Neural Architecture Search: Insights from 1000 Papers

Colin White COLIN@ABACUS.AI
Abacus.AI
San Francisco, CA 94105, USA

Mahmoud Safari SAFARIM@CS.UNI-FREIBURG.DE
University of Freiburg
Freiburg im Breisgau, 79110, Germany

Rhea Sukthanker SUKTHANK@CS.UNI-FREIBURG.DE
University of Freiburg
Freiburg im Breisgau, 79110, Germany

Binxin Ru ROBINRU@SAILYOND.COM
Sailyond Technology & Research Institute of Tsinghua University
Shenzhen, 518071, China

Thomas Elsken THOMAS.ELSKEN@DE.BOSCH.COM
Bosch Center for Artificial Intelligence
Renningen, 71272, Germany

Arber Zela ZELAA@CS.UNI-FREIBURG.DE
University of Freiburg
Freiburg im Breisgau, 79110, Germany

Debadeepta Dey DEDEY@MICROSOFT.COM
Microsoft Research
Redmond, WA 98052, USA

Frank Hutter FH@CS.UNI-FREIBURG.DE
University of Freiburg & Bosch Center for Artificial Intelligence
Freiburg im Breisgau, 79110, Germany

Abstract

In the past decade, advances in deep learning have resulted in breakthroughs in a variety of areas, including computer vision, natural language understanding, speech recognition, and reinforcement learning. Specialized, high-performing neural architectures are crucial to the success of deep learning in these areas. Neural architecture search (NAS), the process of automating the design of neural architectures for a given task, is an inevitable next step in automating machine learning and has already outpaced the best human-designed architectures on many tasks. In the past few years, research in NAS has been progressing rapidly, with over 1000 papers released since 2020 (Deng and Lindauer, 2021). In this survey, we provide an organized and comprehensive guide to neural architecture search. We give a taxonomy of search spaces, algorithms, and speedup techniques, and we discuss resources such as benchmarks, best practices, other surveys, and open-source libraries.

Keywords: neural architecture search, automated machine learning, deep learning

1. Introduction

In the past decade, deep learning has become the dominant paradigm in machine learning for a variety of applications and has been used in a number of breakthroughs across computer vision (He et al., 2016a; Huang et al., 2017; Krizhevsky et al., 2012; Szegedy et al., 2017), natural language understanding (Bahdanau et al., 2015; Hochreiter and Schmidhuber, 1997; Vaswani et al., 2017), speech recognition (Chan et al., 2016; Chorowski et al., 2015; Hannun et al., 2014), and reinforcement learning (Mnih et al., 2015; Silver et al., 2016); it is also becoming a very powerful approach for the analysis of tabular data (Hollmann et al., 2022; Kadra et al., 2021; Somepalli et al., 2021). While many factors played into the rise of deep learning approaches, including deep learning's ability to automate feature extraction, as well as an increase in data and the larger availability of computational resources, the design of high-performing neural architectures has been crucial to the success of deep learning. Recently, just as manual feature engineering was replaced by automated feature learning via deep learning, it is getting more and more common to automate the time-consuming architecture design step via *neural architecture search*. Neural architecture search (NAS), the process of automating the design of neural architectures for a given task, has already outpaced the best human-designed architectures on many tasks (Chen et al., 2018; Du et al., 2020; Ghiasi et al., 2019; So et al., 2019; Zoph et al., 2018), notably ImageNet (Hu et al., 2019; Liu et al., 2018a; Real et al., 2019; Zoph et al., 2018), as well as diverse and less-studied datasets (Shen et al., 2022), and in memory- or latency-constrained settings (Benmeziane et al., 2021). Indeed, in the past few years, research in NAS has been progressing rapidly. Although several surveys have been written for NAS and related areas in the past (Elsken et al., 2019b; Wistuba et al., 2019, also see Section 10.2), **over 1000 new NAS papers have been released in the last two years** (Deng and Lindauer, 2021), warranting the need for a new survey on over-arching advances, which we aim to provide with this work.

1.1 A Brief History of NAS and Relation to Other Fields

NAS emerged as a subfield of *automated machine learning (AutoML)* (Hutter et al., 2019), the process of automating all steps in the machine learning pipeline, from data cleaning, to feature engineering and selection, to hyperparameter and architecture search. NAS has a large overlap with *hyperparameter optimization (HPO)* (Feurer and Hutter, 2019), which refers to the automated optimization of hyperparameters of the machine learning model. NAS is sometimes referred to as a subset of HPO (Li and Talwalkar, 2019), since NAS can be expressed as optimizing only the hyperparameters that correspond to the architecture, a subset of the entire set of model hyperparameters. However, the techniques for HPO vs. NAS are often substantially different.

A typical HPO problem optimizes a mix of continuous and categorical hyperparameters, such as learning rate, dropout rate, batch size, momentum, activation function, normalization strategy, and so on. Typically, the domains of most hyperparameters are independent (that is, the set of possible values for each hyperparameter is not affected by the possible values of other hyperparameters). Therefore, the typical *search space* of an HPO problem is the product space of a mix of continuous and categorical dimensions. By contrast, NAS is specifically focused on optimizing the *topology of the architecture*, which can be much more complex. The topology is typically represented by a directed acyclic graph (DAG), in which the nodes or edges are labeled by neural network operations. Therefore, the *search space* of a NAS problem is typically discrete[1] and can be represented directly as a graph, or as a hierarchical structure of conditional hyperparameters.

Although standard HPO algorithms can sometimes be adapted for NAS (Izquierdo et al., 2021; Klein et al., 2020; Li et al., 2020c; Mendoza et al., 2016; Zela et al., 2018; Zimmer et al., 2021), it is often much more efficient and effective to use NAS techniques which are tailored to optimize the intricate space of neural architectures. Furthermore, most modern NAS techniques go beyond black-box optimization algorithms by exploiting details specific to NAS, such as sharing weights among similar neural architectures to avoid training each of them from scratch.

Historically, NAS has been around since at least the late 1980s (Angeline et al., 1994; Kitano, 1990; Miller et al., 1989; Tenorio and Lee, 1988) but it did not gain widespread attention until the popular paper, *NAS with Reinforcement Learning*, by Zoph and Le (2017). There has since been a huge interest in NAS, with over 1000 papers released in the last two years (see Figure 1).

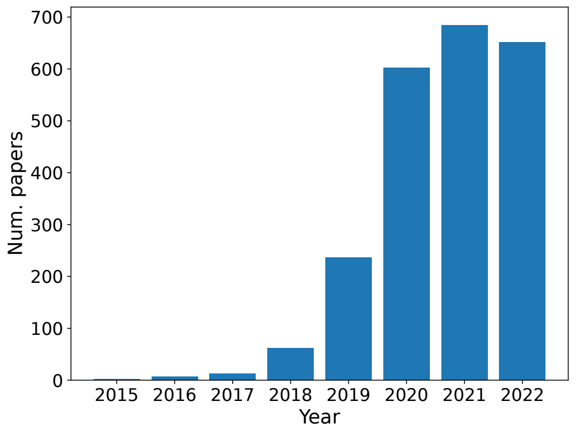

Figure 1: Number of NAS papers by year (Deng and Lindauer, 2021).

By now, many different approaches, such as reinforcement learning, evolutionary algorithms, Bayesian optimization, and NAS-specific techniques based on weight sharing have been explored. Perhaps the most popular recent approaches are one-shot techniques (Bender et al., 2018; Liu et al., 2019c), which often substantially speed up the search process compared to black-box optimization techniques. In recent years, a large body of follow-up work has focused on making one-shot methods more robust and reliable (Wang et al., 2021; Zela et al., 2020a). In parallel, there has been a large push to make NAS research more reproducible and scientific, starting with the release of NAS-Bench-101 (Ying et al., 2019), the first tabular benchmark for NAS. Furthermore, while the early days of NAS has mostly focused on image classification problems such as CIFAR-10 and ImageNet, the field has now expanded to many other domains, such as object detection (Ghiasi et al., 2019; Xu et al., 2019a), semantic segmentation (Chen et al., 2018; Liu et al., 2019a), speech recognition (Mehrotra et al., 2021), partial differential equation solving (Roberts et al., 2021; Shen et al., 2022; Tu et al., 2022a), protein folding (Roberts et al., 2021; Shen et al., 2022), and weather prediction (Tu et al., 2022b), and the field has seen a renewed interest in natural language processing (Chitty-Venkata et al., 2022; Javaheripi et al., 2022).

1.2 Background and Definitions

Prior NAS surveys (e.g. Elsken et al., 2019b; Wistuba et al., 2019) have referred to three dimensions of NAS: *search space, search strategy*, and *performance evaluation strategy* (see

1. Notably, some NAS techniques such as DARTS (Liu et al., 2019c) relax the domain to be continuous during the search, but then the hyperparameters are discretized in order to return the final architecture.

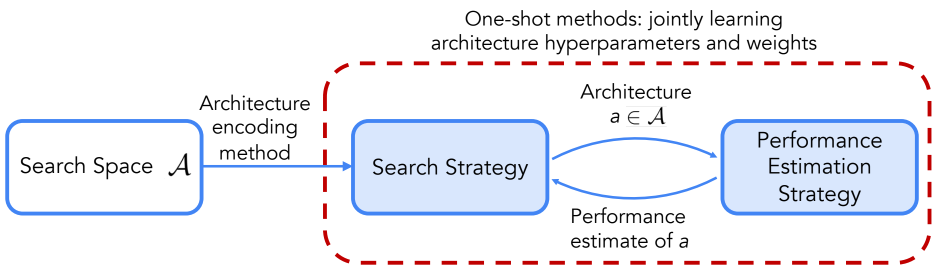
One-shot methods: jointly learning architecture hyperparameters and weights
Search Space $\mathcal{A}$
Architecture encoding method
Search Strategy
Architecture $a \in \mathcal{A}$
Performance Estimation Strategy
Performance estimate of a

Figure 2: Overview of neural architecture search (Elsken et al., 2019b; Weng, 2020). A search strategy iteratively selects architectures (typically by using an architecture encoding method) from a predefined search space $\mathcal{A}$. The architectures are passed to a performance estimation strategy, which returns the performance estimate to the search strategy. For *one-shot methods*, the search strategy and performance estimation strategy are inherently coupled.

Figure 2). We define each term below, as this is a useful disambiguation for understanding many NAS methods. However, it is worth noting that the trichotomy cannot be applied to the large sub-area of one-shot methods, because for these methods, the search strategy is coupled with the performance evaluation strategy (Xie et al., 2021).

A *search space* is the set of all architectures that the NAS algorithm is allowed to select. Common NAS search spaces range in size from a few thousand to over 10^{20}. While the search space in principle can be extremely general, incorporating domain knowledge when designing the search space can simplify the search. However, adding too much domain knowledge introduces human bias, which reduces the chances of a NAS method finding truly novel architectures. Search spaces are discussed in more detail in Section 2.

A *search strategy* is an optimization technique used to find a high-performing architecture in the search space. There are generally two main categories of search strategies: black-box optimization based techniques (including multi-fidelity techniques) and one-shot techniques. However, there are some NAS methods for which both or neither category applies. Black-box optimization based techniques, such as reinforcement learning, Bayesian optimization, and evolutionary search, are surveyed in Section 3. One-shot methods, including supernet- and hypernet-based methods, are surveyed in Section 4.

A *performance estimation strategy* is any method used to quickly predict the performance of neural architectures in order to avoid fully training the architecture. For example, while we can run a discrete search strategy by fully training and evaluating architectures chosen throughout the search, using a performance estimation strategy such as learning curve extrapolation can greatly increase the speed of the search. Performance estimation strategies, and more generally speedup techniques, are surveyed in Section 5.

The most basic definition of NAS is as follows. Given a search space $\mathcal{A}$, a dataset $\mathcal{D}$, a training pipeline $\mathcal{P}$, and a time or computation budget t, the goal is to find an architecture $a \in \mathcal{A}$ within budget t which has the highest possible validation accuracy when trained using dataset $\mathcal{D}$ and training pipeline $\mathcal{P}$. A common method of approaching NAS is to approximately solve the following expression within time t:

$$\min_{a \in \mathcal{A}} \quad \mathcal{L}_{\text{val}}\left(w^*(a), a\right) \quad \text{s.t.} \quad w^*(a) = \text{argmin}_w \, \mathcal{L}_{\text{train}}\left(w, a\right).$$

Here, $\mathcal{L}_{\text{val}}$ and $\mathcal{L}_{\text{train}}$ denote the validation loss and training loss, respectively. While this is the core definition of NAS, other variants will be discussed throughout this survey. For example, we may want to return an architecture with constraints on the number of parameters (Section 6.2), or we may use meta-learning (Section 5.3) to improve performance.

Throughout the rest of this article, we provide a comprehensive guide to the latest NAS techniques and resources. Sections 2 to 5 are devoted to NAS techniques, surveying search spaces, black-box optimization techniques, one-shot techniques, and speedup techniques, respectively. Sections 6 to 10 cover extensions, applications, and resources, and Section 11 concludes by discussing promising future directions.

2. Search Spaces

The search space is perhaps the most essential ingredient of NAS. While other areas of AutoML overlap with NAS in terms of the optimization methods used, the architectural search space is unique to NAS. Furthermore, the search space is often the first step when setting up NAS. The majority of popular search spaces are task-specific and were heavily inspired by the state-of-the-art manual architectures in their respective application domains. For example, NAS-Bench-101, a popular image classification search space (Ying et al., 2019) was inspired by ResNet (He et al., 2016a) and Inception (Szegedy et al., 2017).

In fact, the design of the search space represents an important trade-off between human bias and efficiency of search: if the size of the search space is small and includes many hand-picked decisions, then NAS algorithms will have an easier time finding a high-performing architecture. On the other hand, if the search space is large with more primitive building blocks, a NAS algorithm will need to run longer, but there is the possibility of discovering truly novel architectures (Real et al., 2020).

In this section, we survey the main categories of search spaces for NAS as summarized in Table 1. We start in Section 2.1 by defining general terminology. In Sections 2.2 and 2.3, we discuss the relatively simple macro and chain-structured search spaces, respectively. In Section 2.4, we describe the most popular type of search space: the cell-based search space. In Section 2.5, we describe hierarchical search spaces. Finally, in Section 2.6, we discuss architecture encodings, an important design decision for NAS algorithms that is inherently tied to the choice of search space.

2.1 Terminology

The search space terminologies differ across the literature, depending on the type of search space. For clarity, we define the main terms here and in Appendix Figure 9.

- *Operation/primitive* denotes the atomic unit of the search space. For nearly all popular search spaces, this is a triplet of a fixed activation, operation, and fixed normalization, such as `ReLU-conv_1x1-batchnorm`, where the `ReLU` and BatchNorm are fixed, and the middle operation is a choice among several different operations.

Search Spaces	Structure	Searchable hyperparameters	Levels of Topology
Macro search space e.g. NASBOT (Kandasamy et al., 2018), EfficientNet (Tan and Le, 2019)	DAG	Operation types, DAG topology, macro hyperparameters	1
Chain-structured search space e.g. MobileNetV2 (Sandler et al., 2018)	Chain	Operation types, macro hyperparameters	1
Cell-based search space e.g. DARTS (Liu et al., 2019c)	Duplicated cells	Operation type, cell topology	1
Hierarchical search space e.g. Hier. Repr. (Liu et al., 2018b), Auto-DeepLab (Liu et al., 2019b)	Varied	Operation type, cell/DAG topology, macro hyperparameters	> 1

Table 1: Summary of the types of NAS search spaces.

- *Layer* is often used in chain-structured or macro search spaces to denote the same thing as an operation or primitive. However, it sometimes refers to well-known combinations of operations, such as the `inverted bottleneck residual` (Cai et al., 2019; Sandler et al., 2018; Tan and Le, 2019; Tan et al., 2019).
- *Block/Module* is sometimes used to denote a sequential stack of layers following the notation used in most chain-structured and macro search spaces (Cai et al., 2020; Tan and Le, 2019; Tan et al., 2019).
- *Cell* is used to denote a directed acyclic graph of operations in cell-based search spaces. The maximum number of operations in a cell is often fixed.
- *Motif* is used to denote a sub-pattern formed from multiple operations in an architecture. Some literature refers to a cell as a higher-level motif and a smaller set of operations as a base-level motif.

2.2 Macro Search Spaces

In the NAS literature, macro search spaces may refer to one of two types. First, they may refer to search spaces which encode the entire architecture in one level (as opposed to cell-based or hierarchical search spaces), which were popular in 2017 and 2018. Second, they may refer to search spaces which focus only on macro-level hyperparameters.

For the former, an entire architecture is represented as a single directed acyclic graph (Baker et al., 2017; Kandasamy et al., 2018; Real et al., 2017; Zoph and Le, 2017). These search spaces typically have a choice of operation at each node in the graph, as well as the choice of DAG topology. For example, the NASBOT CNN search space (Kandasamy et al., 2018) consists of choices of different convolution, pooling, and fully connected layers, with any DAG topology, with depth of at most 25.

The second type of macro search spaces (Dong et al., 2021b; Duan et al., 2021; Tan and Le, 2019), focus on the variation of macro-level hyperparameters, such as where and how much to downsample the spatial resolution throughout the architecture, while keeping the architecture topology and operations fixed.[2] For example, Tan and Le (2019) propose a CNN search space by varying the network depth, width, and input feature resolution.

Compared to other search spaces, macro search spaces have high representation power: their flexible structure allows the possibility of discovering novel architectures. However, their main downside is that they are very slow to search. In the next two sections, we discuss types of search spaces which have more rigidity, making them faster to search.

2.3 Chain-Structured Search Spaces

Chain-structured search spaces, as the name suggests, have a simple architecture topology: a sequential chain of operation layers. They often take state-of-the-art manual designs, such as ResNet (He et al., 2016b) or MobileNets (Howard et al., 2017), as the backbone.

There are several chain-structured search spaces based on convolutional networks. ProxylessNAS (Cai et al., 2019) starts with the MobileNetV2 (Sandler et al., 2018) architecture and searches over the kernel sizes and expansion ratios in the inverted bottleneck residual layers. XD (Roberts et al., 2021) and DASH (Shen et al., 2022) start with a LeNet (LeCun et al., 1999), ResNet (He et al., 2016a), or WideResNet (Zagoruyko and Komodakis, 2016), and search over an expressive generalization of convolutions based on Kaleidoscope matrices (Dao et al., 2020), or kernel sizes and dilations, respectively.

Chain-structured search spaces are also popular in transformer-based search spaces. For example, the search space from Lightweight Transformer Search (LTS) (Javaheripi et al., 2022) consists of a chain-structured configuration of the popular GPT family of architectures (Brown et al., 2020; Radford et al., 2019) for autoregressive language modeling, with searchable choices for the number of layers, model dimension, adaptive embedding dimension, dimension of the feedforward neural network in a transformer layer, and number of heads in each transformer layer. The search spaces from NAS-BERT (Xu et al., 2021a) and MAGIC (Xu et al., 2022) both consist of a chain-structured search space over the BERT architecture (Devlin et al., 2019) with up to 26 operation choices consisting of variants of multi-head attention, feedforward layers, and convolutions with different kernel sizes.

Chain-structured search spaces are conceptually simple, making them easy to design and implement. They also often contain strong architectures that can be found relatively quickly. Their main downside is that, due to the simple architecture topology, there is a comparatively lower chance of discovering a truly novel architecture.

2.4 Cell-based Search Spaces

The cell-based search space is perhaps the most popular type of search space in NAS. It is inspired by the fact that state-of-the-art human-designed CNNs often consist of repeated patterns, for example, residual blocks in ResNets (Zoph et al., 2018). Thus, instead of searching for the entire network architecture from scratch, Zoph et al. (2018) proposed to only search over relatively small *cells*, and stack the cells several times in sequence to form the overall architecture. Formally, the searchable cells make up the *micro* structure of the search space, while the outer skeleton (the *macro* structure) is fixed.

2. Strictly speaking, since these search spaces have a fixed architecture topology, they may also be called hyperparameter tuning search spaces instead of NAS search spaces.

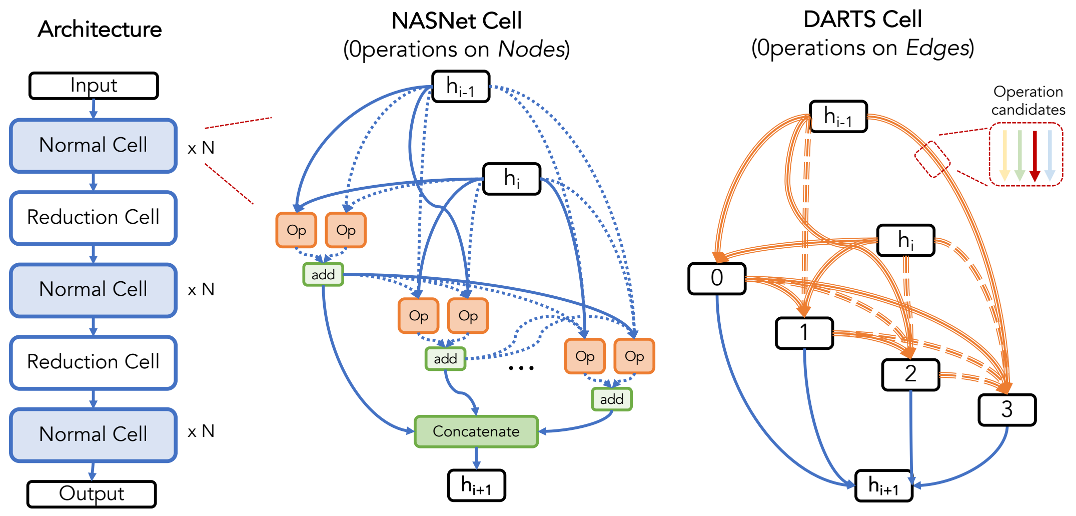

Figure 3: Illustration of cell-based search spaces. The outer skeleton across cells (left) is fixed, while the cells are searchable. NASNet assigns operations to nodes (middle) while DARTS assigns operations to edges (right).

The first modern cell-based search space, NASNet, was proposed by Zoph et al. (2018). It comprises of two types of cells: the *normal* cell and the *reduction* cell. Both types have the same structure, but the initial operations in the reduction cell have a stride of two to halve the input spatial resolution. Each NASNet cell can be represented as a DAG with seventeen non-input nodes (see Figure 3 (middle)). The nodes are arranged in triples of two operation nodes (such as convolution and pooling operations) and a combination node (such as addition or concatenation). The final NASNet architecture is formed by stacking multiple normal and reduction cells in sequence (see Figure 3 (left)). Overall, there are 10^{35} unique architectures in the NASNet search space.

Since the NASNet search space, many other cell search spaces have been proposed, all of which share a high-level similarity to NASNet, with the main differences being the fixed macro structure, the layout and constraints in the cells, and the choices of operations within the cells. Two of the most popular cell-based search spaces are NAS-Bench-101 (Ying et al., 2019) and the DARTS search space (Liu et al., 2019c). NAS-Bench-101 is the first tabular benchmark for NAS (discussed in Section 8), and its cells consist of seven nodes, each with three choices of operations; it contains 423 624 unique architectures. The DARTS search space differs more fundamentally: while it also has two searchable cells, the DARTS cells have operation choices on the *edges* of the graph rather than on the nodes. In the DARTS cell, the nodes represent latent representations and the edges are operations, whereas in the NASNet cell, the latent representations are on the edges and the nodes are operations. The DARTS cells (see Figure 3 (right)) contain eight edges, each of which have eight choices of operations. Overall, the DARTS space contains a total of 10^{18} unique architectures.

Besides image classification, similar cell designs have also been adopted for language models. For example, NAS-Bench-ASR (Mehrotra et al., 2021) provides a search space of convolutional speech model cells for automatic speech recognition, and there are several LSTM-based search spaces (Klyuchnikov et al., 2022; Liu et al., 2019c; Pham et al., 2018).

The cell-based design significantly reduces the complexity of search spaces, while often resulting in a high-performing final architecture. This has led to the cell-based search spaces being the most popular type of search space in recent years. Furthermore, by detaching the depth of an architecture from the search, the cell-based structure is transferable: the optimal cells learned on a small dataset (e.g., CIFAR-10) typically transfer well to a large dataset (e.g., ImageNet) by increasing the number of cells and filters in the overall architecture (Liu et al., 2019c; Zoph et al., 2018).

Despite their popularity, cell-based search spaces face some criticisms. First, while the DARTS search space contains a seemingly large number of 10^{18} architectures, the variance in the performance of DARTS architectures is rather small (Wan et al., 2022b; Yang et al., 2020). This small variance may contribute to the fact that sophisticated search strategies can only give marginal gains over the average performance of randomly sampled architectures (Yang et al., 2020). Moreover, there are many ad-hoc design choices and fixed hyperparameters that come with cell-based search spaces whose impact is unclear (Wan et al., 2022b), such as the separation of normal and reduction cells, number of nodes, and set of operations. Finally, although limiting the search to a cell significantly reduces the search complexity, this practice reduces the expressiveness of the NAS search space, making it difficult to find highly novel architectures with cell search spaces. In light of this, some recent work advocates for searching for macro connections among cells in addition to the micro cell structure. We discuss this in more detail in the next section.

2.5 Hierarchical Search Spaces

Up to this point, all search spaces described have had a *flat* representation, in which an architecture is built by defining its hyperparameters, topology, and operation primitives in a single design level. Specifically, only one level of topology is searched, whether at the cell level or architecture level. On the other hand, *hierarchical* search spaces involve designing motifs at different levels, where each higher-level motif is often represented as a DAG of lower-level motifs (Chrostoforidis et al., 2021; Liu et al., 2018b; Ru et al., 2020b).

A simple class of hierarchical search spaces has two searchable levels by adding macro-level architecture hyperparameters to cell or chain-structured search spaces. For example, the MnasNet search space (Tan et al., 2019) uses MobileNetV2 as the backbone. Liu et al. (2019b) designed a two-level search space for semantic image segmentation, and follow-up work extended it to image denoising (Zhang et al., 2020a) and stereo matching (Kumari and Kaur, 2016). Finally, Chen et al. (2021a) propose a two-level transformer-based search space for vision tasks inspired by ViT (Dosovitskiy et al., 2021) and DeiT (Touvron et al., 2021). The search space consists of a number of sequential blocks which can be a combination of local (convolution) or global (self-attention) layers.

Beyond two levels, Liu et al. (2018b) and Wu et al. (2021) propose hierarchies of three levels. Liu et al. (2018b) propose a three-level hierararchy, where each level is a graph made up of components from the previous level (see Figure 4). Wu et al. (2021) propose a different three-level hierarchy, consisting of kernel hyperparameters, cell-based hyperparameters, and macro hyperparameters. The former design is extended beyond three levels in two follow-up works: Ru et al. (2020b) proposed a hierarchical design of four levels, controlled by a set of hyperparameters corresponding to a random graph generator, and Chrostoforidis et al. (2021) introduced a recursive building process to permit a varying number of hierarchical levels as well as a flexible topology among top-level motifs.

There are multiple benefits to using hierarchical search spaces. First, hierarchical search spaces tend to be more expressive. Most chain-structured, cell-based, and macro search spaces can be seen as a hierarchical search space with a single searchable level, but having two or more levels allows us to search over more diverse and complex architecture designs. Furthermore, a hierarchical representation of a large architecture is an effective way to reduce the search complexity, which can lead to better search efficiency (Chrostoforidis et al., 2021; Liu et al., 2018b; Ru et al., 2020b). On the other hand, hierarchical search spaces can be more challenging to implement and search through.

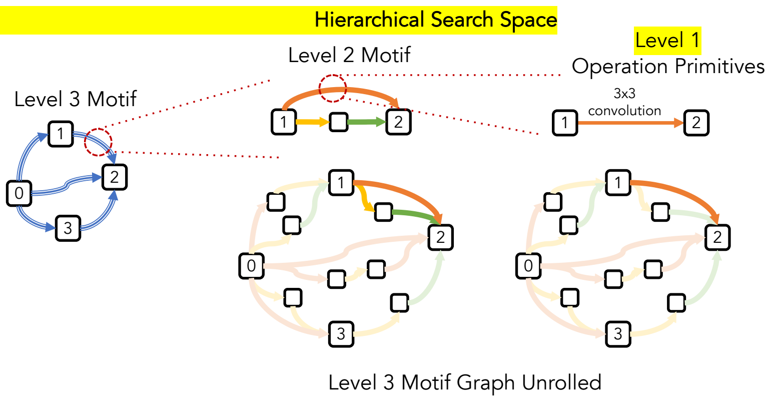

Figure 4: Illustration of hierarchical representation proposed in Liu et al. (2018b). Level 1 of the hierarchy consists of choices of operation primitives. Level 2 consists of selecting the topology across small sets of operation primitives. Level 3 consists of selecting the topology across the constructions from level 2.

2.6 Architecture Encodings

Throughout this section, we have discussed a wide variety of NAS search spaces. As a segue into the next two sections focusing on search strategies, we note that many NAS algorithms and subroutines need to have a succinct representation of each architecture, or *encoding*, in order to perform operations such as mutating an architecture, quantifying the similarity between two architectures, or predicting the test performance of an architecture. This makes architecture encodings important for several areas of NAS, including discrete NAS algorithms (Section 3) and performance prediction (Section 5.1).

In most search spaces, the architecture can be represented compactly as a directed acyclic graph (DAG), where each node or edge represents an operation. For example, architectures in cell-based search spaces and chain-structured search spaces can be represented in this way. However, hierarchical search spaces cannot be represented fully using a DAG, and often need a conditionally-structured encoding, where the number of levels of conditional hyperparameters correspond to the number of levels of the hierarchy.

For cell-based search spaces, one of the most commonly-used encodings is the adjacency matrix along with a list of operations, of the searchable cell(s) (Ying et al., 2019; Zoph and Le, 2017). In order to have better generalizablility, Ning et al. (2020) proposed a graph-based encoding scheme and White et al. (2021a) proposed a path-based encoding scheme, both of which model the flow of propagating information in the network. Finally, another type of encoding for all search spaces is a *learned* encoding using unsupervised pre-training. In this technique, before we run NAS, we use a set of untrained architectures to learn an architecture encoding, for example, by using an autoencoder (Li et al., 2020b; Lukasik et al., 2021, 2022; Yan et al., 2020; Zhang et al., 2019) or a transformer (Yan et al., 2021a).

When choosing an architecture encoding, scalability and generalizability are important traits. Recent work has shown that different NAS subroutines, such as sampling a random architecture, perturbing an architecture, or training a surrogate model, may each perform best with different encodings (White et al., 2020). Furthermore, even small changes to the architecture encoding scheme can have significant effects on the performance of NAS (White et al., 2020; Ying et al., 2019).

3. Black-Box Optimization Techniques

Now that we have covered search spaces, we move to perhaps the most widely-studied component of NAS: the *search strategy*. This is what we run to find an optimal architecture from the search space. Search strategies generally fall into two categories: black-box optimization techniques and one-shot techniques. However, some methods that we discuss include characteristics of both, or neither, of these categories. We first discuss black-box optimization techniques in this section, followed by one-shot techniques in Section 4.

For black-box optimization, we discuss baselines (Section 3.1), reinforcement learning (Section 3.2), evolution (Section 3.3), Bayesian optimization (Section 3.4), and Monte-Carlo tree search (Section 3.5). Black-box optimization techniques are widely used and studied today, due to their strong performance and ease of use. In general, black-box optimization techniques tend to use more computational resources than one-shot techniques, due to training many architectures independently (without sharing weights across architectures like one-shot techniques). However, they also have many advantages over one-shot techniques, such as robustness (and the lack of catastrophic failure modes), simpler optimization of non-differentiable objectives, simpler parallelism, joint optimization with other hyperparameters, and easier adaptation to, e.g., new problems, datasets or search spaces. They are also often conceptually simpler, making them easier to implement and use.

3.1 Baselines

One of the simplest possible baselines for NAS is *random search*: architectures are selected randomly from the search space and then fully trained. In the end, the architecture with the best validation accuracy is outputted. Despite its naïveté, multiple papers have shown that random search performs surprisingly well (Chen et al., 2018; Li and Talwalkar, 2019; Sciuto et al., 2020; Yang et al., 2020). This is especially true for highly engineered search spaces with a high fraction of strong architectures, since random search with a budget of k evaluations will, in expectation, find architectures in the top $100/k\%$ of the search space. However, other works show that random search does not perform well on large,

Algorithm 1 General Reinforcement Learning NAS Algorithm

```
**Input:** Search space $\mathcal{A}$, number of iterations $T$.
Randomly initialize weights $\theta$ of the controller architecture.
for $t = 1, \ldots, T$ do
    Train architecture $a \sim \pi(a; \theta)$, randomly sampled from the controller policy $\pi(a; \theta)$.
    Update controller parameters $\theta$ by performing a gradient update $\nabla_\theta E_{a \sim \pi(a;\theta)}[\mathcal{L}_{\text{val}}(a)]$.
end for
**Output:** Architecture selected from the trained policy $\pi(a; \theta^*)$
```

diverse search spaces (Bender et al., 2020; Real et al., 2020). Still, random search is highly recommended as a baseline comparison for new NAS algorithms (Lindauer and Hutter, 2020; Yang et al., 2020), and can be made highly competitive by incorporating weight sharing (Li and Talwalkar, 2019), zero-cost proxies (Abdelfattah et al., 2021), or learning curve extrapolation (Yan et al., 2021b). Multiple papers (Sciuto et al., 2020; Yang et al., 2020) have also proposed a related, simpler baseline: *random sampling*, the average performance of architectures across the entire search space.

In addition to random search, recent papers showed that local search is a strong baseline for NAS on both small (Ottelander et al., 2021; White et al., 2021b) and large (Siems et al., 2020) search spaces. This is true even for the simplest form of local search: iteratively train and evaluate all of the neighbors of the best architecture found so far, where the neighborhood is typically defined as all architectures which differ by one operation or edge. Local search can be sped up substantially by using network morphisms to warm-start the optimization of neighboring architectures (Elsken et al., 2017).

3.2 Reinforcement Learning

Reinforcement learning (RL) was very prominent in the early days of modern NAS. Notably, the seminal work by Zoph and Le (2017) used RL on 800 GPUs for two weeks to obtain competitive performance on CIFAR-10 and Penn Treebank; this finding received substantial media attention and started the modern resurgence of NAS. This was followed up by several more reinforcement learning approaches (Pham et al., 2018; Zoph et al., 2018).

Most reinforcement learning approaches model the architectures as a sequence of actions generated by a controller (Baker et al., 2017; Zoph and Le, 2017). The validation accuracy of the sampled architectures after training is used as a reward signal to update the controller in order to maximize its expected value. See Algorithm 1. The controller is usually a recurrent neural network (RNN) (Zoph and Le, 2017; Zoph et al., 2018) that outputs a sequence of components corresponding to an architecture. After each outputted architecture is trained and evaluated, the RNN parameters are updated to maximize the expected validation accuracy of outputted architectures, using REINFORCE (Williams, 1992; Zoph and Le, 2017) or proximal policy optimization (Schulman et al., 2017; Zoph et al., 2018). ENAS (Pham et al., 2018) follows a similar strategy but speeds up the reward estimation using weight sharing; we will discuss this in detail in Section 4.

More recently, RL has not been used prominently for NAS, since it has been shown to be outperformed in head-to-head comparisons by evolutionary methods (Real et al., 2019) and Bayesian optimization (Ying et al., 2019), which we will discuss next.

Algorithm 2 General Evolutionary NAS Algorithm

```
Input: Search space $\mathcal{A}$, number of iterations $T$.
Randomly sample and train a population of architectures from the search space $\mathcal{A}$.
for $t = 1, \ldots, T$ do
    Sample (based on accuracy) a set of parent architectures from the population.
    Mutate the parent architectures to generate children architectures, and train them.
    Add the children to the population, and kill off the architectures that are the oldest
    (or have the lowest accuracy) among the current population.
end for
Output: Architecture from the population with the highest validation accuracy.
```

3.3 Evolutionary and Genetic Algorithms

Decades before the recent NAS resurgence, one of the first works in NAS used an evolutionary algorithm (Miller et al., 1989). In other early works, it was common to use evolutionary algorithms to simultaneously optimize the neural architecture and its weights (Angeline et al., 1994; Floreano et al., 2008; Stanley and Miikkulainen, 2002; Stanley et al., 2009). Today, evolutionary algorithms are still popular for the optimization of architectures due to their flexibility, conceptual simplicity, and competitive results (Real et al., 2019), but the weight optimization is typically left to standard SGD-based approaches.

Evolutionary NAS algorithms work by iteratively updating a population of architectures. In each step, one or more "parent" architectures in the population are sampled (typically based on the validation accuracy of the architectures), combined and mutated to create new "children" architectures. These architectures are then trained and added to the population, replacing individuals in the population with worse performance. See Algorithm 2.

There are many other ways in which evolutionary algorithms differ, including sampling the initial population, selecting the parents, and generating the children. For selecting the initial population, approaches include using trivial architectures (Real et al., 2017), randomly sampling architectures from the search space (Real et al., 2019; Sun et al., 2019), or using hand-picked high-performing architectures (Fujino et al., 2017).

Selecting parents from the population makes up one of the core components of the evolutionary algorithm. Perhaps the most popular method to sample parents is tournament selection (Almalaq and Zhang, 2018; Goldberg and Deb, 1991; Real et al., 2017, 2019; Sun et al., 2019, 2020), which selects the best architecture(s) out of a randomly sampled population. Other common approaches include random sampling weighted by fitness (Gibb et al., 2018; Loni et al., 2020; Song et al., 2020; Xie and Yuille, 2017), or choosing the current best architecture(s) as parents (Elsken et al., 2017; Suganuma et al., 2017, 2018). These methods trade off exploration vs. exploiting the best region found so far. One particularly successful evolutionary algorithm is *regularized evolution* by Real et al. (2019). This is a fairly standard evolutionary method, with the novelty of dropping the architecture in each step that has been in the population for longest, even if it has the highest performance. This method outperformed random search and RL in a head-to-head comparison and achieved state-of-the-art performance on ImageNet at the time of its release (Real et al., 2019).

Algorithm 3 General Bayesian Optimization NAS Algorithm

```
Input: Search space $\mathcal{A}$, number of iterations $T$, acquisition function $\phi$.
Randomly sample and train a population of architectures from the search space $\mathcal{A}$.
for $t = 1, \ldots, T$ do
    Train a surrogate model based on the current population.
    Select architecture $a_t$ by maximizing $\phi(a)$, based on the surrogate model.
    Train architecture $a_t$ and add it to the current population.
end for
Output: Architecture from the population with the highest validation accuracy.
```

3.4 Bayesian Optimization

Bayesian optimization (BO, see, e.g. Frazier (2018) or Garnett (2023)) is a powerful method for optimizing expensive functions, and it has seen significant success within NAS. There are two key components to BO: *(1)* building a probabilistic surrogate to model the unknown objective based on past observations, and *(2)* defining an acquisition function to balance the exploration and exploitation during the search. BO is an iterative algorithm which works by selecting the architecture that maximizes the acquisition function (computed using the surrogate), training this architecture, and retraining the surrogate using this new architecture to start the next iteration. See Algorithm 3.

Initial BO-based NAS techniques developed custom distance metrics among architectures, for example, with a specialized architecture kernel (Swersky et al., 2014), an optimal transport-inspired distance function (Kandasamy et al., 2018), or a tree-Wasserstein distance function (Nguyen et al., 2021), allowing a typical Gaussian process (GP) based surrogate with BO. However, using a standard GP surrogate often does not perform well for NAS, as search spaces are typically high-dimensional, non-continuous, and graph-like. To overcome this, one line of work first encodes the architectures, using encodings discussed in Section 2.6, and then trains a model, such as a tree-Parzen estimator (Bergstra et al., 2011; Falkner et al., 2018), random forest (Hutter et al., 2011; Ying et al., 2019), or neural network (Springenberg et al., 2016; White et al., 2021a). Another line of work projects architecture information into a low-dimensional continuous latent space on which conventional BO can be applied effectively (Ru et al., 2020b; Wan et al., 2022a). Another class of surrogate models use graph neural networks (Ma et al., 2019; Ru et al., 2021; Shi et al., 2020) or a graph-based kernel (Ru et al., 2021) to naturally handle the graph representation of architectures without the need for an explicit encoding.

The acquisition function, which trades off exploration and exploitation during the search, is another important design component for BO. There are various types of acquisition functions used in NAS, such as expected improvement (Jones et al., 1998; Močkus, 1975), upper confidence bound (Cox and John, 1992; Srinivas et al., 2010) and information-theoretic ones (Hennig and Schuler, 2012; Hernández-Lobato et al., 2014; Hvarfner et al., 2022; Wang and Jegelka, 2017). In NAS, optimizing the acquisition function in each round of BO is challenging due to the non-continuous search spaces, and furthermore, exhaustively evaluating acquisition function values on all possible architectures is computationally non-viable. The most common method for optimizing the acquisition function in NAS is by randomly mutating a small pool of the best architectures queried so far, and of the mutated architectures, selecting the one(s) with the highest acquisition function value (Kandasamy et al., 2018; Ma et al., 2019; Ru et al., 2021; Schneider et al., 2021; Shi et al., 2020; White et al., 2021a). Other methods for optimizing the acqusition function include local search, evolutionary search, and random search (Ru et al., 2021; Shi et al., 2020; Ying et al., 2019).

3.5 Monte Carlo Tree Search

Another class of NAS methods is based on Monte Carlo Tree Search (MCTS). MCTS is the key backbone search algorithm used in AlphaGO (Silver et al., 2016) and AlphaZero (Silver et al., 2017), which achieve super-human performance in Go and chess, respectively. MCTS finds optimal decisions by recursively sampling new decisions (e.g., making a move in chess, or selecting an operation for an architecture in NAS), running stochastic *rollouts* to obtain the reward (such as winning a chess game, or discovering a high-performing architecture) and then backpropagating to update the weight of the initial decision. Across iterations, the algorithm builds a decision tree to bias the search towards more promising regions by balancing exploration and exploitation in decision making (Browne et al., 2012).

MCTS was first applied to NAS by Negrinho and Gordon (2017) who represented the search space and its hyperparameters using a modular language. This results in a tree-structured, extensible search space, contrary to the fixed search spaces of prior work. Wistuba (2018) introduced a similar method but with two different UCT (Upper Confidence bounds applied to Trees) algorithms. MCTS was first adapted to cell-based search spaces by using a state-action representation (Wang et al., 2018). The authors also improved sample efficiency by using a neural network to estimate the accuracy of sampled architectures, thus enabling a higher number of rollouts. This was followed up by adding further efficiency in pruning the tree by learning partitionings (Wang et al., 2020b), and by application to multi-objective NAS (Zhao et al., 2021a).

4. One-Shot Techniques

Throughout Section 3, we have seen that the predominant methodology in the early stages of NAS research was to iteratively sample architectures from the search space, train them, and use their performance to guide the search. The main drawback of these methods, when applied without speedup techniques, is their immense computational cost, sometimes on the order of thousands of GPU days (Real et al., 2019; Zoph and Le, 2017) due to the need to train thousands of architectures *independently* and *from scratch*.[3]

As an alternative, *one-shot* techniques were introduced to avoid training each architecture from scratch, thus circumventing the associated computational burden. As of 2022, they are currently one of the most popular techniques in NAS research. Rather than training each architecture from scratch, one-shot approaches implicitly train all architectures in the search space via a single ("one-shot") training of a *hypernetwork* or *supernetwork*.

A *hypernetwork* is a neural network which generates the weights of *other* neural networks (Schmidhuber, 1992), while a *supernetwork* (often used synonymously with "one-shot

3. On the other hand, recent developments in performance estimation and speed-up techniques (Section 5) have significantly improved the computational overhead of methods that use black-box optimization as a base, making these methods affordable for many applications and users.

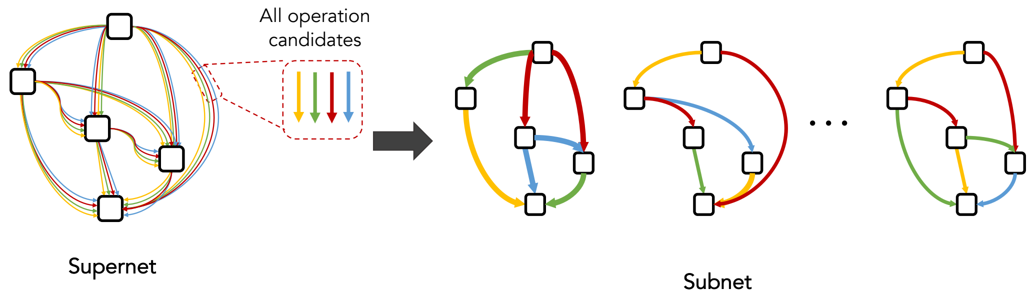

Figure 5: A supernet comprises all possible architectures in the search space. Each architecture is a subnetwork (subgraph) in the supernet.

model" in the literature) is an over-parameterized architecture that contains all possible architectures in the search space as subnetworks (see Figure 5). The idea of a supernetwork was introduced by Saxena and Verbeek (2016) and was popularized in 2018 by works such as Bender et al. (2018), Pham et al. (2018), and Liu et al. (2019c).

Once a supernet is trained, each architecture from the search space can be evaluated by inheriting its weights from the corresponding subnet within the supernet. The reason for the scalability and efficiency of supernets is that a linear increase in the number of candidate operations only causes a linear increase in computational costs for training, but the number of subnets in the supernet increases exponentially. Therefore, supernets allow us to train an exponential number of architectures for a linear compute cost.

A key assumption made in one-shot approaches is that when using the one-shot model to evaluate architectures, the ranking of architectures is relatively consistent with the ranking one would obtain from training them independently. The extent to which this assumption holds true has been substantially debated, with work showing evidence for (Li et al., 2021c; Pham et al., 2018; Yu et al., 2020) and against (Pourchot et al., 2020; Sciuto et al., 2020; Zela et al., 2020b; Zhang et al., 2020b) the claim across various settings. The validity of the assumption is dependent on the search space design, the techniques used to train the one-shot model, and the dataset itself, and it is hard to predict to what degree the assumption will hold in a particular case (Sciuto et al., 2020; Zhang et al., 2020b).

While the supernet allows quick evaluation of all architectures, we must still decide on a search strategy, which can be as simple as running a black-box optimization algorithm while the supernet is training (such as in Pham et al. (2018)) or after the supernet is trained (such as in Bender et al. (2018)). We discuss these families of techniques in Section 4.1. A popular line of work uses gradient descent to optimize the architecture hyperparameters in tandem with training the supernet (such as DARTS (Liu et al., 2019c) and numerous subsequent methods). We discuss this family of techniques in Section 4.2. Finally, in Section 4.3, we discuss hypernetworks. Figure 6 provides a taxonomy of one-shot families.

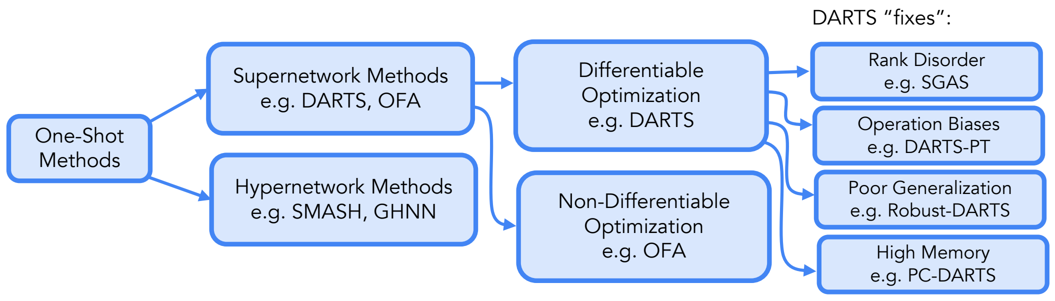
DARTS “fixes”:
One-Shot Methods
Supernetwork Methods e.g. DARTS, OFA
Hypernetwork Methods e.g. SMASH, GHNN
Differentiable Optimization e.g. DARTS
Non-Differentiable Optimization e.g. OFA
Rank Disorder e.g. SGAS
Operation Biases e.g. DARTS-PT
Poor Generalization e.g. Robust-DARTS
High Memory e.g. PC-DARTS

Figure 6: A taxonomy of the predominant one-shot families. A *hypernetwork* is a neural net which generates the weights of other neural nets. A *supernetwork* is an over-parameterized neural net that contains the set of neural nets from the search space as subnetworks, and it can be used with differentiable optimization (including DARTS and follow-ups), or non-differentiable optimization.

4.1 Non-Differentiable Supernet-Based Methods

We start by describing supernet-based methods which do not make use of differentiable optimization. Some methods in this family decouple the supernet training and architecture search: first train a supernet, and then run a black-box optimization algorithm to search for the best architecture. Other methods train a supernet while simultaneously running a non-differentiable search algorithm, such as reinforcement learning, to select subnetworks.

Bender et al. (2018), Li and Talwalkar (2019), and Guo et al. (2020b) propose simple methods to train the supernet and then use a black-box optimization algorithm to extract the best architecture from it. Bender et al. (2018) construct the supernet by creating a separate node corresponding to an operation, in every place where there is a choice of operation; they then train the supernet as if it were a standard neural net, with one exception: nodes are randomly dropped during training, with the level of dropout increasing linearly throughout training. In follow-up work, Li and Talwalkar (2019) and Guo et al. (2020b) take this idea a step further: in each training step, they randomly sample one architecture and only update the weights of the supernet corresponding to that architecture. These techniques better mimic what is happening at evaluation time: only a subnetwork is evaluated rather than the entire supernet. Furthermore, these procedures use significantly less memory than training all the weights of a supernet. Each method concludes by using the trained supernet to quickly evaluate architectures when conducting random search (Bender et al., 2018; Li and Talwalkar, 2019) or evolutionary search (Guo et al., 2020b). The architecture identified in the end is then trained from scratch.

As will be discussed in Section 6.2, deploying neural nets in practice often comes with constraints on latency or memory. While the supernets considered thus far tend to only contain architectures of approximately the same size, Cai et al. (2020) propose a supernet containing subnetworks of various sizes. This *Once-for-all (OFA)* approach uses a progressive shrinking strategy which starts by sampling the largest subnetworks, and then moving

Algorithm 4 DARTS - Differentiable Architecture Search

```
Input: Search space $\mathcal{A}$, number of iterations $T$, hyperparameter $\xi$.
Randomly initialize a one-shot model based on $\mathcal{A}$ with weights $w$ and architecture hyperparameters $\alpha$.
for $t = 1, \ldots, T$ do
    Perform a gradient update on the architecture weights $\alpha$ according to Equation 1.
    Perform a gradient update on $w$ according to $\nabla_w \mathcal{L}_{train}(w, \alpha)$.
end for
Output: Derive the final architecture by taking the argmax of $\alpha$, across all operation choices, and then retrain this architecture from scratch.
```

to smaller subnetworks, in order to minimize the co-adaptation among subnetworks and effectively train networks of different sizes "once for all". In a subsequent *search phase*, architectures are selected based on different constraints on latency and memory. While Cai et al. (2020) uses random search for this search phase, Guo et al. (2020b) proposed to improve this approach further by using evolutionary search in the search phase.

One of the earliest supernet-based approaches is ENAS (Efficient Neural Architecture Search) (Pham et al., 2018), which trains the supernet while running a search algorithm in tandem. Specifically, the search strategy is similar to the RL controller-based approach from Zoph and Le (2017) (described in Section 3.2) but estimates the performance of each architecture using a supernet. The training procedure alternates between selecting an architecture, evaluating it, and updating the weights of the supernet, and updating the weights of the controller by sampling several architectures to estimate the reward of REINFORCE. While this approach searches for an architecture in tandem with training the supernet, it uses a separate controller network to guide the search. In the next section, we discuss methods which conduct the search via gradient descent using only the supernet.

4.2 Differentiable Supernet-Based Methods

In this section, we review supernet-based NAS methods that employ differentiable optimization techniques. We first describe the seminal DARTS (Differentiable Architecture Search) approach by Liu et al. (2019c), and then we move to various follow-up works and other differentiable approaches.

The DARTS approach uses a continuous relaxation of the discrete architecture search space, which enables the use of gradient descent in order to find a high-performing local optimum significantly faster than black-box optimization methods. It can be applied to any DAG-based search space which has different choices of operations on each edge by using a "zero" operation to simulate the absence of an edge.

At the start, each edge (i, j) in the DARTS search space consists of multiple possible candidate operations o, each of which are associated with a continuous hyperparameter $\alpha_o^{(i,j)} \in [0, 1]$. While the supernet is training, edge (i, j) consists of a mix of all candidate operations, weighted by each $\alpha_o^{(i,j)}$. The architecture hyperparameters α are optimized jointly with the supernet model weights w via alternating gradient descent. In particular, in order to update the architecture weights α via gradient descent, DARTS makes use of

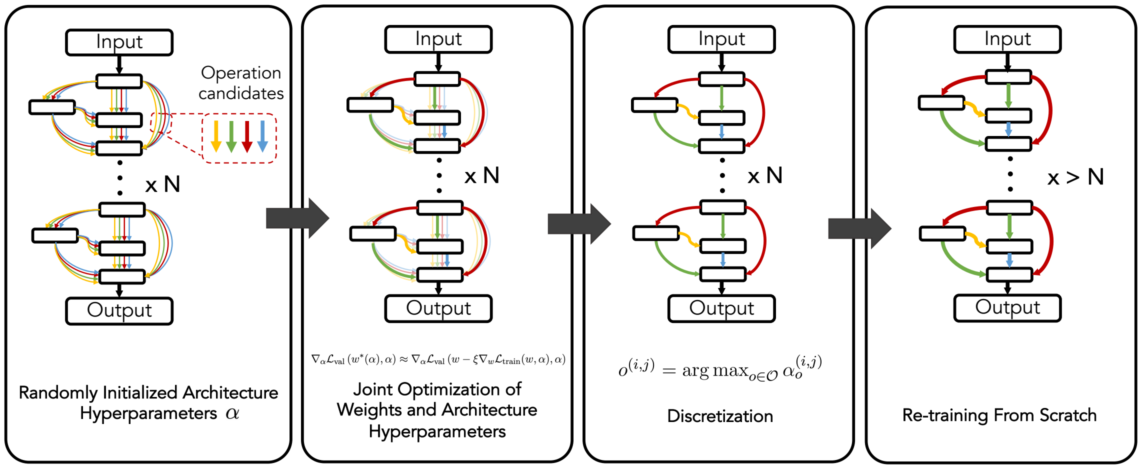
Input
Operation candidates
x N
Output
Randomly Initialized Architecture Hyperparameters α
Input
x N
Output
$\nabla_\alpha \mathcal{L}_{\text{val}}\left(w^*(\alpha), \alpha\right) \approx \nabla_\alpha \mathcal{L}_{\text{val}}\left(w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha), \alpha\right)$
Joint Optimization of Weights and Architecture Hyperparameters
Input
x N
Output
$o^{(i,j)} = \arg\max_{o \in \mathcal{O}} \alpha_o^{(i,j)}$
Discretization
Input
x > N
Output
Re-training From Scratch

Figure 7: Differentiable one-shot NAS algorithms have four main steps: randomly initializing the architecture hyperparameters, optimizing the architecture hyperparameters and weights via alternating gradient descent, discretizing the optimized architecture hyperparameters, and re-training the resulting subnetwork from scratch.

the following approximation:

$$\nabla_\alpha \mathcal{L}_{\text{val}}\left(w^*(\alpha), \alpha\right) \approx \nabla_\alpha \mathcal{L}_{\text{val}}\left(w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha), \alpha\right), \qquad (1)$$

where $\mathcal{L}_{\text{train}}$ denotes the training loss, $\mathcal{L}_{\text{val}}$ denotes the validation loss, ξ is the learning rate, and $w^*(\alpha)$ denotes the weights that minimize the training loss of the architecture corresponding to α. In other words, in order to avoid the expensive inner optimization, $w^*(\alpha)$ is approximated by a single step of gradient descent ($w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha)$). This is similar to MAML (Finn et al., 2017) and other works (Luketina et al., 2016; Metz et al., 2017). Although this strategy is not guaranteed to converge, Liu et al. (2019c) showed that it works well in practice with a suitable choice of ξ. After the training phase, DARTS obtains a discrete architecture by selecting the operation with the maximum value of α on each edge (the discretization step) and then re-trains it from scratch. Figure 7 provides an illustration of DARTS.

DARTS gained significant attention in the AutoML community due to its simplicity, its novelty, and the release of easy-to-use code. Furthermore, the original technique left room for improvement across various axes. Consequently, there has been a large body of follow-up work seeking to improve various parts of the DARTS approach. In the rest of the section, we cover the main categories of improvements (see Figure 6).

4.2.1 Rank Disorder

As mentioned at the start of Section 4, nearly all one-shot methods make a key assumption: the ranking of architectures evaluated with the supernet is relatively consistent with the ranking one would obtain from training them independently; when this assumption is not met, it is known as *rank disorder* (Li et al., 2021c; Sciuto et al., 2020). While there is considerable debate both for (Li et al., 2021c; Pham et al., 2018; Yu et al., 2020) and against (Pourchot et al., 2020; Sciuto et al., 2020; Zela et al., 2020b; Zhang et al., 2020b) the assumption, many works have attempted to reduce the problem of rank disorder.

Several methods propose to gradually increase the network depth, or to gradually prune the set of operation candidates during training, showing that this causes the weights to better adapt to the most-promising operation choices. Progressive-DARTS (Chen et al., 2019a) gradually increases the network depth while simultaneously pruning the operations with the smallest weights. SGAS (Li et al., 2020a) chooses operations throughout the training procedure, based on two criteria: selection certainty (calculated via the entropy of the operation distribution) and selection stability (calculated via the movement of the operation distribution). Finally, XNAS (Nayman et al., 2019) makes use of the exponentiated gradient algorithm (Kivinen and Warmuth, 1997), which dynamically prunes inferior operation choices during the search while also allowing the recovery of "late bloomers", i.e., operation choices which only become accurate later in the training procedure.

4.2.2 Operation Biases

Several works show that differentiable NAS techniques tend to favor skip connections over other operation choices (Liang et al., 2019; Wang et al., 2021; Zela et al., 2020a), which might be caused by the supernet using skip connections to over-compensate for vanishing gradients (Chu et al., 2021). Various methods have been proposed to fix this bias.

DARTS+ (Liang et al., 2019) proposes an early stopping method based on the stability of the ranking of the architecture weights, while DARTS− (Chu et al., 2021) separates the skip connection weights from other operation weights via auxiliary edges. FairDARTS (Chu et al., 2020) sets *all* operation weights independent of all others, and then pushes these architecture weights toward zero or one in the loss function.

Taking a different approach, Wang et al. (2021) show that it is okay for skip connections to have higher weights, as long as we do not select the final architecture based on these weights. Instead, after training the supernet, their algorithm, DARTS-PT, selects each operation whose removal has the largest decrease of accuracy in the supernet.

Rather than fixing the biases among a small hand-picked set of operations, Shen et al. (2022) instead use a search space that significantly reduces human bias: they fix a standard convolutional network and search for the kernel sizes and dilations of its operations. This simple approach is broadly applicable across computer vision, PDE solving, protein folding, and other tasks. In order to make one-shot training more efficient, their algorithm, DASH, computes the mixture-of-operations using the Fourier diagonalization of convolution.

4.2.3 Poor Test Generalization

Several works seek to improve the generalization performance of DARTS through various means. Zela et al. (2020a) and Chen and Hsieh (2020) show that DARTS often converges to sharp local minima in the loss landscape (high validation loss curvature in the architecture hyperparameter space), which, after running the discretization step, can cause the algorithm to return an architecture with poor test generalization. Robust-DARTS (Zela et al., 2020a) fixes this issue by making the training more robust through data augmentation, L_2

regularization of the inner objective $\mathcal{L}_{train}$, and early stopping. Similarly, rather than optimizing the training loss, Smooth-DARTS (Chen and Hsieh, 2020) optimizes the expected or worst-case training loss over a local neighborhood of the architecture hyperparameters.

Taking a different approach, GAEA (Li et al., 2021c), XD (Roberts et al., 2021), and StacNAS (Guilin et al., 2019) all use a single-level optimization rather than the typical bi-level optimization, by treating the architecture hyperparameters as normal architecture weights, showing this leads to better generalization. Furthermore, GAEA re-parameterizes the architecture parameters over the simplex and updates them using the exponentiated gradient algorithm (similar to XNAS from Section 4.2.1), showing this is better-suited to the underlying geometry of the architecture search space.

Finally, Amended-DARTS (Bi et al., 2019) and iDARTS (Zhang et al., 2021a) both take the approach of deriving more accurate approximations of the gradients of α (Equation 1), showing that this leads to a more stable optimization and better generalization.

4.2.4 High Memory Consumption

The memory required to train a supernet is much higher than a normal neural net—it scales linearly with the size of the set of candidate operations. Recall from Section 4.1 that multiple works reduced this memory by, in each training step, masking out all operations except for the ones corresponding to one or a few subnetworks. Various works have proposed techniques to mask out operations for differentiable NAS as well, i.e., while simultaneously optimizing the architecture hyperparameters.

Cai et al. (2019) proposed ProxylessNAS, which solves this problem by modifying the BinaryConnect (Courbariaux et al., 2015) discretization method: in each training step, for each operation choice, all are masked out except one operation that is randomly chosen with probability proportional to its current value of α. Cai et al. (2019) show that this procedure converges to a single high-performing subnetwork. GDAS (Dong and Yang, 2019) and DSNAS (Hu et al., 2020; Xie et al., 2018) use a Gumbel-softmax distribution over a one-hot encoding of the operation choices, which is a different way to allow sampling single operations in each training step while maintaining differentiability.

PC-DARTS (Xu et al., 2019b) proposes a relatively simpler approach: at each training step, and for each edge in the DAG, a subset of channels is sampled and sent through the possible operations, while the remaining channels are directly passed on to the output. While reducing memory due to training fewer channels, this also acts as a regularizer. DrNAS (Chen et al., 2021f) also reduces memory consumption by progressively increasing the number of channels that are forwarded to the mixed operations, and progressively pruning operation choices, modeled by a Dirichlet distribution.

4.3 Hypernetworks

A *hypernetwork* is a neural network which generates the weights of *other* neural networks. Hypernetworks were first considered by Schmidhuber (1992, 1993), and the first modern application was by Ha et al. (2017), who used them to obtain better weights for a fixed LSTM architecture. Hypernetworks have since been used for a variety of tasks, including HPO (Mackay et al., 2019; Navon et al., 2021), calibrating model uncertainty (Krueger et al., 2017), and NAS (Brock et al., 2018; Zhang et al., 2018).

The first work to use hypernetworks for NAS (and among the first to use a one-shot model for NAS) was SMASH (one-Shot Model Architecture Search through Hypernetworks) (Brock et al., 2018). SMASH consists of two phases: first, train a hypernetwork to output weights for any architecture in the search space. Next, randomly sample a large set of architectures, generate their weights using the hypernetwork, and output the one with the best validation accuracy. The hypernetwork, a convolutional neural net, takes as input an architecture encoding and outputs a set of weights for that architecture, and is trained by randomly sampling an architecture, generating its weights, computing its training error, and then backpropagating through the entire system (including the hypernetwork weights).

Another hypernet-based NAS algorithm is GHN (Graph Hypernetworks) (Zhang et al., 2018). The main difference between SMASH and GHN is the architecture encoding and the architecture of the hypernetwork. Specifically, the GHN hypernetwork is a mix between a graph neural network and a standard hypernetwork. It takes as input the computational graph of an architecture a and uses message-passing operations which are typical in GNNs, to output the weights of a. The training of the hypernetwork, and the final NAS algorithm, are both the same as in SMASH.

5. Speedup Techniques

In this section, we cover general speedup techniques for NAS algorithms, including performance prediction (Section 5.1), multi-fidelity methods (Section 5.2), meta-learning approaches (Section 5.3), and weight inheritance (Section 5.4).

5.1 Performance Prediction

A large body of work has been devoted to predicting the performance of neural networks before they are fully trained. Such techniques have the potential to greatly speed up the runtime of NAS algorithms, since they remove the need to fully train each architecture under consideration. These speedup techniques can improve nearly all types of NAS algorithms, from black-box optimization (Ru et al., 2020a; White et al., 2021c) to one-shot NAS (Xiang et al., 2021). In this section, we discuss the performance prediction techniques themselves, while in Section 5.2, we discuss methods of incorporating them into NAS algorithms.

Formally, given a search space $\mathcal{A}$ and architecture $a \in \mathcal{A}$, denote the final validation accuracy obtained with a fixed training pipeline as $f(a)$. A *performance predictor* f' is defined as any function which predicts the accuracy or relative accuracy of architectures, without fully training them. In other words, evaluating $f'(a)$ takes less time than evaluating $f(a)$, and $\{f'(a) \mid a \in \mathcal{A}\}$ ideally has high correlation or rank correlation with $\{f(a) \mid a \in \mathcal{A}\}$. In the rest of this section, we give an overview of different types of performance predictors, including learning curve extrapolation (Section 5.1.1), zero-cost proxies (Section 5.1.2), and other methods (Section 5.1.3). Note that surrogate models (Section 3.4) and one-shot models (Section 4) can also be seen as types of performance predictors.

5.1.1 Learning Curve Extrapolation

Learning curve extrapolation methods seek to predict the final performance of a given architecture after partially training it, by extrapolating from its so-called partial *learning*

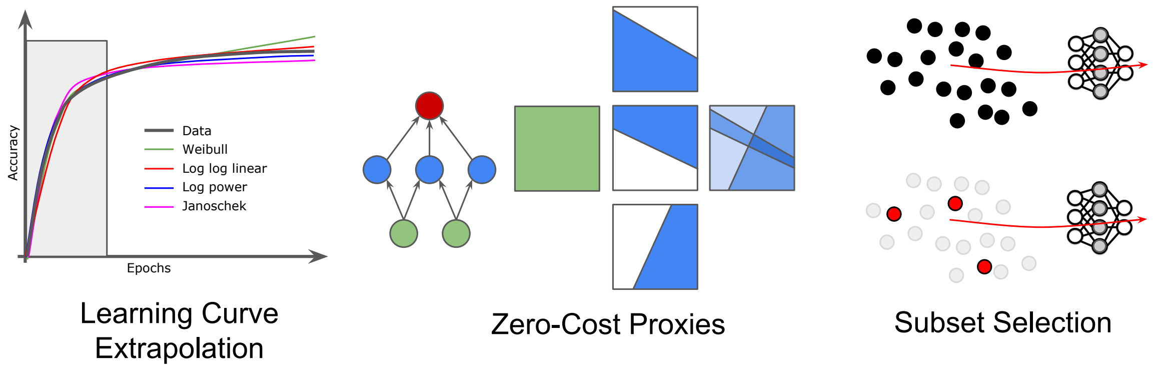

Figure 8: Illustration of the main types of performance predictors: extrapolating the validation accuracy learning curve via a parameteric model (left), assessing the generalizability of an architecture with a single forward pass of a single minibatch of data (middle), and training the architeture on a subset of the data (right).

curve (the series of validation accuracies at all epochs so far). This can, e.g., be accomplished by fitting the partial learning curve to a parametric model (Domhan et al., 2015) (see Figure 8 (left)). Learning curve extrapolation methods can also be used together with a surrogate model: in that case, the model takes as input both an encoding of a and a partial learning curve of a, and outputs a prediction $f'(a)$ (Baker et al., 2018; Klein et al., 2017). Learning curve extrapolation methods can be used to speed up black-box NAS algorithms (Domhan et al., 2015; Ru et al., 2020a; Yan et al., 2021b) or in conjunction with multi-fidelity algorithms such as Hyperband or BOHB (described in Section 5.2).

5.1.2 Zero-Cost Proxies

Zero-cost proxies are a recently developed family of performance prediction techniques. The idea is to run a very fast computation (such as a single forward and backward pass of a single minibatch of data) over a set of architectures that assigns a score to each architecture, with the hope that the scores are correlated with the final accuracies (Mellor et al., 2021). These techniques get their "zero-cost" name since the overall time to score each architecture is negligible (often less than 5 seconds) compared to most other performance prediction techniques (Abdelfattah et al., 2021). While most zero-cost proxies compute architecture scores from a (single) minibatch of data, some are *data-independent*, computing the score solely from the initialized weights or number of parameters of the neural network.

Zero-cost proxies were first introduced by Mellor et al. (2021), who estimated the relative performance of neural networks based on how well different linear regions of the network map are separated (see Figure 8 (middle)). Since the initial technique, several new zero-cost proxies have been introduced. Abdelfattah et al. (2021) made a connection to the pruning-at-initialization literature (Lee et al., 2019b; Tanaka et al., 2020; Theis et al., 2018; Wang et al., 2020a) and used this connection to introduce five zero-cost proxies. Their best-performing method, `synflow` (Tanaka et al., 2020), is a data-independent method which computes the L1 path-norm of the network: it computes the sum of the product of all initialized weights in each path connecting the input to the output.

Since then, two other data-independent methods have been introduced, based on a series of synthetic proxy tasks to test scale invariances and spatial information (Li et al., 2021d), and based on approximating the neural network as a piecewise linear function (Lin et al., 2021). Other data-dependent methods make use of the neural tangent kernel (NTK) (Jacot et al., 2018), based on approximating its trace norm (Shu et al., 2021) or approximating its spectrum (Chen et al., 2021e).

Although zero-cost proxies have received significant attention since they were first introduced, recent work has shown that simple baselines such as "number of parameters" and "FLOPs" are surprisingly competitive with all leading techniques. The main downsides of using zero-cost proxies are that they may be unreliable, especially on larger search spaces (Chen et al., 2022; Ning et al., 2021; White et al., 2022). They also may have biases, such as preferring larger models (Ning et al., 2021) or wide channels (Chen et al., 2022), although the biases can be removed (Krishnakumar et al., 2022).

On the other hand, recent work encourages the viewpoint that zero-cost proxies are "weak learners" which can be combined with other techniques, including other zero-cost proxies, to improve performance (Krishnakumar et al., 2022; White et al., 2022). Initial work shows that zero-cost proxies can be successfully added to both Bayesian optimization-based NAS (Shen et al., 2021; White et al., 2021c) and one-shot NAS (Xiang et al., 2021).

5.1.3 Other Low-Fidelity Predictions

Beside training for fewer epochs, other works give a low-fidelity estimate of the final accuracy by training on a subset of the training data (or a smaller, synthetically generated dataset). This is visualized in Figure 8 (right).

Multiple works have studied different subset selection algorithms, such as random sampling, entropy-based sampling (Na et al., 2021), clustering via core-sets (Shim et al., 2021), facility location (Prasad et al., 2022), and k-center (Na et al., 2021). Prasad et al. (2022) introduce adaptive subset selection to NAS, in which the subset is updated throughout training in order to maximize validation accuracy.

Such et al. (2020) introduce *generative teaching networks* which use a small set of synthetic data to train neural networks much faster than using the original real training data. The synthetic data is created using a data-generating network to match the accuracy of a network trained on real data. A related method is *synthetic petri dish* (Rawal et al., 2020), which evaluates architecture motifs by placing them into a small neural network and then training them using a small synthetic dataset. This latter method also explicitly optimizes the correlation between architecture rankings with the approximation and the full training.

5.2 Multi-Fidelity Algorithms

While the previous section was devoted to *methods* of predicting the performance of neural networks, now we cover algorithms that use these methods to run NAS efficiently.

Formally, the objective function $f : \mathcal{X} \longrightarrow \mathcal{R}$, which is typically expensive to fully evaluate, can be cheaply approximated by a lower-fidelity version $\hat{f}(\cdot, b)$ of $f(\cdot)$, parameterized by the fidelity parameter b. When $b = b_{max}$, we retrieve the true function $f(\cdot) = \hat{f}(\cdot, b_{max})$.

This is a generalization of the definition from Section 5.1. The fidelity parameter can denote the number of training epochs, training data subset size, and it can make use of performance prediction techniques from the previous section. One can even use multiple fidelity parameters at a time (Kandasamy et al., 2017; Zhou et al., 2020). Next, we describe the optimization algorithms that exploit access to multi-fidelity function estimates $\hat{f}(\cdot, b)$.

SuccessiveHalving (SH) (Jamieson and Talwalkar, 2016) is one of the simplest multi-fidelity algorithms. It starts to train a large number of architectures, slowly killing off more and more architectures which are not promising based on lower fidelity evaluations, until only the most promising architectures are evaluated at the highest fidelity. The fidelity thresholds and number of architectures to promote to higher fidelities are controlled by a hyperparameter. A popular improvement to SH is Hyperband (HB) (Li et al., 2018), a multi-armed bandit strategy that repeatedly calls SH as a subroutine, using different values of the minimum budget for each call. Therefore, HB hedges its bets against any single choice of the minimum budget.

While SH and HB are purely based on (smart) random search, recent works have combined HB with both Bayesian optimization and evolution. Bayesian optimization hyperband (BOHB) (Falkner et al., 2018; Lindauer et al., 2022) works similarly to HB in its first iteration, and on later iterations it fits a probabilistic surrogate model for each fidelity in order to make informed sampling decisions. Similarly, DEHB (Mallik and Awad, 2021) combines differential evolution (Storn and Price, 1997) with HB, significantly improving the later iterations of HB. ASHA (Li et al., 2020c) and ABOHB (Klein et al., 2020) improve SH and BOHB further, respectively, by making use of massively parallel asynchronous computation and early stopping strategies. Finally, EcoNAS (Zhou et al., 2020) proposes a hierarchical evolutionary search method that partitions the search space into subsets and allocates increasing fidelities to the most promising architectures in each subset.

5.3 Meta-Learning

A majority of NAS approaches consider solving a single task from scratch, ignoring previously explored solutions. However, this is in contrast to what both researchers and practitioners typically do. Often, architectures are transferred across datasets and even across tasks, and on a new task, researchers typically start with a state-of-the-art solution. So, one might ask: why run NAS from scratch rather than re-using information from, e.g., previous experiments? This question naturally leads to the idea of *meta-learning* or *learning to learn* (Hochreiter et al., 2001; Schmidhuber, 1987; Thrun and Pratt, 1998), which aims at improving a learning algorithm by leveraging information from past, related experiments (Hospedales et al., 2021; Vanschoren, 2019).

Wong et al. (2018) and Zimmer et al. (2021) employ meta-learning strategies in a more general automated machine learning setting. Since the focus is not on NAS, they both solely consider a small set of candidate architectures. In Wong et al. (2018), tasks are encoded in a similar fashion as word embeddings in NLP (Mikolov et al., 2013). In contrast, Zimmer et al. (2021) simply warm-start their search based on previously well-preforming configurations.

Lian et al. (2020) and Elsken et al. (2020) focus on few-shot learning: the problem of learning a new task with just a few data points for training. The authors extend gradient-based, model-agnostic meta-learning approaches such as MAML (Finn et al., 2017) and

REPTILE (Nichol et al., 2018) to not only meta-learning an initial set of weights for a fixed neural network architecture, but also to the architecture itself by incorporating a differentiable method such as DARTS (Liu et al., 2019c) into the meta-learning algorithm.

The work by Lee et al. (2021) is neither restricted to few-shot learning nor to choosing architectures from a small set of candidates. Rather, they employ typical NAS search spaces such as the ones discussed in Section 2. The authors propose a novel set encoder to improve upon deep sets (Zaheer et al., 2017) and set transformers (Lee et al., 2019a). A graph neural network-based decoder is employed to generate neural architectures given a set encoding. Additionally, a graph neural network is employed to encode generated architectures. The architecture encoding in combination with the set encoding is then used to meta-learn a surrogate model to predict the performance of the architecture, dataset tuple. Shala et al. (2022) extend the work by Lee et al. (2021) by employing the dataset and architecture encodings within a Bayesian optimization framework, resulting in a probabilistic surrogate predictor. This further enables adapting the surrogate to datapoints seen at test time.

5.4 Weight Inheritance and Network Morphisms

While black-box optimization-based NAS algorithms train each architecture from scratch, and one-shot methods train *all* architectures with the same set of weights, a line of work proposes an in-between solution: reuse the weights of trained architectures on similar untrained architectures. This idea is especially helpful for black-box optimization approaches that apply only small, sequential changes to architectures when generating a new candidate architecture. For example, Real et al. (2017) propose to copy the weights of all layers that have not been affected by applied mutations from the parent architecture to its offspring.

This idea has also been extended by the concept of *network morphisms* (Chen et al., 2016; Wei et al., 2016). Network morphisms are operators acting on the space of neural network architectures. They change the architecture of a neural network without changing the function they represent, i.e., given an arbitrary input, the output remains identical for the original architecture and the architecture having been modified by a network morphism. This is typically achieved by properly initializing the modified architecture. Network morphisms have been employed in evolutionary algorithms (Elsken et al., 2017, 2019a; Schorn et al., 2020; Wistuba, 2019), reinforcement learning (Cai et al., 2018a,b), Bayesian optimization (Jin et al., 2019b), and even one-shot methods (Fang et al., 2020).

6. Extensions

The previous sections studied the main techniques from the classic instantiation of NAS. In this section, we survey a few common extensions: joint NAS + HPO, constrained/multi-objective NAS, and neural ensemble search.

6.1 Joint NAS + HPO

While a large body of the NAS literature assumes fixed hyperparameters in their experimental setup, it has been shown – perhaps not very surprisingly – that hyperparameters also play a significant role. For example, on the DARTS search space, tuning hyperparameters can lead to a huge improvement, exceeding the performance gains obtained by NAS (Yang et al., 2020). However, the best hyperparameters may vary significantly across architectures even in the same search space (Yang et al., 2020). Therefore, a recent body of work seeks to overcome these challenges and give efficient algorithms for NAS + HPO (Dai et al., 2021; Dong et al., 2020; Izquierdo et al., 2021; Zela et al., 2018; Zhou et al., 2021).

Running joint NAS + HPO is significantly more challenging than running NAS or HPO in isolation. First, the complexity of the search space is substantially increased, due to the increased number of hyperparameters and the heterogeneity of the hyperparameters. Second, the interaction between architectures and training hyperparameters in terms of network performance is difficult to model. Furthermore, some hyperparameters can have different effects on the performance under different evaluation budgets, reducing the effectiveness of many multi-fidelity and performance prediction techniques.

In light of these challenges, several solutions have been proposed. Various methods have been introduced to homogenize the search space, such as reformulating NAS as an HPO problem with categorical hyperparameters (Zela et al., 2018), or standardizing the representation of the NAS and HPO hyperparameters by assigning continuous-valued coefficients in $[0, 1]$ (Dong et al., 2020). The search strategies resemble standard NAS algorithms such as BO (Dai et al., 2021; Izquierdo et al., 2021; Zela et al., 2018), evolution (Dai et al., 2021; Izquierdo et al., 2021), or REINFORCE with weight sharing (Dong et al., 2020).

6.2 Constrained and Multi-Objective NAS

Although NAS has been very popular in recent years, most work focuses on solely optimizing for a single objective, typically the accuracy or error rate. However, there are many settings for which this is not sufficient, such as when the neural network must be deployed on an edge device or must satisfy a legal definition of fairness. In such applications, we may need to constrain the latency, memory usage, or rate of errors across classes (Sukthanker et al., 2022). There has been particular interest in constraints related to edge devices and other hardware, termed *hardware-aware NAS* (Benmeziane et al., 2021). To achieve one or more objectives in addition to accuracy, the standard NAS objective is typically modified to either a *constrained* optimization problem (e.g., Bender et al. (2020); Cai et al. (2019); Tan et al. (2019)) or a *multi-objective* optimization problem (e.g., Elsken et al. (2019a); Hu et al. (2019); Izquierdo et al. (2021); Lu et al. (2019, 2020)).

In constrained optimization, one tries to solve the following equation:

$$\min_{a\in\mathcal{A}} f(a) \text{ subject to } h_i(a) \leq c_i \text{ for } \; i \in \{1, \dots, k\} \tag{2}$$

where $f(a)$ denotes, as before, the original objective function (e.g., validation error), and h_i represent hardware constraints as a function of the architecture. This problem is often solved by a transform into an additive or multiplicative unconstrained problem such as $\min_{a\in\mathcal{A}} f(a)+\sum_i \lambda_i g_i(a)$ with penalty functions g_i penalizing architectures not satisfying the constraints, e.g., $g_i(a) = \max\big(0, h_i(a)-c_i\big)$ and hyperparamters λ_i trading off the objectives and constraints. This single-objective optimization problem is then solved using black-box optimization methods or one-shot methods. In the latter case, the penalty functions g_i needs to be differentiable, which is often not the case. Therefore, discrete metrics such as latency are relaxed to continuous variables through various techniques, such as with a Gumbel softmax function (Wu et al., 2019b).

In multi-objective optimization, the requirements in Equation 2 are treated as separate objectives that are optimized along with the original objective:

$$\min_{a \in \mathcal{A}} \Big(f(a), h_1(a), \ldots, h_k(a) \Big).$$

While this can again be reduced to a single-objective problem via scalarization methods, another common approach is to search for a set of *non-dominated* solutions that are optimal in the sense that one cannot reduce any objective without increasing at least one other objective. The set of non-dominated solutions is called the *Pareto front*. The most common approach in this case is to employ multi-objective evolutionary algorithms which maintain a population of architectures and aim to improve the Pareto front obtained from the current population by evolving the current population (Elsken et al., 2019a; Hu et al., 2019; Izquierdo et al., 2021; Lu et al., 2019). Multi-objective evolutionary algorithms have also been used in combination with weight sharing within one-shot models (Lu et al., 2020; Muñoz et al., 2022).

One of the most widely-studied constrained NAS problems is regarding hardware efficiency such as memory or latency, and many works have been devoted to efficiently approximating hardware metrics of interest. While simple metrics such as number of parameters are easily computed, these are often not correlated enough with other metrics of interest such as memory or latency. Other solutions include computing hardware costs modularly as the sum of the hardware cost of each operation (Cai et al., 2019) or by using a surrogate model that predicts hardware costs (Dudziak et al., 2020; Laube et al., 2022).

6.3 Neural Ensemble Search

While the goal of neural architecture search is to return the best standalone architecture, ensembling methods are popular within the deep learning community for their robust predictions and their easy uncertainty quantification. A newly emerging extension of NAS is concerned with finding the best *ensemble* of neural networks with diverse architectures, which can outperform standard NAS in terms of accuracy, uncertainty calibration, and robustness to dataset shift (Zaidi et al., 2021). Neural ensemble search is defined as follows:

$$\begin{aligned} \min_{a_1,\ldots,a_M \in \mathcal{A}} & \mathcal{L}_{\text{val}} \left(\texttt{Ensemble} \left((w^*(a_1), a_1), \ldots, (w^*(a_M), a_M) \right) \right) \qquad (3) \\ \text{s.t.} \quad & w^*(a) = \text{argmin}_w \; \mathcal{L}_{\text{train}} \left(w, a \right) \quad \forall a \in \mathcal{A}, \end{aligned}$$

where `Ensemble` is the function which aggregates the outputs of $f_1, \ldots, f_M$. Note that the search space cardinality is $|\mathcal{A}|^M$ rather than $|\mathcal{A}|$ as in standard NAS.

Zaidi et al. (2021) propose two simple yet effective procedures based on random search and regularized evolution (Real et al., 2019) that search for architectures that optimize Equation 3. Despite their effectiveness, these algorithms take considerable computation due to the black-box nature of the optimization algorithms. Multi-headed NES (Narayanan et al., 2021) circumvents this issue by applying differentiable NAS methods on the heads of a multi-headed network. The heads are explicitly tuned to optimize the ensemble loss together with a diversity component that encourages uncorrelated predictions coming from the individual heads. Other works have set up neural ensemble search with a one-shot model for the entire architecture. NESBS (Neural Ensemble Search via Bayesian Sampling)

(Shu et al., 2022) propose to use a supernet to estimate the ensemble performance of independently trained base learners and then use Bayesian sampling to find a high-performing ensemble. NADS (Neural Architecture Distribution Search) (Ardywibowo et al., 2020) follows a similar line by training a supernet to optimize an objective that is tailored to provide better uncertainty estimates and out-of-distribution detection. Chen et al. (2021b) run evolutionary search on the supernet to find a high-performing ensemble.

7. Applications

Along with discovering improved architectures for well-known datasets, one of the primary goals of the field of NAS is to quickly and automatically find high-performing architectures for brand new datasets and tasks. Although the majority of the NAS literature focuses on image classification, there are numerous success stories for NAS applied to less well-known settings. In this section, we discuss a few of these successes, including graph neural networks, generative adversarial networks, dense prediction, and transformers.

7.1 Graph Neural Networks

Graph neural networks (GNNs) are designed to process data represented by graphs. Using NAS to design GNNs poses unique problems: the search space for GNNs is more complex than typical convolutional search spaces, and both NAS and GNNs are independently known for their large computational overhead.

Zhou et al. (2019) initiated a line of work applying NAS to GNNs by defining a new search space with GNN-specific operations and then using a reinforcement learning strategy. Follow-up work designed similar search spaces (Gao et al., 2020b; Zhang et al., 2021b). with specialized features such as meta-paths (Ding et al., 2021b), edge features (Jiang and Balaprakash, 2020), or fast sampling operations (Gao et al., 2020b).

Overall, the main difference between NAS for GNNs and more standard NAS settings lies in the construction of the search space. The main search strategies used by GNN NAS algorithms are typical NAS approaches: reinforcement learning (Gao et al., 2020b; Zhao et al., 2020a; Zhou et al., 2019), one-shot methods (Ding et al., 2021b; Zhao et al., 2020b), and evolutionary algorithms (Jiang and Balaprakash, 2020; Nunes and Pappa, 2020). For a detailed survey on NAS for GNNs, see Zhang et al. (2021b).

7.2 Generative Adversarial Network

Generative adversarial networks (GANs) (Goodfellow et al., 2014) are a popular choice for generative modeling in tasks such as computer vision. GANs make use of two separate networks training in tandem: a generator and a discriminator. Due to having two separate networks, and their notoriously brittle training dynamics (Gulrajani et al., 2017), GANs require special techniques for effective NAS.

Different works have achieved improved performance via NAS by searching for only the generator architecture with a fixed discriminator (Doveh and Giryes, 2021), with a predefined progressively growing discriminator (Fu et al., 2020), or by searching both the generator and discriminator architectures simultaneously (Gong et al., 2019). The most popular choice of search space is the cell-based search space. The cell for the generator consists of a standard convolutional cell, with the addition of various upsampling operations (Ganepola and Wirasingha, 2021; Gong et al., 2019; Tian et al., 2020).

The search techniques resemble the techniques used for standard NAS: reinforcement learning (Fu et al., 2020; Tian et al., 2020; Wang and Huan, 2019), one-shot NAS (Doveh and Giryes, 2021; Gao et al., 2020a; Lutz et al., 2018), and evolutionary algorithms (Kobayashi and Nagao, 2020), with scoring based on either Inception Score (IS) (Salimans et al., 2016) or Fréchet Inception Distance (FID) (Heusel et al., 2017). For a comprehensive survey on NAS for GANs, see Ganepola and Wirasingha (2021).

7.3 Dense Prediction Tasks

Dense prediction for computer vision encompasses a variety of popular tasks such as semantic segmentation, object detection, optical flow, and disparity estimation, and it requires more complex architectures compared to standard image classification problems. For example, the architectures often include a decoder (Ronneberger et al., 2015), modules for generating multi-scale features (He et al., 2015) or task-specific heads (Girshick et al., 2014) in addition to the main network. Thus, NAS algorithms have been applied to search for these components, either in isolation (Chen et al., 2018; Ghiasi et al., 2019; Xu et al., 2019a) or jointly (Guo et al., 2020a; Yao et al., 2020), or by discovering novel design patterns (Du et al., 2020). For a survey on NAS for dense prediction, see Elsken et al. (2022).

Once again, standard NAS techniques are used: Guo et al. (2020a); Liu et al. (2019a); Saikia et al. (2019); Xu et al. (2019a) employ gradient-based search via DARTS (Liu et al., 2019c); Du et al. (2020); Ghiasi et al. (2019) use RL; Bender et al. (2020) is inspired by ProxylessNAS (Cai et al., 2019) and ENAS (Pham et al., 2018).

Methods for dense prediction tasks (e.g., Bender et al. (2020); Chen et al. (2019b); Guo et al. (2020a); Shaw et al. (2019); Wu et al. (2019a)) typically build search spaces based on state-of-the-art image classification networks, with task-specific components from well-performing dense prediction architecture components. As many approaches fix the backbone and only search for other task-specific components of the architecture, they often employ pre-trained backbone architectures (Chen et al., 2020; Guo et al., 2020a) or even cache the features generated by a backbone (Chen et al., 2018; Nekrasov et al., 2019; Wang et al., 2020c) to speed up architecture search. Chen et al. (2018); Ghiasi et al. (2019) also use a down-scaled or different backbone architecture during the search process. Methods also sometimes employ multiple search stages, with the goal of first eliminating poorly performing architectures (or parts of the search space) and successively improving the remaining architectures (Du et al., 2020; Guo et al., 2020a).

Overall, while it is much harder to run NAS on dense prediction tasks compared to image classification tasks because of the computational demands of dense prediction, there has been a rapid increase in developments with the rise of computationally efficient one-shot NAS methods. While efforts thus far have focused on semantic segmentation and object detection, avenues for future work include disparity estimation, panoptic segmentation, 3D detection and segmentation, and optical flow estimation.

7.4 Transformers

Transformers were proposed by Vaswani et al. (2017) to help with the issue of longer sequences that RNNs had difficulty modeling, by using self-attention and cross-attention mechanisms such that each token's representation in an input sequence is computed from a weighted average of the representation of all other tokens. The core transformer design was introduced for machine translation, but it has found widespread usage in causal language modeling (Brown et al., 2020; Radford et al., 2019), masked language modeling (Clark et al., 2020; Devlin et al., 2019; Liu et al., 2019d), and more recently, computer vision (Dosovitskiy et al., 2021; Liu et al., 2021b). Since its release, there have been many efforts to improve transformers via NAS. The most common search strategies for transformers are evolutionary (Chen et al., 2021c; So et al., 2019, 2021) or one-shot (Ding et al., 2021a; Gong et al., 2021; Li et al., 2021a; Su et al., 2021) On the other hand, there is a huge variety of different search spaces that have been tried recently, relative to other areas (e.g., in NAS for convolutional architectures, the majority of works use cell-based search spaces). Overall, the field of NAS for transformers has not converged to one "best" type of search space. Below, we survey NAS methods for four types of transformers: decoder-only, encoder-only, encoder-decoder, and vision transformers. See Chitty-Venkata et al. (2022) for an in-depth survey.

Decoder-only architectures, such as the GPT line of architectures (Brown et al., 2020; Radford et al., 2019) directly consume the input text prompt and output the sequence of text tokens that are most likely to follow. Primer (So et al., 2021) is a NAS algorithm that makes use of evolutionary search on a large macro decoder-only search space. The approach found two consistent improvements to the transformer block: squaring the `ReLU` in the feedforward block in the transformer layer, and adding depthwise convolutions after self-attention heads.

Encoder-only architectures, such as BERT (Devlin et al., 2019) encode the input text into a representation which can be used for many kinds of downstream tasks. Multiple works (Xu et al., 2021a, 2022; Yin et al., 2021) seek to discover compressed versions of BERT, in which the desired latency and task are specified by the user. The typical approach is to train a supernet on a standard self-supervised task (masked language modeling), which can then be used to discover compressed models for a given language task.

Encoder-decoder architectures such as T5 (Raffel et al., 2020) are used in sequence-to-sequence tasks such as machine translation, in which the source language is encoded into a representation, which is then decoded into the target language. So et al. (2019) use evolutionary search together with a new technique to dynamically allocate more resources to more promising candidate models, while Zhao et al. (2021b) propose a DARTS-based algorithm with a new technique for memory efficiency in backpropagation. Finally, KNAS (Xu et al., 2021b) and SemiNAS (Luo et al., 2020) speed up the search using zero-cost proxies and a surrogate transformer model, respectively.

A large variety of NAS algorithms have been studied for vision transformer search spaces, with the majority using one-shot methods. AutoFormer (Chen et al., 2021c) searches over vision transformer architectures and hyperparameters using a single-path-one-shot strategy (Guo et al., 2020b) and then running evolutionary search on the trained supernet. A followup work, AutoFormerv2 (Chen et al., 2021d), automated the design of the search space itself by gradually evolving different search dimensions. Other works have improved supernet training via gradient conflict aware training (Gong et al., 2021) or channel-aware training (Su et al., 2021). Finally, Li et al. (2021a) and Ding et al. (2021a) run one-shot methods on hybrid CNN and transformer search spaces for computer vision.

8. Benchmarks

In the early days of NAS research, the most popular metrics were the final test accuracies on CIFAR-10 and ImageNet. This caused inconsistent search spaces and training pipelines across papers, and also drove up computational costs. For example, it became standard to train the final architecture for 600 epochs, even though the test accuracy only increases by a fraction of a percent past 200 epochs. Recently, queryable NAS benchmarks have helped the field reduce computation when developing NAS techniques and to achieve fair, statistically significant comparisons between methods.

A *NAS benchmark* (Lindauer and Hutter, 2020) is defined as a dataset with a fixed train-test split, a search space, and a fixed evaluation pipeline for training the architectures. A *tabular* NAS benchmark is one that additionally gives *precomputed evaluations* for all possible architectures in the search space. A *surrogate* NAS benchmark is a NAS benchmark along with a surrogate model that can be used to predict the performance of any architecture in the search space. A NAS benchmark is *queryable* if it is either a tabular or a surrogate benchmark. Queryable NAS benchmarks can be used to efficiently simulate many NAS experiments using only a CPU, by querying the performance of neural networks from the benchmark, rather than training them from scratch. In the rest of the section, we give an overview of popular NAS benchmarks. See Appendix Table 2 for a summary.

The first tabular NAS benchmark was NAS-Bench-101 (Ying et al., 2019). It consists of a cell-based search space of 423 624 architectures, each with precomputed validation and test accuracies on CIFAR-10 for three different seeds. A follow-up work, NAS-Bench-1Shot1 (Zela et al., 2020b), is able to simulate one-shot algorithms by defining subsets of the NAS-Bench-101 search space which have a fixed number of nodes. NAS-Bench-201 (Dong and Yang, 2020) is another popular tabular NAS benchmark, consisting of 6466 unique architectures, each with precomputed validation and test accuracies on CIFAR-10, CIFAR-100, and ImageNet-16-120 for three seeds each. NATS-Bench (Dong et al., 2021b) is an extension of NAS-Bench-201 which also includes a macro search space. Another extension, HW-NAS-Bench-201 (Li et al., 2021b), gives the measured or estimated hardware cost for all architectures across six hardware devices.

Surr-NAS-Bench-DARTS (formerly called NAS-Bench-301) (Siems et al., 2020) was the first surrogate NAS benchmark, created by training 60 000 architecture from the DARTS (Liu et al., 2019c) search space on CIFAR-10 and then training a surrogate model. The authors also released Surr-NAS-Bench-FBNet for the FBNet search space (Wu et al., 2019b). A follow-up work, NAS-Bench-x11 (Yan et al., 2021b), devised a technique to predict the full learning curve, allowing the validation accuracies to be queried at arbitrary epochs, which is necessary for simulating multi-fidelity NAS algorithms.

TransNAS-Bench-101 (Duan et al., 2021) is a tabular benchmark that covers seven different computer vision tasks from the Taskonomy dataset (Zamir et al., 2018). Beyond computer vision, NAS-Bench-NLP (Klyuchnikov et al., 2022) consists of an LSTM-inspired search space for NLP, and NAS-Bench-ASR (Mehrotra et al., 2021) is a tabular NAS benchmark for automatic speech recognition (Garofolo, 1993). NAS-Bench-360 (Tu et al., 2022a) is a benchmark suite which gives NAS benchmarks on ten diverse problems such as prosthetics control, PDE solving, protein folding, and astronomy imaging, and is search space agnostic, although three of the tasks have pretrained architectures on the NAS-Bench-201 search space. Finally, NAS-Bench-Suite (Mehta et al., 2022) is a benchmark suite which combines the majority of existing queryable NAS benchmarks, 28 total tasks, into a single unified interface. An extension, NAS-Bench-Suite-Zero, offers precomputed zero-cost proxy values across all tasks (Krishnakumar et al., 2022).

Using queryable benchmarks allows researchers to easily simulate hundreds of trials of the algorithms with different initial random seeds, making it easy to report statistically significant comparisons. However, over-reliance on a few benchmarks can lead to the field over-fitting (Koch et al., 2021; Raji et al., 2021) and is not conducive to the discovery of truly novel methods. Therefore, researchers should use a large set of diverse NAS benchmarks whenever possible.

9. Best Practices

The field of NAS has at times seen problems with reproducibility and fair, statistically significant comparisons among methods. These issues impede the overall research progress in the field of NAS. Recently, a few papers have laid out best practices and guidelines for conducting sound NAS research that is reproducible and makes fair comparisons (Li and Talwalkar, 2019; Lindauer and Hutter, 2020; Yang et al., 2020). These best practices are also available as a checklist (Lindauer and Hutter, 2020). We encourage NAS researchers to follow the checklist and to attach it to the appendix of their papers. Now, we summarize these best practices for NAS research.

9.1 Releasing Code and Important Details

It is nearly impossible to reproduce NAS methods without the full code. Even then, random seeds should be specified and reported. Furthermore, releasing easy-to-use code can lead to more follow-up methods and impact. For example, Liu et al. (2019c) released easy-to-use code for DARTS, which facilitated numerous follow-up works.

When releasing code, it is important to release all components, including the training pipeline(s), search space, hyperparameters, random seeds, and the NAS method. Many papers use different architecture training pipelines during the search and during the final evaluation, so it is important to include both. Note that using popular NAS benchmarks such as NAS-Bench-101 or NAS-Bench-201 (see Section 8) makes this substantially easier: the training pipeline is already fixed.

NAS methods often have several moving parts. As a result, they typically have many hyperparameters of their own that could be tuned. In fact, many NAS methods themselves make use of neural networks – one could even run a NAS algorithm on the NAS algorithm! Due to this complexity, it is important to report if, or how, these hyperparameters were tuned. When reporting results on a large set of search spaces and datasets, the best practice is to tune the hyperparameters of the NAS method on one dataset, and then fix these hyperparameters for the remaining evaluations on other datasets. We also note that, in general, devising NAS methods with fewer hyperparameters is more desirable, especially because it has recently been shown that hyperparameters often do not transfer well across datasets and search spaces (Mehta et al., 2022).

9.2 Comparing NAS Methods

When comparing NAS methods, it is not enough to use the same datasets. The exact same NAS benchmarks must be used: a dataset with a fixed train-test split, search space, and evaluation pipeline. Otherwise, it is unclear whether a difference in performance is due to the NAS algorithm or the training pipeline.

Several papers have shown that simple baselines are competitive with state-of-the-art NAS algorithms (Li and Talwalkar, 2019; Ottelander et al., 2021; Sciuto et al., 2020; White et al., 2021b). When desigining a new method for NAS, it is important to compare the method with baselines such as random sampling and random search. Furthermore, many NAS methods are anytime algorithms: a time budget does not necessarily need to be specified upfront, and the method can be stopped at any time, returning the best architecture found so far. The longer the NAS method runs, the better the final result. These NAS methods should be compared on a plot of *performance over time*. Even one-shot algorithms can be compared in this way, since the supernet can be discretized and trained at any point.

We recommend that NAS researchers run thorough ablation studies to show which part(s) of the NAS method lead to the most improved performance. As mentioned in the previous section, NAS methods often have several moving parts, so a clean understanding of the importance of each part and how they work together, is important to report. Finally, we recommend that researchers run multiple trials of their experiments and report the random seeds for each experiment. NAS methods can have high variance in the randomness of the algorithm, so running many trials is important to verify statistically significant comparisons.

10. Resources

In this section, we discuss NAS resources including libraries (Section 10.1), other survey papers (Section 10.2), and additional resources (Section 10.3).

10.1 Libraries

A long line of engineering has been focused on automating machine learning pipelines: Auto-WEKA (Thornton et al., 2013), Auto-Sklearn (Feurer et al., 2015), TPOT (Olson et al., 2016), and AutoGluon-Tabular (Erickson et al., 2020). More recently, a special focus has been given to developing tools that can facilitate the deployment of various NAS algorithms for practitioners, such as Auto-Keras (Jin et al., 2019a), Auto-PyTorch Tabular (Zimmer et al., 2021), AutoGluon (Erickson et al., 2020), and NNI (Microsoft, 2021).

To provide a toolbox for facilitating NAS research, in both developing new NAS methods and applying NAS to new problem domains, various libraries have been proposed. The DeepArchitect library (Negrinho and Gordon, 2017), which separates the search space from the optimizer, was an important first step towards this direction in the NAS community. NASLib (Ruchte et al., 2020) unifies and simplifies NAS research by having a single abstraction for one-shot and BBO algorithms, and a single abstraction for the search spaces of nearly all queryable NAS benchmark. Archai (Hu et al., 2019) also provides unified abstractions for one-shot and discrete NAS algorithms. The aim for Archai is both to support reproducible rapid prototyping for NAS research as well as to be a turnkey solution for data scientists looking to try NAS on their tasks. PyGlove (Peng et al., 2020) introduced a novel approach to constructing NAS methods via symbolic programming, in which the ML programs are mutable and can be manipulated and processed by other programs.

10.2 Other NAS Survey Papers

There are several older NAS survey papers. Elsken et al. (2019b) provides a compact introduction to NAS and introduces the "three pillars" of NAS: search space, search strategy, and performance evaluation strategy. The survey by Wistuba et al. (2019) provides a more comprehensive view of the landscape of NAS research, unifying and categorizing existing methods. Ren et al. (2020) gave a layout that focused on the historical challenges in the field of NAS, as well as the solutions found to remedy these challenges.

Other surveys have been released which focus on a specific sub-area of NAS. Liu et al. (2021a) focus on evolutionary NAS, Benmeziane et al. (2021) focus on hardware-aware NAS (HW-NAS), Zhang et al. (2021b) survey AutoML (with a NAS focus) on graphs, Elsken et al. (2022) survey NAS for dense prediction in computer vision, and Xie et al. (2021), Santra et al. (2021), and Cha et al. (2022) all survey one-shot NAS methods.

Finally, there are more survey papers with a broader focus such as automated machine learning (AutoML) or automated deep learning (AutoDL), which devote a section to NAS (Dong et al., 2021a; He et al., 2021; Kedziora et al., 2020; Yao et al., 2018; Yu and Zhu, 2020). Notably, the first book on automated machine learning (which is open-access) was released in May 2019 by Hutter et al. (2019).

10.3 Additional Resources

There are multiple long-running workshops which focus on NAS and related topics. The AutoML workshop at ICML (2014-2021) and Meta-Learning workshop at NeurIPS (2017-2022) have had a healthy overlap in attendance with the NAS community, especially over the last few years, while ICLR (2020, 2021) and CVPR (2021) have had workshops devoted solely to NAS. Finally, after many years of AutoML and NAS workshops, the community has grown large enough to start the first AutoML conference: `https://automl.cc/`.

For a continuously updated, searchable list of NAS papers, see `https://www.automl.org/automl/literature-on-neural-architecture-search/`. For a continuously updated list of NAS papers published at ML venues, as well as other resources, see `https://github.com/D-X-Y/Awesome-AutoDL`.

11. Future Directions

Neural architecture search has come a long way in the last few years. The efficiency of NAS algorithms has improved by orders of magnitude, tools exist to compare NAS algorithms without GPUs, and researchers have created many novel techniques and diverse search spaces. Architectures discovered by NAS constitute the state of the art on many tasks. However, there are still many unsolved problems and promising future directions. In this section, we discuss a few of the most important directions for future work in NAS.

11.1 Robustness of Efficient Methods

One-shot methods are one of the most popular techniques for NAS due to their orders-of-magnitude speedups over to black-box optimization techniques. While one-shot techniques have already seen major progress, they still face performance issues.

Even though many improvements of one-shot algorithms such as DARTS have been proposed (see Section 4.2), these works generally focus on a single improvement; the field lacks a large-scale, fair comparison among one-shot methods. Furthermore, as it currently stands, applying one-shot methods to a new task requires a significant amount of expertise. Devising one-shot approaches that work robustly and reliably across new datasets and tasks is an important area for future study.

Another more recent set of techniques that promises orders-of-magnitude speedups are zero-cost proxies (see Section 5.1.2). Although recent work has shown that many zero-cost proxies do not consistently outperform simple baselines (Ning et al., 2021), other work argues that there is untapped potential for zero-cost proxies (White et al., 2022), especially when combined with existing NAS techniques (White et al., 2021c; Xiang et al., 2021). Developing a better understanding of *when* and *why* zero-cost proxies work in certain settings is an important area for future research.

11.2 Going Beyond Hand-Crafted, Rigid Search Spaces

The search spaces for NAS methods are typically carefully hand-designed by human experts. While carefully designing search spaces decreases search times, it also contradicts the idea of having an automated system that can be employed by non-experts, and it limits the scope of NAS to domains where strong search spaces are available. Furthermore, in the last few years, the most-studied type of search space by far has been the cell-based search space, which is significantly more rigid than other types of search spaces.

Hierarchical search spaces offer a better trade-off between flexibility and ease of search, yet they are relatively under-explored when compared to cell-based search spaces (see Section 2.5). Furthermore, hierarchical search spaces by nature have a higher diversity when compared to cell-based search spaces, reducing the overall human bias of the search space.

Optimizing search spaces in an automated manner (Ru et al., 2020b) such as starting with large, diverse search spaces and then iteratively pruning low-performing parts of the space (Guo et al., 2020a; Radosavovic et al., 2020) could allow researchers to consider a significantly larger variety of architectures.

11.3 Fully Automated Deep Learning

Although NAS has seen a huge amount of interest, recent work has shown that on popular search spaces such as the DARTS search space, optimizing the training hyperparameters leads to a greater increase in performance than optimizing the architecture (Yang et al., 2020; Zela et al., 2020b). While these results show that for some search spaces, optimizing hyperparameters may be more important than optimizing the architecture, the best case scenario is to optimize both hyperparameters and the architecture simultaneously.

A new thread of research seeks to simultaneously optimize the hyperparameters and architecture: NAS + HPO (see Section 6.1). Varying hyperparameters along with the architecture also significantly reduces human bias, making it possible to discover previously unknown combinations of architectures and hyperparameters that substantially outperform existing methods. Therefore, while this problem is significantly more challenging than NAS or HPO alone, the potential improvements are much higher.

Furthermore, we do not need to stop just at NAS + HPO: we can optimize the full deep learning pipeline, including problem formulation, data processing, data augmentation, model deployment, and continuous monitoring. In other words, the goal is to run fully automated deep learning (AutoDL) (Dong et al., 2021a). As the field of NAS matures, AutoDL has the potential to play a big role in realizing substantial improvements in performance for real-world problems.

Acknowledgments and Disclosure of Funding

We thank Difan Deng and Marius Lindauer for creating and maintaining their literature list on neural architecture search, which has made this survey significantly easier to write (Deng and Lindauer, 2021). This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. We acknowledge funding by European Research Council (ERC) Consolidator Grant "Deep Learning 2.0" (grant no. 101045765). Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the ERC. Neither the European Union nor the ERC can be held responsible for them.

Funded by
the European Union

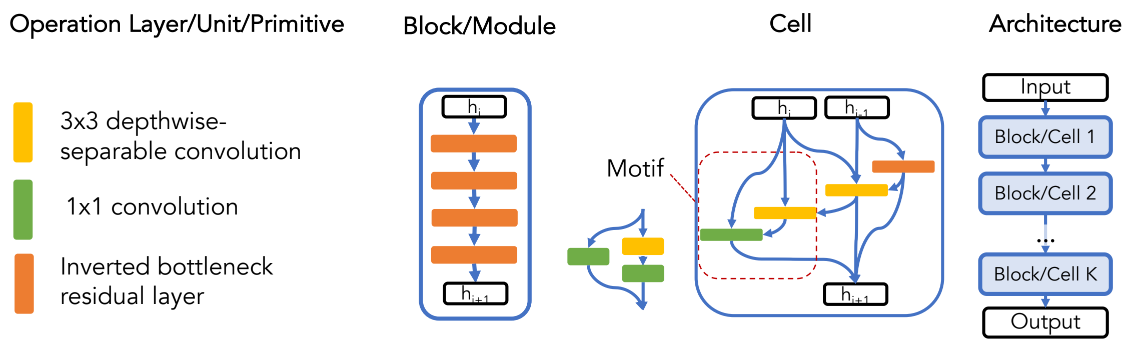

Figure 9: NAS search space terminology. Operation layers/units/primitives consist of sets of 1-3 operations. A block/module denotes a sequential stack of layers in chain-structured or macro search spaces. A cell denotes a directed acyclic graph of operations (and a motif denotes a small subset of the cell).

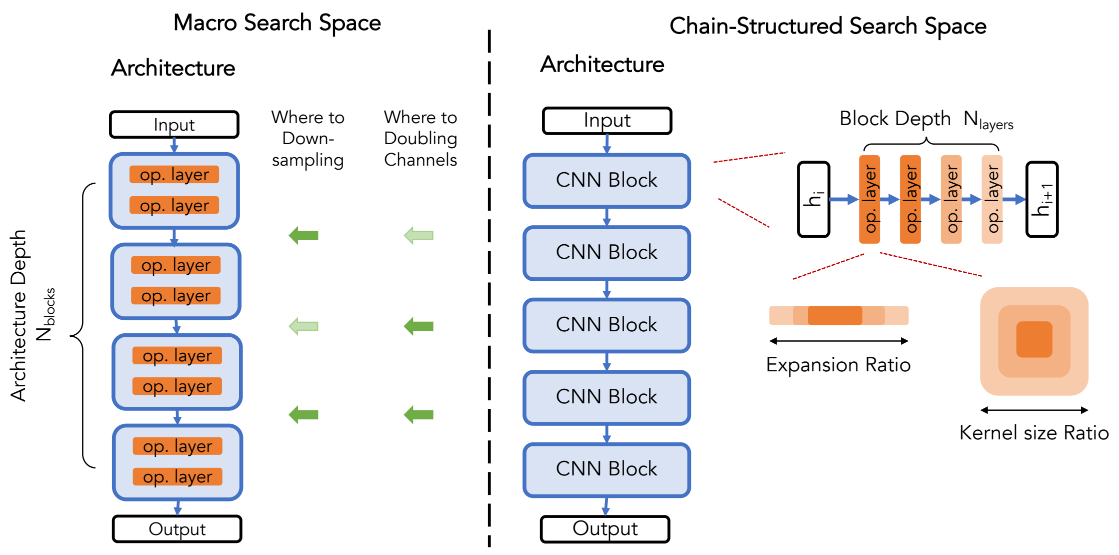

Figure 10: Illustration of macro search space based on Borsos et al. (2019)(left) and chain-structured search space based on Cai et al. (2020)(right).

A. Additional Figures and Tables

For a visualization of the search space terminologies, see Figure 9. In Figure 10, we show chain-structured and macro search spaces. Architecture encodings are illustrated in Figure 11. Finally, for an overview of NAS benchmarks, see Table 2.

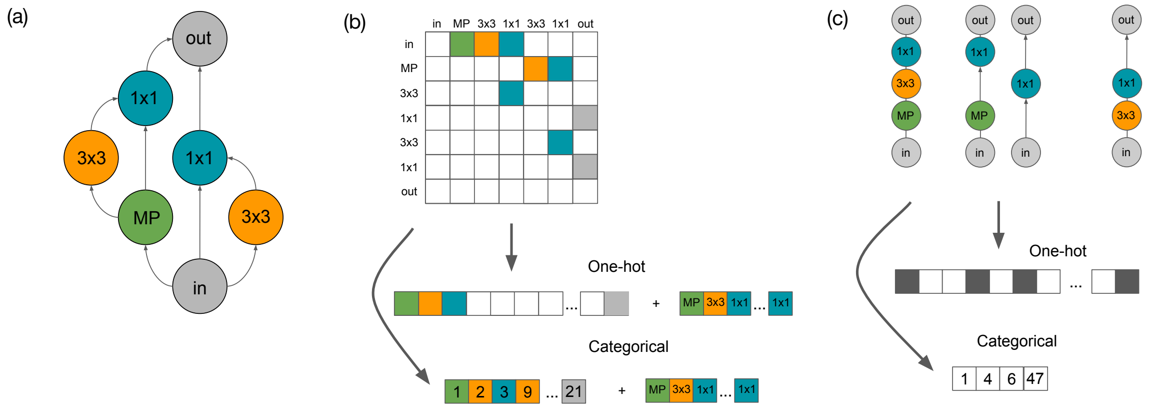

Figure 11: A neural architecture *(a)* can be encoded using an adjacency matrix *(b)* or path-based representation *(c)*, with a one-hot or categorical encoding.

Benchmark	Size	Type	Queryable Tab.	Queryable Surr.	LCs	One-Shot	Task	#Tasks
NAS-Bench-101	423k	cell	✓				Image class.	1
NATS-Bench-TSS (NAS-Bench-201)	6k	cell	✓		✓	✓	Image class.	3
NATS-Bench-SSS	32k	macro	✓		✓	✓	Image class.	3
NAS-Bench-NLP	$> 10^{53}$	cell			✓		NLP	1
NAS-Bench-1Shot1	364k	cell	✓			✓	Image class.	1
Surr-NAS-Bench-DARTS (NAS-Bench-301)	10^{18}	cell		✓		✓	Image class.	1
Surr-NAS-Bench-FBNet	10^{21}	chain		✓			Image class.	1
NAS-Bench-ASR	8k	cell	✓			✓	ASR	1
TransNAS-Bench-101-Micro	4k	cell	✓		✓	✓	Var. CV	7
TransNAS-Bench-101-Macro	3k	macro	✓		✓	✓	Var. CV	7
NAS-Bench-111	423k	cell		✓	✓		Image class.	1
NAS-Bench-311	10^{18}	cell		✓	✓	✓	Image class.	1
NAS-Bench-NLP11	$> 10^{53}$	cell		✓	✓		NLP	1
NAS-Bench-MR	10^{23}	cell		✓		✓	Var. CV	9
NAS-Bench-Macro	6k	macro	✓			✓	Image class.	1
HW-NAS-Bench-201	6k	cell	✓				Image class.	3
HW-NAS-Bench-FBNet	10^{21}	chain	✓				Image class.	1
NAS-Bench-360	Var.	suite	✓		✓	✓	Var.	3
NAS-Bench-Suite	Var.	suite	✓	✓	✓	✓	Var.	25
NAS-Bench-Suite-Zero	Var.	suite	✓	✓	✓	✓	Var.	28

Table 2: An overview of NAS benchmarks.

References

Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas Donald Lane. Zero-cost proxies for lightweight {nas}. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Abdulaziz Almalaq and Jun Jason Zhang. Evolutionary deep learning-based energy consumption prediction for buildings. *ieee access*, 7:1520–1531, 2018.

Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1): 54–65, 1994.

Randy Ardywibowo, Shahin Boluki, Xinyu Gong, Zhangyang Wang, and Xiaoning Qian. Nads: Neural architecture distribution search for uncertainty awareness. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 356–366. PMLR, 2020.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. arXiv preprint arXiv:1409.0473.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. In *Meta-Learning Workshop at NeurIPS*, 2018.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V. Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. *A Comprehensive Survey on Hardware-Aware Neural Architecture Search*. PhD thesis, LAMIH, Université Polytechnique des Hauts-de-France, 2021.

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyperparameter optimization. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2011.

Kaifeng Bi, Changping Hu, Lingxi Xie, Xin Chen, Longhui Wei, and Qi Tian. Stabilizing darts with amended gradient estimation on architectural parameters. *arXiv preprint arXiv:1910.11831*, 2019.

Zalán Borsos, Andrey Khorlin, and Andrea Gesmundo. Transfer nas: Knowledge transfer between search spaces with transformer agents. *6th ICML Workshop on Automated Machine Learning, arXiv preprint arXiv:1906.08102*, 2019.

Andrew Brock, Theo Lim, JM Ritchie, and Nick Weston. Smash: One-shot model architecture search through hypernetworks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33:1877–1901, 2020.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2018a.

Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-Level Network Transformation for Efficient Architecture Search. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018b.

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Stephen Cha, Taehyeon Kim, Hayeon Lee, and Se-Young Yun. Supernet in neural architecture search: A taxonomic survey. *arXiv preprint arXiv:2204.03916*, 2022.

William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4960–4964. IEEE, 2016.

Bo Chen, Golnaz Ghiasi, Hanxiao Liu, Tsung-Yi Lin, Dmitry Kalenichenko, Hartwig Adam, and Quoc V. Le. Mnasfpn: Learning latency-aware pyramid architecture for object detection on mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Boyu Chen, Peixia Li, Chuming Li, Baopu Li, Lei Bai, Chen Lin, Ming Sun, Junjie Yan, and Wanli Ouyang. Glit: Neural architecture search for global and local image transformer.

In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12–21, 2021a.

Hanlin Chen, Ming Lin, Xiuyu Sun, and Hao Li. NAS-bench-zero: A large scale dataset for understanding zero-shot neural architecture search, 2022. URL `https://openreview.net/forum?id=hP-SILoczR`.

Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. One-shot neural ensemble architecture search by diversity-guided search space shrinking. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16525–16534, 2021b.

Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12270–12280, 2021c.

Minghao Chen, Kan Wu, Bolin Ni, Houwen Peng, Bei Liu, Jianlong Fu, Hongyang Chao, and Haibin Ling. Searching the search space of vision transformer. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 34, 2021d.

Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021e. arXiv preprint arXiv:2102.11535.

Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1554–1565. PMLR, 2020.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021f.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1294–1303, 2019a.

Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. Detnas: Backbone search for object detection. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2019b.

Krishna Teja Chitty-Venkata, Murali Emani, Venkatram Vishwanath, and Arun K Somani. Neural architecture search for transformers: A survey. *IEEE Access*, 2022.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 28, 2015.

Aristeidis Chrostoforidis, George Kyriakides, and Konstantinos Margaritis. A novel evolutionary algorithm for hierarchical neural architecture search. *arXiv preprint arXiv:2107.08484*, 2021.

Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. In *European conference on computer vision*, pages 465–480. Springer, 2020.

Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. Darts-: robustly stepping out of performance collapse without indicators. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. arXiv preprint arXiv:2009.01027.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. arXiv preprint arXiv:2003.10555.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

Dennis D Cox and Susan John. A statistical method for global optimization. In *[Proceedings] 1992 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1241–1246. IEEE, 1992.

Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. Fbnetv3: Joint architecture-recipe search using predictor pretraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16276–16285, 2021.

Tri Dao, Nimit Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Difan Deng and Marius Lindauer. Literature list on Neural Architecture Search, 2021. URL `https://www.automl.org/automl/literature-on-neural-architecture-search/`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.

Mingyu Ding, Xiaochen Lian, Linjie Yang, Peng Wang, Xiaojie Jin, Zhiwu Lu, and Ping Luo. Hr-nas: Searching efficient high-resolution neural architectures with lightweight transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2982–2992, 2021a.

Yuhui Ding, Quanming Yao, Huan Zhao, and Tong Zhang. Diffmg: Differentiable meta graph search for heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 279–288, 2021b.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *The International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Xuanyi Dong, Mingxing Tan, Adams Wei Yu, Daiyi Peng, Bogdan Gabrys, and Quoc V Le. Autohas: Efficient hyperparameter and architecture search. *arXiv preprint arXiv:2006.03656*, 2020.

Xuanyi Dong, David Jacob Kedziora, Katarzyna Musial, and Bogdan Gabrys. Automated deep learning: Neural architecture search is not the end. *arXiv preprint arXiv:2112.09245*, 2021a.

Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. Nats-bench: Benchmarking nas algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021b.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. arXiv preprint arXiv:2010.11929.

Sivan Doveh and Raja Giryes. Degas: differentiable efficient generator search. *Neural Computing and Applications*, 33(24):17173–17184, 2021.

Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V. Le, and Xiaodan Song. Spinenet: Learning scale-permuted backbone for recognition and localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo 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 (CVPR)*, pages 5251–5260, 2021.

Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. Brp-nas: Prediction-based nas using gcns. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019a.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. In *JMLR*, 2019b.

Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. Meta-learning of neural architectures for few-shot learning. In *CVPR*, 2020.

Thomas Elsken, Arber Zela, Jan Hendrik Metzen, Benedikt Staffler, Thomas Brox, Abhinav Valada, and Frank Hutter. Neural architecture search for dense prediction tasks in computer vision, 2022.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

Jiemin Fang, Yuzhu Sun, Kangjian Peng, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Fast neural network adaptation via parameter remapping and architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 2962–2970, 2015.

Matthias Feurer and Frank Hutter. Hyperparameter optimization. In Hutter et al. (2019), pages 3–38.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary intelligence*, 1(1):47–62, 2008.

Peter I Frazier. A tutorial on bayesian optimization. *stat*, 1050:8, 2018.

Yonggan Fu, Wuyang Chen, Haotao Wang, Haoran Li, Yingyan Lin, and Zhangyang Wang. Autogan-distiller: searching to compress generative adversarial networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3292–3303, 2020.

Saya Fujino, Naoki Mori, and Keinosuke Matsumoto. Deep convolutional networks for human sketches by means of the evolutionary deep learning. In *2017 Joint 17th World Congress of International Fuzzy Systems Association and 9th International Conference on Soft Computing and Intelligent Systems (IFSA-SCIS)*, pages 1–5. IEEE, 2017.

Vayangi Vishmi Vishara Ganepola and Torin Wirasingha. Automating generative adversarial networks using neural architecture search: A review. In *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*, pages 577–582. IEEE, 2021.

Chen Gao, Yunpeng Chen, Si Liu, Zhenxiong Tan, and Shuicheng Yan. Adversarialnas: Adversarial neural architecture search for gans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5680–5689, 2020a.

Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In *The International Joint Conference on Artificial Intelligence (IJCAI)*, volume 20, pages 1403–1409, 2020b.

Roman Garnett. *Bayesian Optimization.* Cambridge University Press, 2023. to appear.

John S Garofolo. Timit acoustic phonetic continuous speech corpus. *Linguistic Data Consortium, 1993*, 1993.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Spencer Gibb, Hung Manh La, and Sushil Louis. A genetic algorithm for convolutional network structure optimization for concrete crack detection. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier, 1991.

Chengyue Gong, Dilin Wang, Meng Li, Xinlei Chen, Zhicheng Yan, Yuandong Tian, Vikas Chandra, et al. Nasvit: Neural architecture search for efficient vision transformers with gradient conflict aware supernet training. In *International Conference on Learning Representations*, 2021.

Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3224–3234, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 27, 2014.

Li Guilin, Zhang Xing, Wang Zitong, Li Zhenguo, and Zhang Tong. Stacnas: Towards stable and consistent optimization for differentiable neural architecture search. *Openreview submission https://openreview.net/forum?id=rygpAnEKDH*, 2019.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 30, 2017.

Jianyuan Guo, Kai Han, Yunhe Wang, Chao Zhang, Zhaohui Yang, Han Wu, Xinghao Chen, and Chang Xu. Hit-detector: Hierarchical trinity architecture search for object detection. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020a.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pages 544–560. Springer, 2020b.

David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016b.

Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(Jun):1809–1837, 2012.

José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 918–926, 2014.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 30, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. Learning to learn using gradient descent. In Georg Dorffner, Horst Bischof, and Kurt Hornik, editors, *Artificial Neural Networks — ICANN 2001*, pages 87–94, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*, 2022.

T. M. Hospedales, A. Antoniou, P. Micaelli, and A. J. Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hanzhang Hu, John Langford, Rich Caruana, Saurajit Mukherjee, Eric Horvitz, and Debadeepta Dey. Efficient forward architecture search. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Shou-Yong Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. Dsnas: Direct neural architecture search without parameter retraining. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12081–12089, 2020.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, page 507–523, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 9783642255656. doi: 10.1007/978-3-642-25566-3_40. URL `https://doi.org/10.1007/978-3-642-25566-3_40`.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

Carl Hvarfner, Frank Hutter, and Luigi Nardi. Joint entropy search for maximally-informed bayesian optimization. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

Sergio Izquierdo, Julia Guerrero-Viu, Sven Hauns, Guilherme Miotto, Simon Schrodi, André Biedenkapp, Thomas Elsken, Difan Deng, Marius Lindauer, and Frank Hutter. Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 31, 2018.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

Mojan Javaheripi, Shital Shah, Subhabrata Mukherjee, Tomasz Lukasz Religa, Caio Cesar Teodoro Mendes, Gustavo Henrique de Rosa, Sebastien Bubeck, Farinaz Koushanfar, and Debadeepta Dey. Litetransformersearch: Training-free on-device search for efficient autoregressive language models. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

Shengli Jiang and Prasanna Balaprakash. Graph neural network architecture search for molecular property prediction. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 1346–1353. IEEE, 2020.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019a.

Haifeng Jin, Qingquan Song, and Xia 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. ACM, 2019b.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Regularization is all you need: Simple neural nets can excel on tabular data. *arXiv preprint arXiv:2106.11189*, 2021.

Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabás Póczos. Multi-fidelity Bayesian optimisation with continuous approximations. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

David Jacob Kedziora, Katarzyna Musial, and Bogdan Gabrys. Autonoml: Towards an integrated framework for autonomous machine learning. *arXiv preprint arXiv:2012.12600*, 2020.

Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex systems*, 4(4):461–476, 1990.

Jyrki Kivinen and Manfred K Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *information and computation*, 132, 1997.

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Aaron Klein, Louis Tiao, Thibaut Lienart, Cedric Archambeau, and Matthias Seeger. Model-based asynchronous hyperparameter and neural architecture search. *arXiv preprint arXiv:2003.10865*, 2020.

Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, Alexander Filippov, and Evgeny Burnaev. Nas-bench-nlp: neural architecture search benchmark for natural language processing. *IEEE Access*, 10:45736–45747, 2022.

Masayuki Kobayashi and Tomoharu Nagao. A multi-objective architecture search for generative adversarial networks. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 133–134, 2020.

Bernard Koch, Emily Denton, Alex Hanna, and Jacob G Foster. Reduced, reused and recycled: The life of a dataset in machine learning research. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021. arXiv preprint arXiv:2112.01716.

Arjun Krishnakumar, Colin White, Arber Zela, Renbo Tu, Mahmoud Safari, and Frank Hutter. Nas-bench-suite-zero: Accelerating research on zero cost proxies. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2022.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

Deepika Kumari and Kamaljit Kaur. A survey on stereo matching techniques for 3d vision in image processing. *Int. J. Eng. Manuf*, 4:40–49, 2016.

Kevin Alexander Laube, Maximus Mutschler, and Andreas Zell. What to expect of hardware metric predictors in NAS, 2022. URL `https://openreview.net/forum?id=2DJn3E7lXu`.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, 1999.

Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. Rapid neural architecture search by learning to generate graphs from datasets. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019a.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019b.

Changlin Li, Tao Tang, Guangrun Wang, Jiefeng Peng, Bing Wang, Xiaodan Liang, and Xiaojun 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.

Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. {HW}-{nas}-bench: Hardware-aware neural architecture search benchmark. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021b.

Guohao Li, Guocheng Qian, Itzel C Delgadillo, Matthias Muller, Ali Thabet, and Bernard Ghanem. Sgas: Sequential greedy architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1620–1630, 2020a.

Jian Li, Yong Liu, Jiankun Liu, and Weiping Wang. Neural architecture optimization with graph vae. *arXiv preprint arXiv:2006.10310*, 2020b.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence (UAI)*, 2019.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In *Proceedings of the Conference on Machine Learning Systems (MLSys)*, 2020c.

Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021c.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *JMLR*, 2018.

Yuhong Li, Cong Hao, Pan Li, Jinjun Xiong, and Deming Chen. Generic neural architecture search via regression. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 34:20476–20490, 2021d.

Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao. Towards fast adaptation of neural architectures with meta learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. Zen-nas: A zero-shot nas for high-performance image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 347–356, 2021.

Marius Lindauer and Frank Hutter. Best practices for scientific research on neural architecture search. In *JMLR*, 2020.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 2022.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018a.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019a.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019b.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018b.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019c.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019d.

Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 2021a.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021b.

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

Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. Nsga-net: Neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2019.

Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *Computer Vision – ECCV 2020*, pages 35–51, Cham, 2020. Springer International Publishing.

Jovita Lukasik, David Friede, Arber Zela, Frank Hutter, and Margret Keuper. Smooth variational graph embeddings for efficient neural architecture search. In *International Joint Conference on Neural Networks (IJCNN)*, 2021.

Jovita Lukasik, Steffen Jung, and Margret Keuper. Learning where to look–generative nas is surprisingly efficient. In *The European Conference on Computer Vision (ECCV)*, 2022.

Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2952–2960, 2016.

Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Sebastian Lutz, Konstantinos Amplianitis, and Aljoscha Smolic. Alphagan: Generative adversarial networks for natural image matting. In *The British Machine Vision Conference (BMVC)*, 2018.

Lizheng Ma, Jiaxu Cui, and Bo Yang. Deep neural architecture search with deep graph bayesian optimization. In *2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 500–507. IEEE, 2019.

Matthew Mackay, Paul Vicol, Jonathan Lorraine, David Duvenaud, and Roger Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Neeratyoy Mallik and Noor Awad. Dehb: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization. In *The International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

Abhinav Mehrotra, Alberto Gil C. P. Ramos, Sourav Bhattacharya, Łukasz Dudziak, Ravichander Vipperla, Thomas Chau, Mohamed S Abdelfattah, Samin Ishtiaq, and Nicholas Donald Lane. Nas-bench-asr: Reproducible neural architecture search for speech recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Yash Mehta, Colin White, Arber Zela, Arjun Krishnakumar, Guri Zabergja, Shakiba Moradian, Mahmoud Safari, Kaicheng Yu, and Frank Hutter. Nas-bench-suite: Nas evaluation is (now) surprisingly easy. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 7588–7598. PMLR, 2021.

H Mendoza, A Klein, M Feurer, J Springenberg, and F Hutter. Towards automatically-tuned neural networks. In *ICML 2016 AutoML Workshop*, 2016.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

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

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2013.

Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

Jonas Močkus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

J. Pablo Muñoz, Nikolay Lyalyushkin, Yash Akhauri, Anastasia Senina, Alexander Kozlov, and Nilesh Jain. Enabling NAS with automated super-network generation. *AAAI 1st International Workshop on Practical Deep Learning in the Wild*, 2022.

Byunggook Na, Jisoo Mok, Hyeokjun Choe, and Sungroh Yoon. Accelerating neural architecture search via proxy data. *The International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

Ashwin Raaghav Narayanan, Arber Zela, Tonmoy Saikia, Thomas Brox, and Frank Hutter. Multi-headed neural ensemble search. In *Workshop on Uncertainty and Robustness in Deep Learning (UDL@ICML'21)*, 2021.

Aviv Navon, Aviv Shamsian, Gal Chechik, and Ethan Fetaya. Learning the pareto front with hypernetworks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik. Xnas: Neural architecture search with expert advice. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures. *stat*, 1050:28, 2017.

Vladimir Nekrasov, Hao Chen, Chunhua Shen, and Ian Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Vu Nguyen, Tam Le, Makoto Yamada, and Michael A Osborne. Optimal transport kernels for sequential and parallel neural architecture search. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 8084–8095. PMLR, 2021.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint*, 2018.

Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. In *European Conference on Computer Vision*, pages 189–204. Springer, 2020.

Xuefei Ning, Changcheng Tang, Wenshuo Li, Zixuan Zhou, Shuang Liang, Huazhong Yang, and Yu Wang. Evaluating efficient performance estimators of neural architectures. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 34, 2021.

Matheus Nunes and Gisele L Pappa. Neural architecture search in graph neural networks. In *Brazilian Conference on Intelligent Systems*, pages 302–317. Springer, 2020.

R. Olson, N. Bartley, R. Urbanowicz, and J. Moore. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In T. Friedrich, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'16)*, pages 485–492. ACM, 2016.

T Den Ottelander, Arkadiy Dushatskiy, Marco Virgolin, and Peter AN Bosman. Local search is a remarkably strong baseline for neural architecture search. In *International Conference on Evolutionary Multi-Criterion Optimization*, 2021.

Daiyi Peng, Xuanyi Dong, Esteban Real, Mingxing Tan, Yifeng Lu, Gabriel Bender, Hanxiao Liu, Adam Kraft, Chen Liang, and Quoc Le. Pyglove: Symbolic programming for automated machine learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

Aloïs Pourchot, Alexis Ducarouge, and Olivier Sigaud. To share or not to share: A comprehensive appraisal of weight-sharing. *arXiv preprint arXiv:2002.04289*, 2020.

Vishak Prasad, Colin White, Paarth Jain, Sibasis Nayak, Rishabh Iyer, and Ganesh Ramakrishnan. Speeding up NAS with adaptive subset selection. *arXiv preprint arXiv:2211.01454*, 2022.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollar. Designing network design spaces. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140), 2020.

Inioluwa Deborah Raji, Emily M Bender, Amandalynne Paullada, Emily Denton, and Alex Hanna. Ai and the everything in the whole wide world benchmark. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2021.

Aditya Rawal, Joel Lehman, Felipe Petroski Such, Jeff Clune, and Kenneth O. Stanley. Synthetic petri dish: A novel surrogate model for rapid architecture search, 2020.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

Esteban Real, Chen Liang, David So, and Quoc Le. Automl-zero: Evolving machine learning algorithms from scratch. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 8007–8019. PMLR, 2020.

Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903*, 2020.

Nicholas Roberts, Mikhail Khodak, Tri Dao, Liam Li, Christopher Ré, and Ameet Talwalkar. Rethinking neural operations for diverse tasks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, 2015.

Binxin Ru, Clare Lyle, Lisa Schut, Mark van der Wilk, and Yarin Gal. Revisiting the train loss: an efficient performance estimator for neural architecture search. *stat*, 1050:8, 2020a.

Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Neural architecture search using bayesian optimisation with weisfeiler-lehman kernel. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Robin Ru, Pedro Esperança, and Fabio Maria Carlucci. Neural architecture generator optimization. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33, 2020b.

Michael Ruchte, Arber Zela, Julien Siems, Josif Grabocka, and Frank Hutter. Naslib: a modular and flexible neural architecture search library, 2020.

Tonmoy Saikia, Yassine Marrakchi, Arber Zela, Frank Hutter, and Thomas Brox. Autodispnet: Improving disparity estimation with automl. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 29, 2016.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

Santanu Santra, Jun-Wei Hsieh, and Chi-Fang Lin. Gradient descent effects on differential neural architecture search: A survey. *IEEE Access*, 9:89602–89618, 2021.

Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

Jurgen Schmidhuber. Evolutionary principles in self-referential learning. on learning how to learn: The meta-meta-meta...-hook. Master's thesis, Technische Universitaet Muenchen, Germany, 1987.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Jürgen Schmidhuber. A 'self-referential'weight matrix. In *International conference on artificial neural networks*, pages 446–450. Springer, 1993.

Lennart Schneider, Florian Pfisterer, Martin Binder, and Bernd Bischl. Mutation is all you need. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021.

Christoph Schorn, Thomas Elsken, Sebastian Vogel, Armin Runge, Andre Guntoro, and Gerd Ascheid. Automated design of error-resilient and hardware-efficient deep neural networks. In *Springer Neural Computing and Applications*, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Gresa Shala, Thomas Elsken, Frank Hutter, and Josif Grabocka. Transfer NAS with meta-learned bayesian surrogates. In *Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2022.

Albert Shaw, Daniel Hunter, Forrest Landola, and Sammy Sidhu. Squeezenas: Fast neural architecture search for faster semantic segmentation. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2019.

Junhong Shen, Mikhail Khodak, and Ameet Talwalkar. Efficient architecture search for diverse tasks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

Yu Shen, Yang Li, Jian Zheng, Wentao Zhang, Peng Yao, Jixiang Li, Sen Yang, Ji Liu, and Cui Bin. Proxybo: Accelerating neural architecture search via bayesian optimization with zero-cost proxies. *arXiv preprint arXiv:2110.10423*, 2021.

Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James Kwok, and Tong Zhang. Bridging the gap between sample-based and one-shot neural architecture search with bonas. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Jae-hun Shim, Kyeongbo Kong, and Suk-Ju Kang. Core-set sampling for efficient neural architecture search. *arXiv preprint arXiv:2107.06869*, 2021.

Yao Shu, Shaofeng Cai, Zhongxiang Dai, Beng Chin Ooi, and Bryan Kian Hsiang Low. Nasi: Label-and data-agnostic neural architecture search at initialization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Yao Shu, Yizhou Chen, Zhongxiang Dai, and Bryan Low. Neural ensemble search via bayesian sampling. In *Uncertainty in Artificial Intelligence (UAI)*, 2022.

Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

David So, Quoc Le, and Chen Liang. The evolved transformer. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 2019.

David R. So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V. Le. Primer: Searching for efficient transformers for language modeling, 2021.

Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

Dehua Song, Chang Xu, Xu Jia, Yiyi Chen, Chunjing Xu, and Yunhe Wang. Efficient residual dense block search for image super-resolution. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 12007–12014, 2020.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 4134–4142, 2016.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning*. Omnipress, 2010.

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, dec 1997.

Xiu Su, Shan You, Jiyang Xie, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, Xiaogang Wang, and Chang Xu. Vitas: Vision transformer architecture search. *arXiv preprint arXiv:2106.13700*, 2021.

Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth Stanley, and Jeffrey Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 9206–9216. PMLR, 2020.

Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference*, pages 497–504, 2017.

Masanori Suganuma, Mete Ozay, and Takayuki Okatani. Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 4771–4780. PMLR, 2018.

Rhea Sukthanker, Samuel Dooley, John P Dickerson, Colin White, Frank Hutter, and Micah Goldblum. On the importance of architectures and hyperparameters for fairness in face recognition. *arXiv preprint arXiv:2210.09943*, 2022.

Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24 (2):394–407, 2019.

Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Jiancheng Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE transactions on cybernetics*, 50(9):3840–3854, 2020.

Kevin Swersky, David Duvenaud, Jasper Snoek, Frank Hutter, and Michael A. Osborne. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. *arXiv preprint arXiv:1409.4011*, 2014.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 6105–6114. PMLR, 2019.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33:6377–6389, 2020.

Manoel Tenorio and Wei-Tsih Lee. Self organizing neural networks for the identification problem. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 1, 1988.

Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In I. Dhillon, Y. Koren, R. Ghani, T. Senator, P. Bradley, R. Parekh, J. He, R. Grossman, and R. Uthurusamy, editors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, pages 847–855, 2013.

Sebastian Thrun and Lorien Pratt. Learning to learn. In *Springer Science+Business Media*, 1998.

Yuan Tian, Qin Wang, Zhiwu Huang, Wen Li, Dengxin Dai, Minghao Yang, Jun Wang, and Olga Fink. Off-policy reinforcement learning for efficient and effective gan architecture search. In *European Conference on Computer Vision*, pages 175–192. Springer, 2020.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

Renbo Tu, Nicholas Roberts, Mikhail Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. NAS-bench-360: Benchmarking neural architecture search on diverse tasks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2022a.

Renbo Tu, Nicholas Roberts, Vishak Prasad, Sibasis Nayak, Paarth Jain, Frederic Sala, Ganesh Ramakrishnan, Ameet Talwalkar, Willie Neiswanger, and Colin White. Automl for climate change: A call to action. *arXiv preprint arXiv:2210.03324*, 2022b.

Joaquin Vanschoren. Meta-learning. In Hutter et al. (2019), pages 39–68.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

Xingchen Wan, Binxin Ru, Pedro M Esparança, and Fabio Maria Carlucci. Approximate neural architecture search via operation distribution learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2377–2386, 2022a.

Xingchen Wan, Binxin Ru, Pedro M Esperança, and Zhenguo Li. On redundancy and diversity in cell-based neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022b.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020a.

Hanchao Wang and Jun Huan. Agan: Towards automated design of generative adversarial networks. *arXiv preprint arXiv:1906.11080*, 2019.

Linnan Wang, Yiyang Zhao, Yuu Jinnai, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1805.07440*, 2018.

Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Neural architecture search using deep neural networks and monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, number 06, pages 9983–9991, 2020b.

Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, and Yanning Zhang. Nas-fcos: Fast neural architecture search for object detection. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020c.

Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable nas. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3627–3635. PMLR, 2017.

Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

Lilian Weng. Neural architecture search, 2020. URL `https://lilianweng.github.io/posts/2020-08-06-nas/`.

Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021a.

Colin White, Sam Nolen, and Yash Savani. Exploring the loss landscape in neural architecture search. In *Uncertainty in Artificial Intelligence (UAI)*, pages 654–664. PMLR, 2021b.

Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021c.

Colin White, Mikhail Khodak, Renbo Tu, Shital Shah, Sébastien Bubeck, and Dey Debadeepta. A deeper look at zero-cost proxies for lightweight nas. In *ICLR Blog Track*, 2022. URL `http://0.0.0.0:4000/2021/12/01/zero-cost-proxies/`.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, may 1992.

Martin Wistuba. Finding competitive network architectures within a day using uct. *Proceedings of the 5th IEEE International Conference on Data Science and Advanced Analytics, pages 263-272*, 2018. arXiv preprint arXiv:1712.07420.

Martin Wistuba. Deep learning architecture search by neuro-cell-based evolution with function-preserving mutations. In Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 243–258, Cham, 2019. Springer International Publishing.

Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.

Catherine Wong, Neil Houlsby, Yifeng Lu, and Andrea Gesmundo. Transfer learning with neural automl. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019a.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt 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 (CVPR)*, pages 10734–10742, 2019b.

Yan Wu, Zhiwu Huang, Suryansh Kumar, Rhea Sanjay Sukthanker, Radu Timofte, and Luc Van Gool. Trilevel neural architecture search for efficient single image super-resolution. *arXiv preprint arXiv:2101.06658*, 2021.

Lichuan Xiang, Łukasz Dudziak, Mohamed S Abdelfattah, Thomas Chau, Nicholas D Lane, and Hongkai Wen. Zero-cost proxies meet differentiable architecture search. *arXiv preprint arXiv:2106.06799*, 2021.

Lingxi Xie and Alan Yuille. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1379–1388, 2017.

Lingxi Xie, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Lanfei Wang, Zhengsu Chen, An Xiao, Jianlong Chang, Xiaopeng Zhang, et al. Weight-sharing neural architecture search: A battle to shrink the optimization gap. *ACM Computing Surveys (CSUR)*, 54 (9):1–37, 2021.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

Hang Xu, Lewei Yao, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019a.

Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. Nas-bert. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, Aug 2021a. doi: 10.1145/3447548.3467262. URL `http://dx.doi.org/10.1145/3447548.3467262`.

Jin Xu, Xu Tan, Kaitao Song, Renqian Luo, Yichong Leng, Tao Qin, Tie-Yan Liu, and Jian Li. Analyzing and mitigating interference in neural architecture search. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 2022.

Jingjing Xu, Liang Zhao, Junyang Lin, Rundong Gao, Xu Sun, and Hongxia Yang. Knas: green neural architecture search. In *International Conference on Machine Learning*, pages 11613–11625. PMLR, 2021b.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019b.

Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. Does unsupervised architecture representation learning help neural architecture search? In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Shen Yan, Kaiqiang Song, Fei Liu, and Mi Zhang. Cate: Computation-aware neural architecture encoding with transformers. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021a.

Shen Yan, Colin White, Yash Savani, and Frank Hutter. Nas-bench-x11 and the power of learning curves. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021b.

Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. Nas evaluation is frustratingly hard. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Lewei Yao, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Sm-nas: Structural-to-modular neural architecture search for object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

Yichun Yin, Cheng Chen, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Autotinybert: Automatic hyper-parameter optimization for efficient pre-trained language models. In *ACL*, 2021.

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

Kaicheng Yu, Rene Ranftl, and Mathieu Salzmann. How to train your super-net: An analysis of training heuristics in weight-sharing nas. *arXiv preprint arXiv:2003.04276*, 2020.

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

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference*, 2016.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Sheheryar Zaidi, Arber Zela, Thomas Elsken, Chris C Holmes, Frank Hutter, and Yee Teh. Neural ensemble search for uncertainty estimation and dataset shift. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 34:7898–7911, 2021.

Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3712–3722, 2018.

Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906*, 2018.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020a.

Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020b.

Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

Haokui Zhang, Ying Li, Hao Chen, and Chunhua Shen. Memory-efficient hierarchical neural architecture search for image denoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3657–3666, 2020a.

Miao Zhang, Steven W Su, Shirui Pan, Xiaojun Chang, Ehsan M Abbasnejad, and Reza Haffari. idarts: Differentiable architecture search with stochastic implicit gradients. In *International Conference on Machine Learning*, pages 12557–12566. PMLR, 2021a.

Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Yuge Zhang, Zejun Lin, Junyang Jiang, Quanlu Zhang, Yujing Wang, Hui Xue, Chen Zhang, and Yaming Yang. Deeper insights into weight sharing in neural architecture search. *arXiv preprint arXiv:2001.01431*, 2020b.

Ziwei Zhang, Xin Wang, and Wenwu Zhu. Automated machine learning on graphs: A survey. *IJCAI Survey Track*, 2021b. arXiv preprint arXiv:2103.00742.

Huan Zhao, Lanning Wei, and Quanming Yao. Simplifying architecture search for graph neural network. *arXiv preprint arXiv:2008.11652*, 2020a.

Yiren Zhao, Duo Wang, Xitong Gao, Robert Mullins, Pietro Lio, and Mateja Jamnik. Probabilistic dual network architecture search on graphs. *arXiv preprint arXiv:2003.09676*, 2020b.

Yiyang Zhao, Linnan Wang, Kevin Yang, Tianjun Zhang, Tian Guo, and Yuandong Tian. Multi-objective optimization by learning space partition. In *International Conference on Learning Representations*, 2021a.

Yuekai Zhao, Li Dong, Yelong Shen, Zhihua Zhang, Furu Wei, and Weizhu Chen. Memory-efficient differentiable transformer architecture search. *Findings of the Association for Computational Linguistics*, 2021b.

Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11396–11404, 2020.

Kaichen Zhou, Lanqing Hong, Shoukang Hu, Fengwei Zhou, Binxin Ru, Jiashi Feng, and Zhenguo Li. Dha: End-to-end joint optimization of data augmentation policy, hyper-parameter and architecture. *arXiv preprint arXiv:2109.05765*, 2021.

Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-gnn: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184*, 2019.

Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.