet et al. Keras, 2015. URL https://github.com/fchollet/keras.

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

X. Dai, A. Wan, P. Zhang, B. Wu, Z. He, Z. Wei, K. Chen, Y. Tian, M. Yu, P. Vajda, et al. Fbnetv3: Joint architecture-recipe search using predictor pretraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16276–16285, 2021a.

Z. Dai, A. Damianou, J. González, and N. Lawrence. Variational auto-encoded deep gaussian processes. *arXiv preprint arXiv:1511.06455*, 2015.

Z. Dai, H. Liu, Q. V. Le, and M. Tan. Coatnet: Marrying convolution and attention for all data sizes. *Advances in neural information processing systems*, 34:3965–3977, 2021b.

Dartmouth College. Artificial intelligence (ai) coined at dartmouth, 2024. URL https://home.dartmouth.edu/about/artificial-intelligence-ai-coined-dartmouth. Accessed: 2024-08-16.

DeepMind. Gemini nano, 2024. URL https://deepmind.google/technologies/gemini/nano/. Accessed: 2024-08-16.

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

J. Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

X. Ding, Y. Guo, G. Ding, and J. Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1911–1920, 2019.

X. Ding, X. Zhang, J. Han, and G. Ding. Diverse branch block: Building a convolution as an inception-like unit. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10886–10895, 2021a.

X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13733–13742, 2021b.

T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In *Proceedings of the European conference on computer vision (ECCV)*, pages 517–531, 2018.

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

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

X. Dong et al. Nats-bench: Benchmarking nas algorithms for architecture topology and size, 2020. URL https://github.com/D-X-Y/NATS-Bench. GitHub repository.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

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

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

N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

Falah.G.Salieh. Alzheimer mri dataset, 2023. URL https://huggingface.co/datasets/Falah/Alzheimer_MRI.

S. Falkner, A. Klein, and F. Hutter. Practical hyperparameter optimization for deep learning. 2018.

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

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28 (2015)*, pages 2962–2970, 2015.

M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv:2007.04074 [cs.LG]*, 2020.

P. I. Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

P. I. Frazier, W. B. Powell, and S. Dayanik. A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, 47(5):2410–2439, 2008.

A. F. Gad. Pygad: An intuitive genetic algorithm python library. *Multimedia Tools and Applications*, pages 1–14, 2023.

M. A. Ganaie, M. Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, 2022.

R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

R. Garnett, M. A. Osborne, and S. J. Roberts. Bayesian optimization for sensor set selection. In *Proceedings of the 9th ACM/IEEE international conference on information processing in sensor networks*, pages 209–219, 2010.

J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett. Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon technical report n*, 93:27403, 1993.

Gartner. What edge computing means for infrastructure and operations leaders, 2024. URL https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders. Accessed: 2024-08-16.

D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1487–1495, 2017.

Google. Introducing google tensor, 2024. URL https://blog.google/products/pixel/introducing-google-tensor/. Accessed: 2024-08-16.

Google Cloud. Neural architecture search overview - vertex ai. https://cloud.google.com/vertex-ai/docs/training/neural-architecture-search/overview, August 2024. Accessed: August 2024.

B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, and M. Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12259–12269, 2021.

S. Guo, J. M. Alvarez, and M. Salzmann. Expandnets: Linear over-parameterization to train compact convolutional networks. *Advances in Neural Information Processing Systems*, 33: 1298–1310, 2020a.

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

D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

A. Hagg, M. Zaefferer, J. Stork, and A. Gaier. Prediction of neural network performance by phenotypic modeling. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1576–1582, 2019.

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

K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016b.

X. He, K. Zhao, and X. Chu. Automl: A survey of the state-of-the-art. *Knowledge-based systems*, 212:106622, 2021.

P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(6), 2012.

M. Herdy et al. Evolutionary optimization based on subjective selection-evolving blends of coffee. In *Proc. 5th European Congress on Intelligent Techniques and Soft Computing*, pages 640–644, 1997.

J. M. Hernández-Lobato, M. Gelbart, M. Hoffman, R. Adams, and Z. Ghahramani. Predictive entropy search for bayesian optimization with unknown constraints. In *International conference on machine learning*, pages 1699–1707. PMLR, 2015.

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

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

J. Howard. Imagenette: A smaller subset of 10 easily classified classes from imagenet, March 2019a. URL https://github.com/fastai/imagenette.

J. Howard. Imagewoof: a subset of 10 classes from imagenet that aren't so easy to classify, March 2019b. URL https://github.com/fastai/imagenette#imagewoof.

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

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

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pages 507–523. Springer, 2011.

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

Intel Corporation. Moore's law, 2024. URL https://www.intel.com/content/www/us/en/newsroom/resources/moores-law.html. Accessed: 2024-08-16.

M. S. Iqbal, J. Su, L. Kotthoff, and P. Jamshidi. Flexibo: A decoupled cost-aware multi-objective optimization approach for deep neural networks. *Journal of Artificial Intelligence Research*, 77:645–682, 2023.

W. Jiang, L. Yang, E. H.-M. Sha, Q. Zhuge, S. Gu, S. Dasgupta, Y. Shi, and J. Hu. Hardware/software co-exploration of neural architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):4805–4815, 2020.

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

H. Jin, F. Chollet, Q. Song, and X. Hu. Autokeras: An automl library for deep learning. *Journal of Machine Learning Research*, 24(6):1–6, 2023. URL http://jmlr.org/papers/v24/20-1355.html.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13:455–492, 1998.

L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun. Hands-on bayesian neural networks—a tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 17 (2):29–48, 2022.

N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, et al. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

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

S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

M. Kim, Y. Ding, P. Malcolm, J. Speeckaert, C. J. Siviy, C. J. Walsh, and S. Kuindersma. Human-in-the-loop bayesian optimization of wearable device parameters. *PloS one*, 12(9):e0184054, 2017.

A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. Learning curve prediction with bayesian neural networks. In *International conference on learning representations*, 2022.

N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, A. Filippov, and E. Burnaev. Nas-bench-nlp: neural architecture search benchmark for natural language processing. *IEEE Access*, 10:45736–45747, 2022.

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

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

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

M. Lázaro-Gredilla, J. Quinonero-Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal. Sparse spectrum gaussian process regression. *The Journal of Machine Learning Research*, 11: 1865–1881, 2010.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Q. Lhoest, A. V. Del Moral, Y. Jernite, A. Thakur, P. Von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall, et al. Datasets: A community library for natural language processing. *arXiv preprint arXiv:2109.02846*, 2021.

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

C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, and Y. Lin. Hw-nas-bench: Hardware-aware neural architecture search benchmark. *arXiv preprint arXiv:2103.10584*, 2021b.

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

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

L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *ICLR (Poster)*, page 53, 2017.

L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-Tzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020a.

Y. Li, K. Zhang, J. Cao, R. Timofte, and L. Van Gool. Localvit: Bringing locality to vision transformers. *arXiv preprint arXiv:2104.05707*, 2021c.

Y. Li, G. Yuan, Y. Wen, J. Hu, G. Evangelidis, S. Tulyakov, Y. Wang, and J. Ren. Efficientformer: Vision transformers at mobilenet speed. *Advances in Neural Information Processing Systems*, 35:12934–12949, 2022.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020b.

M. Lindauer and F. Hutter. Best practices for scientific research on neural architecture search. *Journal of Machine Learning Research*, 21(243):1–18, 2020.

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

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

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

Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

D. J. Lizotte, T. Wang, M. H. Bowling, D. Schuurmans, et al. Automatic gait optimization with gaussian process regression. In *IJCAI*, volume 7, pages 944–949, 2007.

M. Loni, S. Sinaei, A. Zoljodi, M. Daneshtalab, and M. Sjödin. Deepmaker: A multi-objective optimization framework for deep neural networks in embedded systems. *Microprocessors and Microsystems*, 73:102989, 2020.

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

J. Lukasik, S. Jung, and M. Keuper. Learning where to look–generative nas is surprisingly efficient. In *European Conference on Computer Vision*, pages 257–273. Springer, 2022.

R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu. Neural architecture optimization. *Advances in neural information processing systems*, 31, 2018.

L. Ma, J. Cui, and B. Yang. Deep neural architecture search with deep graph bayesian optimization. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 500–507, 2019.

N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

M. Maaz, A. Shaker, H. Cholakkal, S. Khan, S. W. Zamir, R. M. Anwer, and F. Shahbaz Khan. Edgenext: efficiently amalgamated cnn-transformer architecture for mobile vision applications. In *European conference on computer vision*, pages 3–20. Springer, 2022.

A. Maraval, M. Zimmer, A. Grosnit, R. Tutunov, J. Wang, and H. B. Ammar. Sample-efficient optimisation with probabilistic transformer surrogates. *arXiv preprint arXiv:2205.13902*, 2022.

R. Marchant and F. Ramos. Bayesian optimisation for intelligent environmental monitoring. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 2242–2249. IEEE, 2012.

A. Marchisio, A. Massa, V. Mrazek, B. Bussolino, M. Martina, and M. Shafique. Nascaps: A framework for neural architecture search to optimize the accuracy and hardware efficiency of convolutional capsule networks. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

A. Marchisio, V. Mrazek, A. Massa, B. Bussolino, M. Martina, and M. Shafique. Rohnas: A neural architecture search framework with conjoint optimization for adversarial robustness and hardware efficiency of convolutional and capsule networks. *IEEE Access*, 10:109043–109055, 2022.

R. Martinez-Cantin. Bayesian optimization with adaptive kernels for robot control. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3350–3356. IEEE, 2017.

R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and systems*, volume 3, pages 321–328, 2007.

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

S. Mehta and M. Rastegari. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

S. Mehta and M. Rastegari. Separable self-attention for mobile vision transformers. *arXiv preprint arXiv:2206.02680*, 2022.

J. Mellor, J. Turner, A. Storkey, and E. J. Crowley. Neural architecture search without training. In *International conference on machine learning*, pages 7588–7598. PMLR, 2021.

J. H. Metzen, R. Hutmacher, N. G. Hua, V. Boreiko, and D. Zhang. Identification of systematic errors of image classifiers on rare subgroups. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5064–5073, 2023.

Y. Miao, X. Song, J. D. Co-Reyes, D. Peng, S. Yue, E. Brevdo, and A. Faust. Differentiable architecture search for reinforcement learning. In *International Conference on Automated Machine Learning*, pages 20–1. PMLR, 2022.

Microsoft. Neural Network Intelligence, 1 2021. URL https://github.com/microsoft/nni.

R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*, pages 269–287. Elsevier, 2024.

J. Mockus. On bayesian methods for seeking the extremum. In *Proceedings of the IFIP Technical Conference*, pages 400–404, 1974.

P. Morere, R. Marchant, and F. Ramos. Sequential bayesian optimization as a pomdp for environment monitoring with uavs. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6381–6388. IEEE, 2017.

Y. Morita, S. Rezaeiravesh, N. Tabatabaei, R. Vinuesa, K. Fukagata, and P. Schlatter. Applying bayesian optimization with gaussian process regression to computational fluid dynamics problems. *Journal of Computational Physics*, 449:110788, 2022.

J. P. Muñoz, N. Lyalyushkin, Y. Akhauri, A. Senina, A. Kozlov, and N. Jain. Enabling nas with automated super-network generation. *arXiv preprint arXiv:2112.10878*, 2021.

J. Pan, A. Bulat, F. Tan, X. Zhu, L. Dudziak, H. Li, G. Tzimiropoulos, and B. Martinez. Edgevits: Competing light-weight cnns on mobile devices with vision transformers. In *European Conference on Computer Vision*, pages 294–311. Springer, 2022.

S. Park, J. Na, M. Kim, and J. M. Lee. Multi-objective bayesian optimization of chemical reactor design using computational fluid dynamics. *Computers & Chemical Engineering*, 119:25–37, 2018.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

D. Peng, X. Dong, E. Real, M. Tan, Y. Lu, G. Bender, H. Liu, A. Kraft, C. Liang, and Q. Le. Pyglove: Symbolic programming for automated machine learning. *Advances in Neural Information Processing Systems*, 33:96–108, 2020.

H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

W. Rawat and Z. Wang. Hybrid stochastic ga-bayesian search for deep convolutional neural network model selection. *J. Univers. Comput. Sci.*, 25(6):647–666, 2019.

E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *International conference on machine learning*, pages 2902–2911. PMLR, 2017.

E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Aging evolution for image classifier architecture search. In *AAAI conference on artificial intelligence*, volume 2, page 2, 2019a.

E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019b.

S. Ren and A. Wierman. The uneven distribution of ai's environmental impacts. *Harvard Business Review*, 2024. URL https://hbr.org/2024/07/the-uneven-distribution-of-ais-environmental-impacts.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

B. Ru, X. Wan, X. Dong, and M. Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. *arXiv preprint arXiv:2006.07556*, 2020.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

S. Sajedi and X. Liang. Deep generative bayesian optimization for sensor placement in structural health monitoring. *Computer-Aided Civil and Infrastructure Engineering*, 37(9):1109–1127, 2022.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

K. A. Sankararaman, S. De, Z. Xu, W. R. Huang, and T. Goldstein. The impact of neural network overparameterization on gradient confusion and stochastic gradient descent. In *International conference on machine learning*, pages 8469–8479. PMLR, 2020.

Secondmind Labs. Deep ensembles for bayesian optimization, 2023. URL https://secondmind-labs.github.io/trieste/1.0.0/notebooks/deep_ensembles.html. Accessed: August 2024.

M. W. Seeger, C. K. Williams, and N. D. Lawrence. Fast forward selection to speed up sparse gaussian process regression. In *International Workshop on Artificial Intelligence and Statistics*, pages 254–261. PMLR, 2003.

K. Seshadri, B. Akin, J. Laudon, R. Narayanaswami, and A. Yazdanbakhsh. An evaluation of edge tpu accelerators for convolutional neural networks. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*, pages 79–91. IEEE, 2022.

D. Sha, K. Ozbay, and Y. Ding. Applying bayesian optimization for calibration of transportation simulation models. *Transportation Research Record*, 2674(10):215–228, 2020.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

A. Shaker, M. Maaz, H. Rasheed, S. Khan, M.-H. Yang, and F. S. Khan. Swiftformer: Efficient additive attention for transformer-based real-time mobile vision applications. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17425–17436, 2023.

H. Shi, R. Pi, H. Xu, Z. Li, J. Kwok, and T. Zhang. Bridging the gap between sample-based and one-shot neural architecture search with bonas. *Advances in Neural Information Processing Systems*, 33:1808–1819, 2020.

C. A. Shoemaker, R. G. Regis, and R. C. Fleming. Watershed calibration using multistart local optimization and evolutionary optimization with radial basis function approximation. *Hydrological sciences journal*, 52(3):450–465, 2007.

K. Simonyan. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18, 2005.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR, 2015.

D. So, Q. Le, and C. Liang. The evolved transformer. In *International conference on machine learning*, pages 5877–5886. PMLR, 2019.

N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

E.-G. Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

E.-G. Talbi. Automated design of deep neural networks: A survey and unified taxonomy. *ACM Computing Surveys (CSUR)*, 54(2):1–37, 2021.

C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A survey on deep transfer learning. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27*, pages 270–279. Springer, 2018.

M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

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

H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

H. Tsai, J. Ooi, C.-S. Ferng, H. W. Chung, and J. Riesa. Finding fast transformers: One-shot neural architecture search by component composition. *arXiv preprint arXiv:2008.06808*, 2020.

R. Tu, N. Roberts, M. Khodak, J. Shen, F. Sala, and A. Talwalkar. Nas-bench-360: Benchmarking neural architecture search on diverse tasks. *Advances in Neural Information Processing Systems*, 35:12380–12394, 2022a.

Z. Tu, H. Talebi, H. Zhang, F. Yang, P. Milanfar, A. Bovik, and Y. Li. Maxvit: Multi-axis vision transformer. In *European conference on computer vision*, pages 459–479. Springer, 2022b.

P. K. A. Vasu, J. Gabriel, J. Zhu, O. Tuzel, and A. Ranjan. Fastvit: A fast hybrid vision transformer using structural reparameterization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5785–5795, 2023a.

P. K. A. Vasu, J. Gabriel, J. Zhu, O. Tuzel, and A. Ranjan. Mobileone: An improved one millisecond mobile backbone. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7907–7917, 2023b.

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

J. Vendrow, S. Jain, L. Engstrom, and A. Madry. Dataset interfaces: Diagnosing model failures using controllable counterfactual generation. *arXiv preprint arXiv:2302.07865*, 2023.

Z. Wang, G. E. Dahl, K. Swersky, C. Lee, Z. Nado, J. Gilmer, J. Snoek, and Z. Ghahramani. Pre-trained gaussian processes for bayesian optimization. *Journal of Machine Learning Research*, 25(212):1–83, 2024.

C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, and J. Liang. Npenas: Neural predictor guided evolution for neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8441–8455, 2022.

T. Wei, C. Wang, Y. Rui, and C. W. Chen. Network morphism. In *International conference on machine learning*, pages 564–572. PMLR, 2016.

C. White, W. Neiswanger, S. Nolen, and Y. Savani. A study on encodings for neural architecture search. *Advances in neural information processing systems*, 33:20309–20319, 2020.

C. White, W. Neiswanger, and Y. Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10293–10301, 2021a.

C. White, S. Nolen, and Y. Savani. Exploring the loss landscape in neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 654–664. PMLR, 2021b.

C. White, M. Khodak, R. Tu, S. Shah, S. Bubeck, and D. Dey. A deeper look at zero-cost proxies for lightweight nas. In *ICLR Blog Track*, 2022. URL https://iclr-blog-track.github.io/2022/03/25/zero-cost-proxies/. https://iclr-blog-track.github.io/2022/03/25/zero-cost-proxies/.

O. Wiles, I. Albuquerque, and S. Gowal. Discovering bugs in vision models using off-the-shelf image generation and captioning. *arXiv preprint arXiv:2208.08831*, 2022.

A. Wilson, A. Fern, and P. Tadepalli. Using trajectory data to improve bayesian optimization for reinforcement learning. *The Journal of Machine Learning Research*, 15(1):253–282, 2014.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10734–10742, 2019.

H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 22–31, 2021.

J. Wu and P. Frazier. The parallel knowledge gradient method for batch bayesian optimization. *Advances in neural information processing systems*, 29, 2016.

S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

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

S. Yan, Y. Zheng, W. Ao, X. Zeng, and M. Zhang. Does unsupervised architecture representation learning help neural architecture search? *Advances in neural information processing systems*, 33:12486–12498, 2020.

C. Yang, S. Qiao, Q. Yu, X. Yuan, Y. Zhu, A. Yuille, H. Adam, and L.-C. Chen. Moat: Alternating mobile convolution and attention brings strong vision models. In *The Eleventh International Conference on Learning Representations*, 2022.

Y. Yang, A. Nam, M. Nasr-Azadani, and T. Tung. Resource-aware pareto-optimal automated machine learning platform. In *2020 3rd international seminar on research of information technology and intelligent systems (ISRITI)*, pages 1–6. IEEE, 2020.

C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International conference on machine learning*, pages 7105–7114. PMLR, 2019.

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

T. Yu and H. Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.

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

A. Zela, J. Siems, and F. Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. *arXiv preprint arXiv:2001.10422*, 2020a.

A. Zela, J. Siems, L. Zimmer, J. Lukasik, M. Keuper, and F. Hutter. Surrogate nas benchmarks: Going beyond the limited search spaces of tabular nas benchmarks. *arXiv preprint arXiv:2008.09777*, 2020b.

C. Zhang, M. Ren, and R. Urtasun. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018.

J. Zhang, P. Fiers, K. A. Witte, R. W. Jackson, K. L. Poggensee, C. G. Atkeson, and S. H. Collins. Human-in-the-loop optimization of exoskeleton assistance during walking. *Science*, 356 (6344):1280–1284, 2017.

Y. Zhou, Y. Koyama, M. Goto, and T. Igarashi. Generative melody composition with human-in-the-loop bayesian optimization. *arXiv preprint arXiv:2010.03190*, 2020.

Y. Zhou, X. Dong, B. Akin, M. Tan, D. Peng, T. Meng, A. Yazdanbakhsh, D. Huang, R. Narayanaswami, and J. Laudon. Rethinking co-design of neural architectures and hardware accelerators. *arXiv preprint arXiv:2102.08619*, 2021.

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

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