A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions

PENGZHEN REN and YUN XIAO, Northwest University, USA
XIAOJUN CHANG, Monash University, Australia
PO-YAO HUANG, Carnegie Mellon University, USA
ZHIHUI LI, Qilu University of Technology (Shandong Academy of Sciences), China
XIAOJIANG CHEN and XIN WANG, Northwest University, USA

Deep learning has made substantial breakthroughs in many fields due to its powerful automatic representation capabilities. It has been proven that neural architecture design is crucial to the feature representation of data and the final performance. However, the design of the neural architecture heavily relies on the researchers' prior knowledge and experience. And due to the limitations of humans' inherent knowledge, it is difficult for people to jump out of their original thinking paradigm and design an optimal model. Therefore, an intuitive idea would be to reduce human intervention as much as possible and let the algorithm automatically design the neural architecture. ***Neural Architecture Search*** **(NAS)** is just such a revolutionary algorithm, and the related research work is complicated and rich. Therefore, a comprehensive and systematic survey on the NAS is essential. Previously related surveys have begun to classify existing work mainly based on the key components of NAS: search space, search strategy, and evaluation strategy. While this classification method is more intuitive, it is difficult for readers to grasp the challenges and the landmark work involved. Therefore, in this survey, we provide a new perspective: beginning with an overview of the characteristics of the earliest NAS algorithms, summarizing the problems in these early NAS algorithms, and then providing solutions for subsequent related research work. In addition, we conduct a detailed and comprehensive analysis, comparison, and summary of these works. Finally, we provide some possible future research directions.

CCS Concepts: • **Computing methodologies** → **Machine learning algorithms**;

Additional Key Words and Phrases: Neural architecture search, AutoDL, modular search space, continuous search strategy, neural architecture recycle, incomplete training

Pengzhen Ren and Yun Xiao contributed equally to this research.

This work was partially supported by the NSFC under Grants No. 61972315 and No. 61906109, the Shaanxi Science and Technology Innovation Team Support Project under Grant Agreement No. 2018TD-026, and the Australian Research Council Discovery Early Career Researcher Award No. DE190100626.

Authors' addresses: P. Ren and Y. Xiao, No.1, Xuefu Avenue, Changan District, Xian City, Shanxi Province, 710127, China; emails: pzhren@foxmail.com, yxiao@nwu.edu.cn; X. Chang, Monash University, No.1, Xuefu Avenue, Changan District, Xian City, Shanxi Province, 710127, China; email: cxj273@gmail.com; P.-Y. Huang, Carnegie Mellon University, 5000 Forbes AVE Pittsburgh PA 15213; Z. Li (corresponding author), Qilu University of Technology (Shandong Academy of Sciences), 19th Keyuan Road Lixia District, Jinan, Shandong Province China, 250014; X. Chen and X. Wang, Northwest University, 324 Wellington Rd, Clayton VIC 3800, Australia.

0360-0300/2021/05-ART76 $15.00
https://doi.org/10.1145/3447582

ACM Reference format:
Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2021. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *ACM Comput. Surv.* 54, 4, Article 76 (May 2021), 34 pages.
https://doi.org/10.1145/3447582

1 INTRODUCTION

Deep learning has already exhibited strong learning capabilities in many fields, including machine translation [1–3], image recognition [4, 6, 7], and object detection [8, 9, 10]. This is mainly due to the powerful automatic feature extraction capabilities offered by deep learning for unstructured data. Deep learning has transformed the traditional approach of manually designing features [13, 14] into automatic extraction [4, 30, 31], which has allowed researchers to focus on the design of neural architecture [11, 12, 19]. However, designing neural architecture heavily relies on the researchers' prior knowledge and experience; this makes it difficult for beginners to make reasonable modifications to the neural architecture in line with their actual needs. In addition, people's existing prior knowledge and fixed thinking paradigms are likely to limit the discovery of new neural architectures to a certain extent. As a result, ***Neural architecture search*** (**NAS**) was developed.

NAS aims to design a neural architecture that achieves the best possible performance using limited computing resources in an automated way with minimal human intervention [26, 114]. NAS-RL [11] and MetaQNN [12] are considered pioneers in the field of NAS. The neural architectures obtained by these works using ***reinforcement learning*** (**RL**) methods have reached state-of-the-art classification accuracy on image classification tasks. This demonstrates that automated neural architecture design is feasible. Subsequently, the work of Large-scale Evolution [15] once again verified the feasibility of this concept by using evolutionary learning to achieve similar results. However, these methods consume hundreds of GPU days or even more computing resources. This huge amount of computation is almost catastrophic for everyday researchers. Therefore, a large body of work has emerged on how to reduce the amount of calculation and accelerate the search for neural architecture [18, 19, 20, 49, 50, 53, 85, 106]. As the efficiency of search has improved, NAS has also been rapidly applied in the fields of object detection [66, 76, 112, 120], semantic segmentation [63, 65, 122], adversarial learning [54], speech recognition [64], architectural scaling [116, 124, 126], multi-objective optimization [40, 117, 127], platform-aware [29, 35, 104, 119], data augmentation [123, 125], and so on. Also, some works have considered how to strike a balance between performance and efficiency [118, 121]. Although NAS-related research has been highly abundant, it is still difficult to compare and reproduce NAS methods [79, 128, 129, 156]. This is because different NAS methods vary widely in terms of search space, hyperparameters, tricks, and so forth. Some works have also been devoted to providing a unified evaluation platform for popular NAS methods [79, 128].

1.1 Motivation

Due to the deepening and rapid development of NAS-related research, some methods that were previously accepted by researchers have proven to be imperfect by new research, leading to the development of an improved solution. For example, early incarnations of NAS trained each candidate neural architecture from scratch during the architecture search phase, leading to a surge in computation [11, 12]. ENAS [19] proposes to accelerate the architecture search process using a parameter sharing strategy. Due to the superiority of ENAS in terms of search efficiency, the weight sharing strategy was quickly recognized and adopted by a large number of researchers [23, 54, 55]. However, soon afterward, new research found that the widely accepted weight

sharing strategy is likely to lead to inaccurate ranking of candidate architectures [24]; this makes the algorithm difficult to the optimal neural architecture from a large number of candidate architectures, thereby further deteriorating the performance of the neural architecture that is eventually selected. Shortly afterward, DNA [21] modularized the large search space of NAS into blocks, enabling the candidate architecture to be fully trained to reduce the representation shift problem caused by the weight sharing. In addition, GDAS-NSAS [25] proposes a ***Novelty Search based Architecture Selection*** (**NSAS**) loss function to solve the problem of multi-model forgetting (i.e., when weight sharing is used to sequentially train a new neural architecture, the performance of the previous neural architecture is reduced) caused by weight sharing during the super network training process. Similar research clues are very common in the rapidly developing field of NAS research.

More concisely, this survey has the following motivations:

- Previous surveys often use the basic components of NAS to associate NAS-related work, which makes it difficult for readers to grasp the research ideas of NAS-related work.
- NAS-related fields are developing rapidly, and related work is complex and rich. There are obvious connections between different works, and existing surveys have not conducted a detailed and clear analysis of these links.

So a comprehensive and systematic survey based on challenges and solutions is highly beneficial to NAS research.

1.2 Our Contributions and Related Surveys

The contributions of this survey are summarized as follows:

- As far as we know, this is the first comprehensive and systematic review from the perspective of NAS challenges and corresponding solutions.
- We conduct a comprehensive analysis and comparison on the performance of existing NAS-related work and the optimization strategies they adopted.
- We analyze multiple possible development directions of NAS and point out two issues worthy of vigilance. The hyperparameter search of NAS is also discussed. They are very beneficial to the development of NAS.

Previous related surveys classify existing work mainly based on the basic components of NAS: search space, search strategy, and evaluation strategy [27, 28]. Although this classification method is more intuitive, it is not conducive to assisting readers in capturing the research clues. Therefore, in this survey, we first summarize the characteristics and corresponding challenges of the early NAS methods. Based on these challenges, we then summarized and categorized existing research to present readers with a comprehensive and systematic overview of the extant challenges and solutions. A more specific NAS-related survey framework comparison is shown in Table 1.

1.3 Article Organization

We first made an insightful summary of the characteristics of early NAS in Section 2. Then, in response to the challenges faced by early NAS, we conducted a comprehensive and systematic analysis of the optimization strategy used by NAS in Section 3 according to the following four parts: modular search space, continuous search strategy, neural architecture recycling, and incomplete training. In Section 4, we conducted a comprehensive performance comparison of NAS-related work. In Section 5, we discussed the future direction of NAS in detail. In Section 6, we explained the corresponding review threats. Finally, in Section 7, we made a summary and conclusion of this survey.

Table 1. Comparison of Article Frameworks of Different NAS-Related Surveys

Surveys	Classification standard	Main frame
Survey 1 [27]	Basic components	Search space, Search strategy, and Evaluation strategy
Survey 2 [28]	Basic components	
Our	Challenges and solutions	(Challenge → Solution:) Global search space → Modular search space Discrete search strategy → Continuous search strategy Search from scratch → Neural architecture recycling Fully trained → Incomplete training

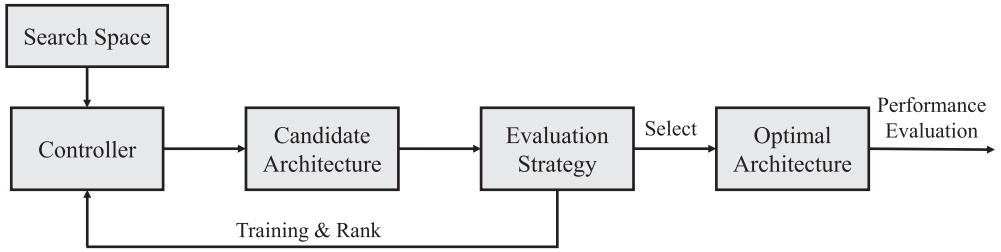


Fig. 1. The general framework of NAS.

2 CHARACTERISTICS OF EARLY NAS

In this section, we first analyze the early work of NAS and then summarize their overall framework and characteristics. The general framework of NAS is summarized in Figure 1. NAS generally begins with a set of predefined operation sets and uses a controller to obtain a large number of candidate neural architectures based on the search space created by these operation sets. The candidate architectures are then trained on the training set and ranked according to their accuracy on the validation set. The ranking information of the candidate neural architecture is then used as feedback information to adjust the search strategy, enabling a set of new candidate neural architectures to be obtained. When the termination condition is reached, the search process is terminated to select the best neural structure. The chosen neural architecture then conducts performance evaluation on the test set.

Early NAS also followed the above process to a large extent [11, 12, 15, 16]. The idea behind NAS-RL [11] comes from the very simple observation that the architecture of a neural network can be described as a variable-length string. Therefore, an intuitive idea is that we can use RNN as a controller to generate such a string, then use RL to optimize the controller, and finally obtain a satisfactory neural architecture. MetaQNN [12] regards the selection process of the neural architecture as a Markov decision process, and uses Q-learning to record rewards to obtain the optimal neural architecture. Large-scale Evolution [15] aims to learn an optimal neural architecture automatically using ***evolutionary algorithms*** (**EAs**) while reducing human intervention as much as possible. This approach uses the simplest network structure to initialize a large population, then obtains the best neural architecture by reproducing, mutating, and selecting the population. GeNet [16], which also uses EAs, proposes a new neural architecture coding scheme that represents the neural architecture as a fixed-length binary string. It randomly initializes a group of individuals, employs a predefined set of genetic operations to modify the binary string to generate new individuals, and finally selects the most competitive individual as the final neural architecture.

These early NAS approaches eventually made the automatically generated neural architecture a reality. To understand the reasons behind restricting the widespread use of early NAS, we have summarized the common characteristics existing in early NAS work from the perspective of a latecomer, as follows:

- **Global search space.** This requires the NAS to use a search strategy that searches all necessary components of the neural architecture. This means that NAS needs to find an optimal neural architecture within a very large search space. The larger the search space, the higher the corresponding search cost.
- **Discrete search strategy.** This regards the differences between different neural architectures as a limited set of basic operations; that is, by discretely modifying an operation to change the neural architecture. This means that we cannot use the gradient strategy to quickly adjust the neural architecture.
- **Search from scratch.** In this approach, the model is built from scratch until the final neural architecture is generated. These methods ignore the existing neural architecture design experience and are unable to utilize the existing excellent neural architecture.
- **Fully trained.** This approach requires training each candidate neural architecture from scratch until convergence. The network structures of the subsequent network and previous neural architectures are similar, as are those of the neural architectures at the same stage. Therefore, it is clear that this relationship not be fully utilized if each candidate neural architecture is trained from scratch. Also, we only need to obtain the relative performance ranking of the candidate architecture. Whether it is necessary to train each candidate architecture to convergence is also a question worth considering.

To more clearly show the relationship between the characteristics of early NAS and NAS key components, we follow the expressions in the previous two NAS-related surveys [27, 28]; they generally regard search space, optimization methods, and performance estimation strategies as the three major components of NAS. In this article, the global search space and full training of early NAS correspond to the two components of NAS search space and performance evaluation strategy, respectively. The discrete search strategy and search from scratch in the early NAS correspond to the search strategy used in the RL and EA optimization methods in NAS.

More specifically, the search space is determined by the predefined operation set and the hyperparameters of the neural architecture (e.g., architectural template, connection method, the number of convolutional layer channels used for feature extraction in the initial stage). These parameters define which neural architectures can be searched by the NAS. Figure 2 presents examples of two common global search spaces with a chain structure in early NAS work. o_i is an operation in the candidate operation set and the i-th operation in the chain structure. The feature map generated by o_i is represented as $z^{(i)}$. The input undergoes a series of operations to obtain the final output. Figure 2 (left): The simplest example of a chain structure MetaQNN [12]. At this point, for any feature map $z^{(i)}$, there is only one input node $z^{(i-1)}$, and $z^{(i)} = o_i\{(z^{(i-1)})\}$. Figure 2 (right): The example after skip connections are added [11, 15, 16]. At this time, there can be multiple inputs for any feature map $z^{(i)}$, and

$$z^{(i)} = o^{(i)}\left(\left\{z^{(i-1)}\right\} \odot \left\{z^{(k)} | \alpha_{k,i} = 1, k < i-1\right\}\right), \tag{1}$$

where $\odot$ can be a sum operation or a merge operation; for example, $\odot$ is a merge operation in NAS-RL [11], and $\odot$ is an element-wise summation operation in GeNet [16]. It should be pointed out here that NASNet [32] considers these two operations in the experiment, but the experimental results demonstrate that the sum operation is better than the merge operation. Accordingly, since then, a large number of works have taken the summation operation as the connection method of

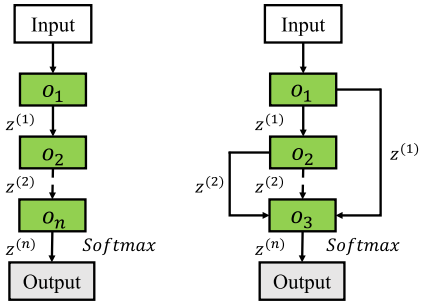


Fig. 2. Two common global search spaces with chain structure in early NAS work. Left: The simplest example of a chain structure. Right: The chain structure example after adding skip connections. o_i is an operation in the candidate set of operations and the i-th operation in the chain structure. The feature map generated by o_i is represented as $z^{(i)}$. The input undergoes a series of operations to obtain the final output.

the feature map obtained between different branch operations [17, 37, 38]. Like the chain structure, Mnasnet [29] suggests searching for a neural architecture that is composed of multiple segments and connected in sequence, with each segment having its repeating structure.

In addition, in early NAS works, searching from scratch was a commonly adopted strategy. NAS-RL [11] expresses the neural architecture as a string of variable length that is generated by RNN as a controller. The corresponding neural architecture is then generated according to the string, after which reinforcement learning is used as the corresponding search strategy to adjust the NAS. MetaQNN [12] trains an agent to sequentially select the layer structure of the neural network on the search space constructed by the predefined operation set. This approach regards the layer selection process as a Markov decision process, and also uses Q-learning as a search strategy to adjust the agent's selection behavior. Similar to NAS-RL [11], GeNet [16] also adopts the concept of encoding the network structure. The difference is that in GeNet [16], the neural architecture representation is regarded as a string of fixed-length binary codes, which are regarded as the DNA of the neural architecture. The population is initialized randomly, after which evolutionary learning is used to reproduce, mutate, and select the population, and then to iterate to select the best individual. It can be seen from the above analysis that these methods do not employ the existing excellent artificially designed neural architecture, but instead, search the neural architecture from scratch in their respective methods. More simply, Large-scale Evolution [15] uses only a single-layer model without convolution as the starting point for individual evolution. Evolutionary learning methods are then used to evolve the population, and then to select the most competitive individuals in the population. We take Large-scale Evolution [15] as an example and present an example of searching from scratch in Figure 3.

The common characteristics of these early NAS works are also the collective challenges faced by the automatic generation of neural architecture. We summarize the solutions to the above-mentioned challenges in the subsequent NAS-related research work in Section 3.

3 OPTIMIZATION STRATEGY

Regarding the characteristics and challenges of early NAS [11, 12, 15, 16], in this section, we summarize the existing NAS research work regarding the following four aspects:

- ***Modular search space.*** Corresponding to the global search space, the modular search space treats the neural architecture as a stack of multiple different types of modules. Therefore,

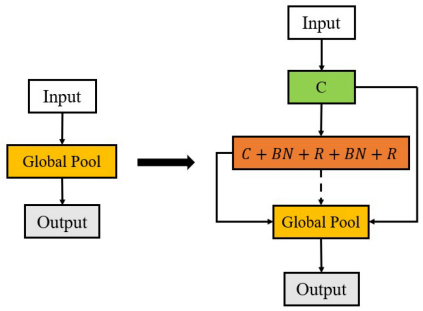


Fig. 3. Taking Large-scale Evolution [15] as an example, beginning from the simplest neural architecture to gradually generate the final neural architecture. C, BN, and R denote the convolution, Batch Normalization, and ReLU operations in sequence.

the search task is simplified from the original global search to only one or more modules of different types.

- ***Continuous search strategy.*** Corresponding to the discrete search strategy, the continuous search strategy continuously relaxes the structural parameters of the neural architecture so that they can be gradient optimized like network parameters.
- ***Neural architecture recycling.*** Corresponding to the search from scratch, neural architecture recycling takes the existing artificially designed high-performance neural architecture as a starting point and uses the NAS method to perform network transformations on them to improve their performance.
- ***Incomplete training.*** Corresponding to fully training, incomplete training aims to speed up the relative performance ranking of candidate architectures by making full use of the shared structure between candidate architectures or performance prediction, thereby avoiding resource consumption caused by complete training of all candidate architectures.

3.1 Modular Search Space

Search space design has a critical impact on the final performance of the NAS algorithm. It not only determines the freedom of the NAS but also directly determines the NAS algorithm's upper-performance limit to some extent. Therefore, the reconstruction of the search space is necessary.

One widely used approach is to transform the global search into a modular search space. As a result, cell or block-based search space is commonly used in various NAS tasks because it can effectively reduce the complexity of NAS search tasks. This is mainly because the cell-based search space often needs to search only a few small cell structures, after which it repeatedly stacks such cells to form the final neural architecture. However, the global search space needs to search for all the components involved in building the entire neural architecture. Besides, the cell-based search space can be migrated to different dataset tasks by stacking different numbers of cells, but this is often not possible when the global search space is used. Therefore, compared with the global search space, the cell-based search space is more compact and flexible.

This concept mainly stems from the observation of the excellent neural architectures that have been artificially designed in recent years [4, 30, 31]. These artificial neural architectures typically accomplish the construction of the overall neural architecture by repeatedly stacking a certain unit operation or a small structure. In the NAS context, this small repeating structure is often called a cell. The construction of cell-based neural architecture is based on this idea. Neural architecture constructed in this way is not only superior in terms of performance but also easy to generalize.

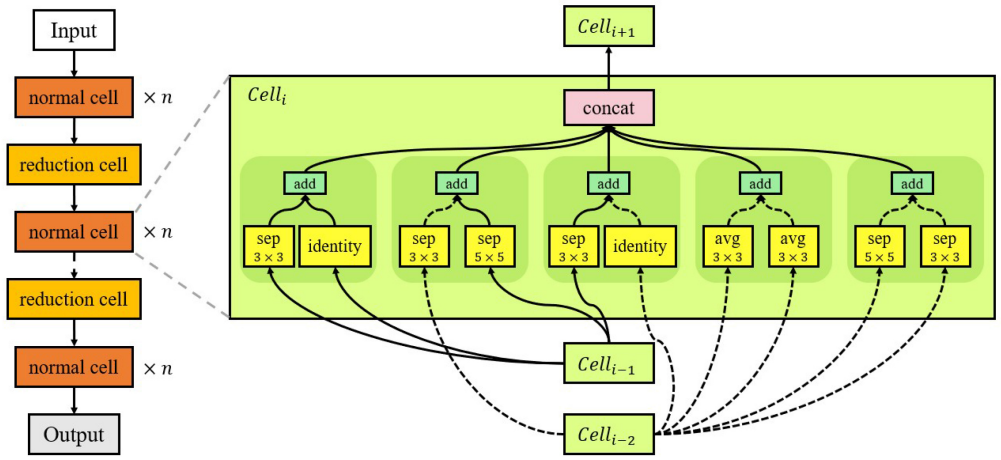


Fig. 4. Left: The structure of the search space example based on two cells in [32]. The normal cell is repeated n times and then connected to a reduction cell. The basic operation of the normal cell has a stride of 1, and the size of the feature map remains unchanged before and after output. The basic operation stride of the reduction cell is 2, and the size of the feature map is halved. Right: The best normal cell with five blocks searched in [32].

NASNet [32] is one of the first works to explore this idea. It proposes to search for two types of cells, namely, normal cells and reduction cells. Normal cells are used to extract advanced features while keeping the spatial resolution unchanged, and reduction cells are mainly used to reduce the spatial resolution. Multiple repeated normal cells are followed by a reduction cell; this connection is then repeated multiple times to form the final neural architecture. In Figure 4 (left), we illustrate this kind of neural architecture based on two cells. In Figure 4 (right), we present the internal structure of an optimal normal cell in NASNet [32]. The structures of the corresponding reduction cell and normal cell are similar; the difference is that the basic operation step of the reduction cell is 2. A large number of subsequent works [17, 43, 44] have used a search space similar to NASNet [32].

In ENAS [19], its experiments provide strong evidence for the utilization of this similar cell-based search space. Subsequently, this cell-based search space is widely used in other research work. In [33, 34, 35, 45], to complete downsampling, some unit operations are selected to replace reduction cell; at this time, the model only needs to search for a normal cell. We illustrate this structure in Figure 5. Here, the curved dotted line indicates the dense connection in Dpp-net [35]. At the same time as Block-QNN [33] of NASNet [32], the pooling operation is used in place of the reduction cell to reduce the size of the feature map. Hierarchical-EAS [34] uses convolution with a kernel size of 3×3 and a stride of 2 instead of the reduction cell to reduce the spatial resolution. Furthermore, the idea of meta-operation is used to hierarchically build the cell structure. Dpp-net [35] is similar to Block-QNN [33], but uses average pooling operation instead of a reduction cell. The difference is that Dpp-net [35] draws on the concept of DenseNet [36] to use dense connections, including cells, to build a neural architecture, and further proposes to take devices into account for multi-objective optimization tasks. In [33, 34, 35], the structure of each cell is the same, and it is only necessary to search for a cell. For video tasks, [45] uses $L \times 3 \times 3, stride = 1, 2, 2$ max-pooling instead of a reduction cell.

Moreover, to adapt to the complex task of video and expand the search space, the structure of each cell can be made different. AutoDispNet [38] proposes to apply automatic architecture search

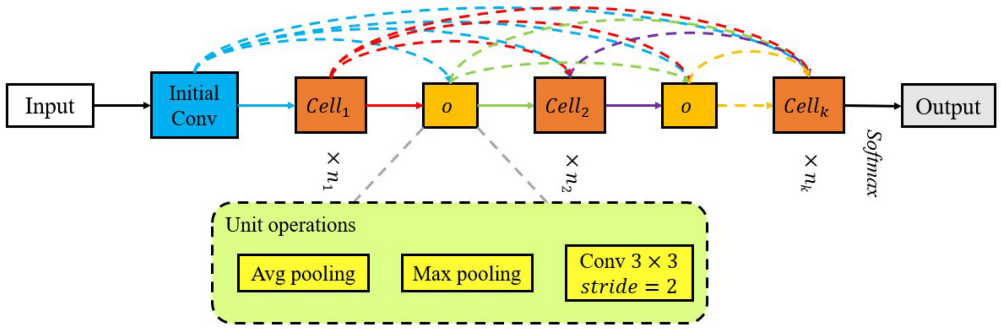


Fig. 5. Using unit operations to replace the reduction cell in [32], while having a densely connected neural architecture. The curved dotted line indicates the dense connection in Dpp-net [35]. The initial convolution is to extract low-level features. After the cell is repeated multiple times, a unit operation is used for downsampling. This connection is repeated multiple times to form the final neural architecture.

technology in order to optimize large-scale U-Net-like encoder-decoder architectures. Therefore, it searches for three types of cells: normal, reduction, and upsampling. In the coding stage, the neural architecture comprises alternate connections of normal cells and reduction cells. In the decoding stage, it consists of a stack of multiple upsampling cells. [18] studies the structural commonality of the cells obtained from some popular cell-based search spaces [17, 19, 32, 43, 46] and defines the width and depth of cells. [18] further proves theoretically and experimentally that due to the existence of the common connection mode, wide and shallow cells are easier to converge during training and easier to be searched, but the generalization effect is poor. This provides guidance that helps us to understand the cell-based NAS. In addition, there are many cell-based NAS-related works [54, 92].

In addition to repeatedly stacking one or more identical cells, FPNAS [39] also considers the diversity of blocks when stacking blocks. The experimental results of FPNAS show that stacking diversified blocks is beneficial to the improvement of neural architecture performance, and FPNAS treats the search process as a bi-level optimization problem, which reduces the search cost to a level similar to that of the most advanced NAS method [17, 19, 92]. Similar to FPNAS, FBNet [103] explores a layer-wise search space. Specifically, FBNet fixes the macro-architecture and searches for blocks with multiple layers. Moreover, each block can have a different layer structure, and the blocks can also be different.

In this section, we conduct a comprehensive review of the modular search space. Compared with global search, the modular search space more effectively reduces the search space and makes NAS more accessible to researchers. Of course, this does not mean that the modular search space can meet all task requirements. Global search still has a unique research value because it provides the neural architecture design with a higher degree of freedom [104, 154].

3.2 Continuous Search Strategy

NAS is regarded as a revolution in neural architecture design. However, NAS also requires high computational demand. For example, NASNet [32] uses the RL methods to spend 2,000 GPU days to obtain the best architecture in CIFAR-10 and ImageNet. Similarly, AmoebaNet-A [43] spends 3,150 GPU days using evolutionary learning. One internal reason why these mainstream search methods based on RL [11, 12, 32], EA [15, 43], Bayesian optimization [61], SMBO [37], and MCTS [62] are so inefficient is that they regard NAS as a black-box optimization problem in a discrete search strategy.

To address this issue, DAS [69] explores the possibility of transforming the discrete neural architecture space into a continuously differentiable form, and further uses gradient optimization techniques to search the neural architecture. This approach mainly focuses on the search of the hyperparameters of convolutional layers: filter sizes, number of channels, and grouped convolutions. MaskConnect [70] find that the existing cell-based neural architecture tends to adopt a predefined fixed connection method between modules; for example, each module only connects its first two modules [30], or connects all previous modules [36]. This connection method may not be optimal. Moreover, it uses the modified gradient method to explore the connection method between modules. In addition, other works [71, 72, 73] have also explored searching for neural architecture on continuous domains. However, the search for these neural architectures is limited to fine-tuning the specific structure of the network.

In order to solve the above challenges, DARTS [17] was developed. DARTS continuously relaxes the originally discrete search strategy, which makes it possible to use gradients to efficiently optimize the architecture search space. DARTS follows the cell-based search space of NASNet [32] and further normalizes it. Every cell is regarded as a ***directed acyclic graph*** (**DAG**), which is formed by sequentially connecting N nodes. Each of these cells has two input nodes and one output node. For convolutional cells, the input node is the output of the first two cells; for the recurrent cell, one is the input of the current step, while the other is the state of the previous step. The cell output is the concatenation result of all intermediate nodes. Each intermediate node $x^{(j)}$ in the cell is a potential feature representation, and is linked with each previous intermediate node $x^{(i)}$ in the cell through a directed edge operation $o^{(i,j)}$. For a discrete search strategy, each intermediate node can be expressed as follows:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}\left(x^{(i)}\right). \tag{2}$$

The DARTS approach makes the discrete search strategy continuous by relaxing the selection of candidate operations to a *softmax* of all possible operations. The mixed operation $\bar{o}^{(i,j)}(x)$ applied to feature map x can be expressed as follows:

$$\bar{o}^{(i,j)}(x) = \sum_{o\in O} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o'\in O}\exp\left(\alpha_{o'}^{(i,j)}\right)} o(x), \tag{3}$$

where O represents a set of candidate operations, while $\alpha_o^{(i,j)}$ represents the weight of operation o on directed edge $e^{(i,j)}$. Therefore, the NAS has evolved into an optimization process for a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$. Once the search is complete, the most likely operation $o^{(i,j)}$ on the directed edge $e^{(i,j)}$ is selected while other operations are discarded.

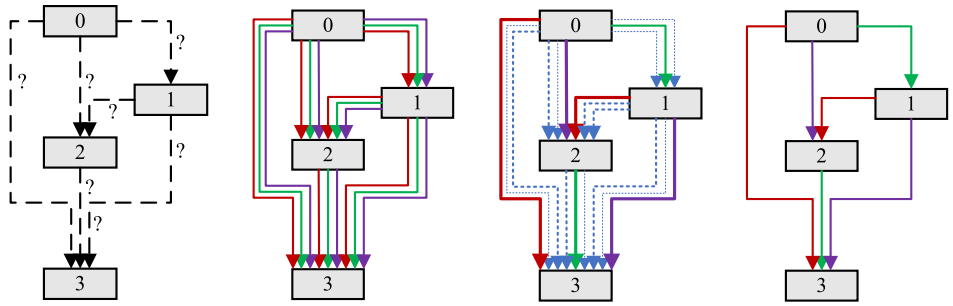
$$o^{(i,j)} = \text{argmax}_{o\in O}\, \alpha_o^{(i,j)}. \tag{4}$$

By solving a bi-level optimization problem [67, 68], the probability of mixed operations (the parameters α of the neural architecture) and network weights w can be jointly optimized as follows:

$$\begin{aligned} &\min_{\alpha}\ \mathcal{L}_{val}\left(w^*(\alpha), \alpha\right), \\ &\text{s.t.}\ \ w^*(\alpha) = \text{argmin}_w\, \mathcal{L}_{train}(w, \alpha), \end{aligned} \tag{5}$$

where $\mathcal{L}_{val}$ and $\mathcal{L}_{train}$ denote validation and training losses, respectively, while α is the upper-level variable and w is the lower-level variable. By jointly optimizing this problem, the optimal α is obtained, after which the discretization is performed to obtain the final neural architecture. We illustrate this process in Figure 6.

Compared with DARTS, the NAS process is changed from the selection of discrete candidate operations to the optimization of the probability of continuous mixed operations. During the same

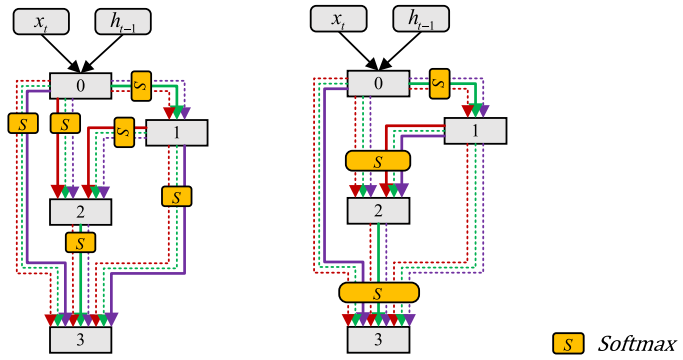


(a) Connection to be determined (b) Continuous relaxation (c) Joint optimization (d) Discrete neural architecture

Fig. 6. Continuous relaxation and discretization of search space in DARTS [17]. (a) A cell structure to be learned; the specific operation on the side is unknown. (b) Continuous relaxation of the cell-based search space, where each edge $e^{(i,j)}$ is a mixture of all candidate operations. (c) Joint optimization of the probability of mixed operations and network weights. (d) Discrete searched neural architecture.

period, NAO [74] opts to encode the entire neural architecture to map the originally discrete neural architecture to continuously embedded encoding. Subsequently, the output of the performance predictor is maximized by the gradient optimization method to enable the optimal embedded coding to be obtained. Finally, a decoder is used to discretize the optimal continuous representation (i.e., the optimal embedded coding) into the optimal neural architecture. Furthermore, DARTS uses the *argmax* strategy to eliminate the less probable operations among the mixed operations as a means of discretizing the neural architecture. However, common non-linear problems in network operation can introduce bias into the loss function; this bias exacerbates the performance difference between the derived child networks and the converged parent networks, which results in the need to retrain the parameters of the derived child networks. Therefore, a NAS solution with reduced performance deviation between the derived child networks and the converged parent networks is necessary. To this end, SNAS [46] begins with the delayed reward of reinforcement learning, then determines why delayed reward leads to the slow convergence speed of reinforcement learning when performing architecture search. Accordingly, SNAS proposes remodeling the NAS to theoretically bypass the delayed reward problem for reinforcement learning, while simultaneously ensuring that neural architecture parameters are continuous, so those network operation parameters and neural architecture parameters can be jointly optimized using a gradient method. Based on this, SNAS has a more efficient and automated NAS framework that still maintains the completeness and differentiability of the NAS pipeline.

In works about SNAS, DARTS, and many other NAS [76, 77, 78, 79], the feasible paths of the searched neural architecture depend on each other and are closely coupled during the search phase. While SNAS does to some extent reduce the performance difference between the derived child network and the converged parent network, SNAS and DARTS are still required to choose only one path during the verification phase. This crude decoupling inevitably leads to a gap between neural architectures during the search and verification phases. To address this issue, DATA [75] developed the ***Ensemble Gumbel-Softmax*** (EGS) estimator, which can decouple the relationship between different paths of the neural architecture and achieve the seamless transfer of gradients between different paths. This solves the problem of the architecture's inability to be seamlessly connected between search and verification.



(a) DARTS cell (b) I-DARTS cell

Fig. 7. Comparison of the cell structure of DARTS [17] and I-DARTS [80] in a recurrent neural network. (a) In DARTS, edges from different nodes cannot be compared. Moreover, when discretizing the neural architecture, there is only one correlation edge between each pair of nodes. (b) In I-DARTS, a given node uses a *softmax* while considering all input edges. Given a specific node, I-DARTS can determine whether edges are connecting related nodes according to the importance of all input edges: either there are multiple connected edges or no related edges.

Furthermore, I-DARTS [80] notes that the *softmax*-based relaxation constraint between each pair of nodes may cause DARTS to be a "local" model. In DARTS, the cell's middle node is connected to all the precursor nodes, and when discretizing the neural architecture, there is an only one-directional edge between each pair of nodes. This results in the existence of edges from different nodes that are unable to be compared with each other. In addition, DARTS requires only one directed edge between each pair of nodes; this constraint design has no theoretical basis and also limits the size of the DARTS search space. These local decisions, which are caused by the bias problem in graph-based models [81, 82] compel DARTS unable to make the best choice of architecture. Based on this, I-DARTS proposes an interesting and simple idea: namely, using a *softmax* to simultaneously consider all input edges of a given node. We present the cell structure comparison of DARTS and I-DARTS in the recurrent neural network in Figure 7. As can be seen from Figure 7(b), when given a node, I-DARTS can determine whether edges are connecting related nodes depending on the importance of all input edges: there are either multiple connected edges or no related edges.

P-DARTS [44] starts with the deep gap between the search and evaluation of neural architecture, and improves DARTS overall. In DARTS, due to limitations on computational resources, DARTS uses a shallow cell stack architecture in the search phase; in the evaluation phase, moreover, it stacks more searched cells to enable the processing of datasets with higher resolution. Therefore, the basic cell of the neural architecture used for evaluation is designed for shallow architecture, which differs from the deep neural architecture used in the evaluation stage. Based on this, P-DARTS uses progressive search to gradually increase the depth of the network during the search phase. Moreover, it also gradually reduces the candidate operation set according to the weights of the mixed operation in the search process, to cope with the problem of the increase in the calculation volume caused by the increase in depth. At the same time, P-DARTS proposes regularization of the search space to deal with the problem of insufficient stability during the process of searching in deep architectures (algorithms are heavily biased toward skip-connect).

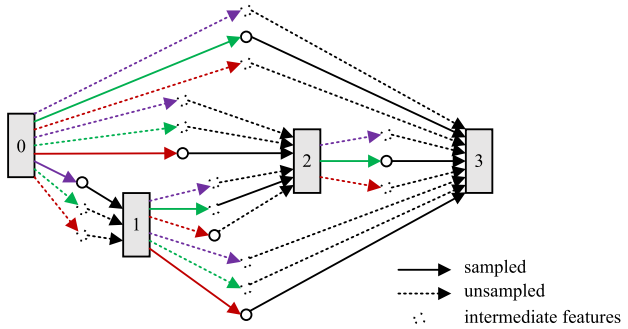


Fig. 8. In GDAS [83], an example of sampling subgraphs in a DAG with three intermediate nodes. Colored lines indicate operations in the candidate operation set, while black lines indicate corresponding information flows. The circles indicate the intermediate features following operation processing. Each intermediate node is equal to the sum of all the sampled intermediate features. In one iteration, only one sampled subgraph is trained.

Compared with NAS based on RL and EA, DARTS greatly improves the search efficiency, but to an insufficient extent. As shown in Figure 6(c), in the search phase, DARTS continuously relaxes the cell's search space and optimizes all parameters in the DAG simultaneously. This causes DARTS to occupy too much memory on the device during searching, which slows down the search speed. At the same time, the effects of different operations between the same pair of nodes may be canceled by each other, thereby destroying the entire optimization process. To this end, GDAS [83] proposes to use a differentiable architecture sampler in each training iteration to sample only one subgraph in the DAG, meaning that only one part of the DAG needs to be optimized in any one iteration. We illustrate this process in Figure 8. At the same time, in the verification stage, the architecture sampler can be optimized by using the gradient-based method, thereby further improving the search efficiency of GDAS.

To reduce DARTS's memory usage during search and improve the search efficiency, PC-DARTS [84] opts to start from the channel, as opposed to GDAS' sampling subgraphs in DAG and training only one subgraph in one iteration. During the search process, PC-DARTS samples the channels and convolves only the sampled channel features to achieve efficient gradient optimization. To deal with the problem of inconsistent information brought about by the channel sampling strategy, PC-DARTS uses edge normalization to solve this problem. It reduces the uncertainty in the search process by adding a set of edge-level parameters. As a result, PC-DARTS can save memory and is more efficient and stable. [111] recently find that DARTS [17] exhibits poor test performance for architecture generated in a wide search space. This work contends that when the discovered solutions are consistent with the high verification loss curvature in the architecture space, the discovered architecture is difficult to promote. Moreover, various types of regularization are added to explore how to make DARTS more robust. Finally, [111] proposes several simple variants and achieves good generalization performance. Although we have conducted many reviews, there are still many improvements that have been made based on DARTS [115, 153].

In the above-mentioned gradient-based methods, local optimization is a common problem. Therefore, we conduct a comprehensive review of the solution to this problem here. The experimental results of DARTS [17] show that an appropriate learning rate helps the model converge to a better local optimal value. As shown in Figure 7(b), I-DARTS [80] relaxes the softmax-based relaxation on each edge of DARTS to all incoming edges of a given node, thereby alleviating the impact of bias caused by local decision-making. PC-DARTS [84] uses channel sampling to replace

the convolution operation on all channels in DARTS, thereby reducing the possibility of falling into a local optimum. In general, local optimization is still an important challenge faced by gradient-based optimization methods, so more related research is needed in the future.

In this section, we provide a comprehensive and systematic overview of optimized NAS work that employs gradient strategies on a continuous search strategy. Due to the simplicity and elegance of the DARTS architecture, the research work related to DARTS is quite rich. Moreover, gradient optimization in a continuous search strategy is an important trend of NAS.

3.3 Neural Architecture Recycling

Early NAS works [11, 12, 15, 16] and many subsequent works [17, 39, 40, 41] aim to search the neural architecture from scratch. From a certain perspective, this type of approach does increase the freedom of neural architecture design, and it is very likely to result in the design of a new high-performance network structure unknown to humans. However, it is clear that this idea also increases the time complexity of searching for the best neural architecture; this is because it does not make full use of the prior knowledge regarding the existing artificially designed high-performance neural architecture. Therefore, a new idea would be to use the existing, artificially designed high-performance neural architecture as a starting point, then use the NAS method to modify or evolve these neural architectures, as this would enable a more promising neural architecture to be obtained at a lower computing cost. This process is generally referred to as "network transformation."

Net2Net [47] conducts a detailed study of network transformation technology and proposes function-preserving transformations to facilitate the reuse of model parameters after transformation. This approach can effectively accelerate the training of new and larger networks. Based on this idea, [51] proposes ***efficient architecture search*** (**EAS**), which uses the encoder network as a meta-controller to learn the low-dimensional representation of the existing neural architecture, and further refers to the multiple actor networks in Net2Net [47] to decide whether to make corresponding adjustments to the neural architecture at the layer level (deepening or widening layer). In addition, this approach uses reinforcement learning strategies to update the parameters in the meta-controller. EAS takes the view that the network transformation at the layer level needs to combine the information of the entire neural architecture; thus, a *bidirectional recurrent network* (Bi-LSTM) [48] is used as the network encoder. Since EAS is a network transformation on an existing network, models and weights can be reused to substantially reduce the amount of calculation required. We illustrate the overall neural architecture of EAS in Figure 9. In Figure 10, we also present the internal structure of two actor networks: Net2Wider and Net2Deeper. In Net2Wider, the actor network shares the same *sigmoid* classifier and decides whether to widen the layer according to each hidden encoder state. In Net2Deeper, the actor network inputs the state of the final hidden Bi-LSTM layer into the recurrent network, after which the recurrent network decides both where to insert the layer and the parameters of the inserted layer.

Rather than widening or deepening the layers of the existing network in EAS [51], N2N learning [52] compresses the teacher network by removing or shrinking the layers. In more detail, it compresses the teacher network through a two-stage operation selection: first, the layer removal is performed on the macro level, after which the layer shrinkage is performed on the micro-level. Reinforcement learning is used to explore the search space, while knowledge distillation [56] is used to train each generated neural architecture. In the next step, a locally optimal student network is learned. Using this method, under similar performance conditions, a compression ratio of more than 10× is achieved for networks such as ResNet-34 [30]. Moreover, unlike EAS [51] and N2N learning [52], which can only deepen (remove) and widen (shrink) the network at the layer level, Path-level EAS [57] realizes a network transformation at the path level. The inspiration behind

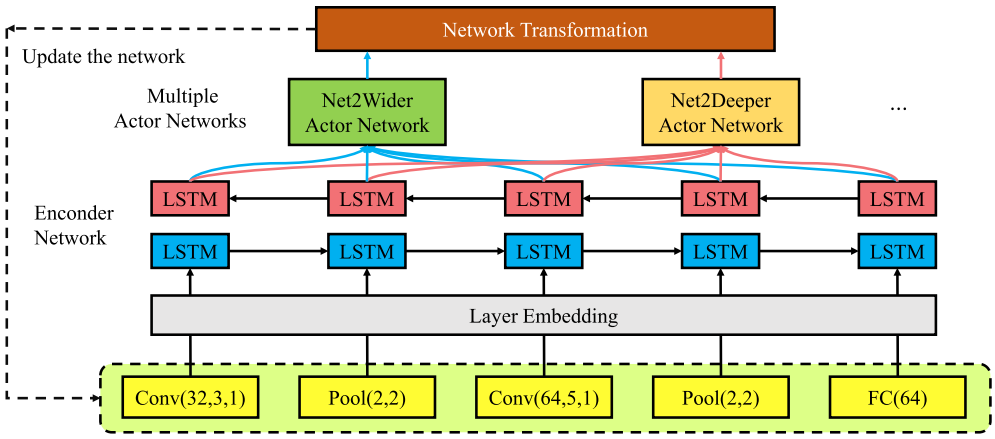


Fig. 9. In EAS [51], architecture search based on network transformation. After the existing network layer is encoded by the layer encoder, Bi-LSTM [48] is used as a meta-controller to learn the low-dimensional feature representation of the neural architecture. The multi-actor network combines these features to decide whether the corresponding network transformation operation (deepening layer or widening layer) should be adopted. Finally, reinforcement learning is used to update the meta-controlled parameters.

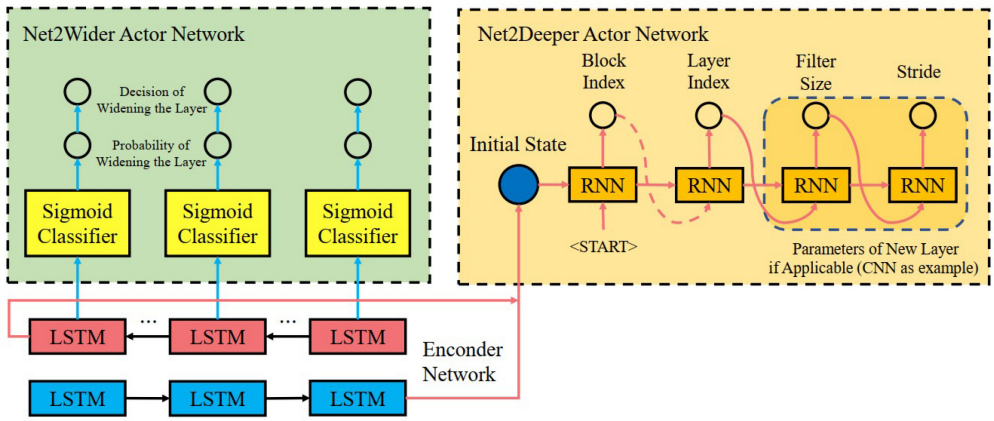


Fig. 10. The internal structure of two actor networks in EAS [51]: Net2Wider and Net2Deeper. In Net2Wider, the actor network shares the same sigmoid classifier and decides whether to widen the layer according to each hidden state of the encoder. In Net2Deeper, the actor network inputs the state of the final hidden layer of Bi-LSTM into the recurrent network, after which the recurrent network decides where to insert the layer and the parameters of the inserted layer.

this concept stems from the performance gains achieved by the multi-branch neural architecture included in the manually designed network [30, 31, 58, 59], which achieves network path-level transformation by replacing a single layer with multi-branch operations incorporating allocation and merge strategies. Allocation strategies include *replication* and *split*, while merge strategies include *add* and *concatenation*. We present an example of the process of implementing a path-level network transformation by using a multi-branch operation rather than a single layer in Figure 11. Another similar work, NASH-Net [85], further proposes four network morphism types based on Net2Net [47]. NASH-Net can begin with a pre-trained network, apply network morphism to gen-

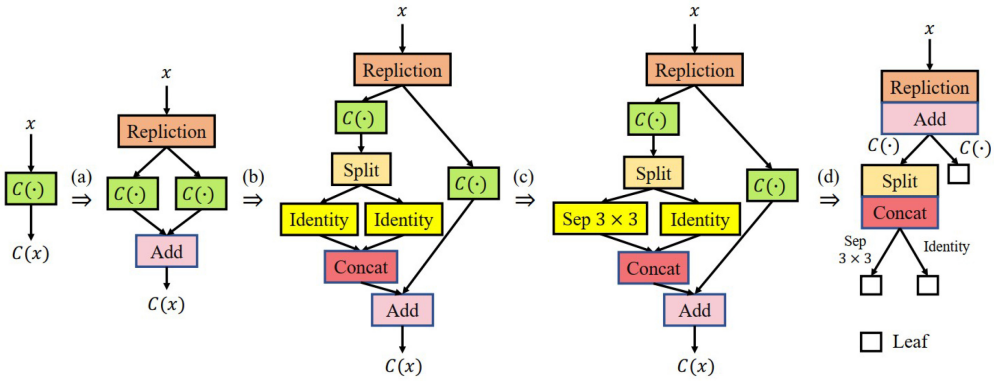


Fig. 11. In Path-level EAS [57], an example of a process for implementing path-level network transformation using a multi-branch operation rather than a single layer. (a) Using the *replication-add* strategy to add branches to a single layer. (b) Using the *split-concat* strategy to further add branches to the network. (c) Replacing identity mapping with a 3×3 depthwise-separable convolution. (d) The tree structure of the neural architecture in (c).

erate a set of sub-networks, and obtain the best sub-network following a short period of training. Then, after starting from the best sub-network, this process is iterated using the ***Neural Architecture Search by Hill-climbing*** (**NASH**) to get the best neural architecture.

For complex tasks such as semantic segmentation or object detection, previous works have often used networks designed for image classification, such as the backbone network. Under these circumstances, performance gains can be obtained by specifically designing networks for complex target tasks. Although some works [63, 65, 66] have used NAS to design backbone networks for semantic segmentation or object detection tasks, pre-training is still necessary and the computational cost is high. ***Fast Neural Network Adaptation*** (**FNA**) [60] proposes a method that can adapt a network's architecture and parameters to new tasks at almost zero cost. It starts from a seed network (a manually designed high-performance network), expands it into a super network in its operation set, and then uses the NAS method [17, 19, 43] to adapt the neural architecture in a way that allows it to obtain the target architecture. Moreover, it uses the seed network to map the parameters to the super network and the target network to initialize the parameters. Finally, the target network is obtained by fine-tuning the target task. We illustrate this process in Figure 12. It is precisely due to the low cost of FNA in network transformation that NAS can design a special neural architecture for large-scale tasks, such as detection and segmentation.

Different from the above methods which mainly focus on using the NAS method to improve the visual model, Evolved Transformer [155] is committed to using the NAS method to design a better feedforward architecture for seq2seq tasks. Specifically, Evolved Transformer first constructed a large search space and then ran an evolutionary architecture search with warm starting by seeding our initial population with the Transformer, thereby searching for better alternatives to the Transformer. In addition, to be able to dynamically allocate more computing resources to more promising candidate networks, Evolved Transformer also developed the Progressive Dynamic Hurdles method and achieved continuous improvement on four well-established language tasks.

In this section, we present a comprehensive overview of NAS based on architecture recycling. This method makes it possible to utilize a large number of high-performance networks that are previously artificially designed. This saves the NAS from having to search the neural architecture

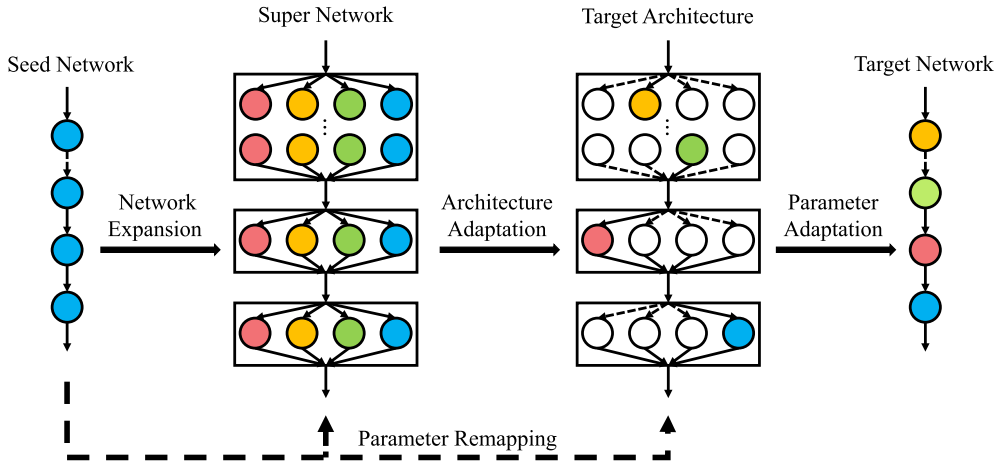


Fig. 12. The overall framework of FNA [60]. It starts from a seed network, expands this network into a super network in its operation set, and then uses the NAS method to adapt the neural architecture to obtain the target architecture. The seed network is then used to map the parameters to the super network and the target network to initialize the parameters. Finally, the target network is obtained via fine-tuning the target task.

from scratch, which substantially reduces the number of unnecessary random searches required in the massive search space. Compared with other optimization strategies, there are relatively few studies on NAS based on neural architecture recycling.

3.4 Incomplete Training

The key technology behind NAS involves using a search strategy to find the best neural architecture by comparing the performance of a large number of candidate neural architectures. Accordingly, the performance ranking of candidate neural architectures is extremely important. Early versions of NAS [11, 12, 15, 16] usually fully train the candidate neural architecture, then obtain the rankings of the candidate neural architectures based on their performance on the validation set. However, this method is excessively time-consuming because there are excessively many candidate neural architectures to compare.

It should, however, be noted here that these works have also used some methods to accelerate the ranking of candidate neural architectures. For example, NAS-RL [11] uses parallel and asynchronous updates [42] to accelerate candidate neural architecture training. MetaQNN [12] compares the performance of the candidate neural architecture after the first epoch of training with the performance of the random predictor to determine whether it is necessary to reduce the learning rate and restart training. Large-scale Evolution [15] allows the mutated child neural architecture to inherit the weight of the parent to the greatest possible extent, thereby reducing the burden associated with retraining the candidate neural architecture. However, there are still a large number of child networks whose structural changes are unable to inherit the weight of their parents after mutation, meaning that these candidate networks will be forced to retrain. Although the above methods also accelerate the training of candidate neural architectures to a certain extent, they still require a lot of computing power and their acceleration effects are relatively limited. Therefore, it is necessary to conduct some research that can be used to further accelerate the training of candidate neural architectures to obtain their relative ranking.

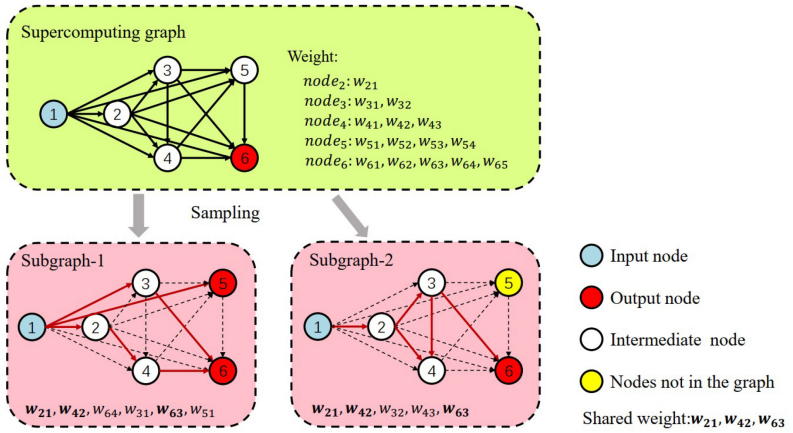


Fig. 13. Parameter sharing mechanism in ENAS [19]. The candidate neural architecture in NAS can be represented as a directed acyclic subgraph that is sampled in a supercomputing graph constructed by the search space. The nodes in the graph represent local calculations, while the edges represent information flow. Each node has its corresponding weight parameter, as indicated in the upper right. Only when a specific edge is sampled will the corresponding parameter be activated and updated. The ENAS mechanism allows all subgraphs (i.e., candidate neural architectures) to share parameters.

3.4.1 Training from Scratch? Can we only train each candidate's neural architecture from scratch? This may ignore the interconnection between neural architectures. Parameter sharing brings us a new possibility.

When treating the candidate neural architecture as an independent individual, each candidate neural architecture is trained from scratch, after which the candidate neural architecture is ranked according to their performance on the validation set. This may provide a more accurate ranking, as has occurred in other works [11, 32, 34, 37]. In this process, the parameters of each trained candidate neural architecture are directly discarded. This does not result in full utilization of these trained parameters; accordingly, a new idea of parameter sharing has emerged.

ENAS [19] is the first NAS work to explicitly propose parameter sharing. The ENAS work has noted that the candidate neural architecture in NAS can be regarded as a directed acyclic subgraph, which is in a supercomputing graph constructed by the search space. We illustrate this sampling process in Figure 13. Based on this observation, ENAS uses LSTM as a controller for use in searching the optimal subgraph on a large computation graph to obtain the neural architecture. In transfer learning and multi-task learning, the weights obtained by training a model designed for a specific task on a dataset are also applicable to other models designed for other tasks [86, 87, 88]. Encouraged by this, ENAS proposes forcing the sharing of parameters among all different child models (candidate architecture). Through the use of this mechanism, the child models can obtain empirical performance, thereby avoiding the need to completely train each child model from scratch. We present an example of different subgraphs sharing weights in Figure 13. The supercomputing graph can be expressed as a DAG: the nodes in the graph are defined as local calculations, while the edges represent the flow of information. Each node has its corresponding weight parameter, as illustrated in the upper right of Figure 13. However, the corresponding parameters can only be activated when a specific edge is sampled. The ENAS mechanism allows all subgraphs (i.e., candidate neural architectures) to share parameters. Therefore, EANS has greatly improved search efficiency compared to [11, 32, 34, 37].

Subsequently, CAS [55] explored a multi-task architecture search based on ENAS. This approach extends NAS to transfer learning across data sources, and further introduces a novel continuous architecture search to solve this "forgetting" problem in the continuous learning process. This enables CAS to inherit the experience gained from the previous task when training a new task, thereby allowing the model parameters to be continuously trained. This is highly beneficial to NAS research on multi-tasking. Moreover, AutoGAN [54] first introduced NAS into ***generative adversarial networks*** (**GANs**) [93] and used the ***Inception score*** (**IS**) [94] as the reward value of RL to accelerate the search process through parameter sharing ENAS [19] and dynamic-resetting. Further, progressive GAN training [95] has been used to introduce ***multi-level architecture search*** (**MLAS**) into AutoGAN and gradually implement NAS. Compared with the most advanced manual GANs [95, 96, 97, 98], AutoGAN is highly competitive. The parameter sharing mechanism is also used to accelerate the deployment research of the NAS architecture model across multiple devices and multiple constrained environments. At the kernel level, OFA [100] uses an elastic kernel mechanism to meet the application needs of multi-platform deployment and the diverse visual needs of different platforms. Small kernels share the weight of large kernels; this is done to avoid repeatedly centering sub-kernels (centering sub-kernels is used for both an independent kernel and a part of a large kernel) to reduce the performance of certain sub-networks. OFA also introduces a kernel transformation matrix. At the network level, OFA recommends training the largest network first, while the smaller network shares the weight of the larger network before fine-tuning. Moreover, the weight of the large network can provide a small network with good initialization, which greatly accelerates the training efficiency.

In addition, the one-shot–based method also employs parameter sharing. SMASH [23] proposes to train an auxiliary HyperNet [89], which is used to generate weights for other candidate neural architectures. In addition, SMASH also uses the early training performance of different networks derived from the research in Hyperband [90] to provide meaningful guidance suggestions for the ranking of candidate neural architectures. Parameter sharing is primarily reflected in the hypernetwork and between candidate neural architectures. The use of the auxiliary HyperNet avoids the need to completely train each candidate neural architecture. By comparing the performance of the candidate neural architectures using weights generated by the HyperNet on the verification set, their relative rankings can be obtained; this allows SMASH to quickly obtain the optimal neural architecture at the cost of a single training session. Understanding One-Shot Models [22] conducts a comprehensive analysis of the rationality of the parameter sharing approaches used in SMASH [23] and ENAS [19]. In addition, Understanding One-Shot Models also discussed the necessity of the hypernetwork in SMASH and the RL controller in ENAS, pointing out that a good enough result can be obtained without the use of the hypernetwork and RL controller. Unlike SMASH, which encodes an architecture into a three-dimensional tensor via a memory channel scheme, **Graph HyperNetwork** (**GHN**) [91] employs computation graphs to represent the neural architecture, then uses graph neural networks to perform architecture searches. Compared to SMASH, which can only use the hypernetwork to predict some weights, GHN can predict all the free weights through the use of a graph model. Therefore, network topology modeling-based GHN can predict network performance faster and more accurately than SMASH.

A typical one-shot NAS must randomly sample a large number of candidate architectures from the hypernetwork using parameter sharing, then evaluate these architectures to find the best one [22, 23]. SETN [92] noted that finding the best architecture from these sampled candidate architectures is extremely difficult. This is because, in the relevant NAS [22, 23, 83], the shared parameters are closely coupled with the learnable architectural parameters. This will introduce deviations into the template parameters, which will cause some of the learnable architectural parameters to be more biased toward simple networks (these networks have fewer layers and are

more lightweight). Because they converge faster than more complex networks, this will lead to a simplified search architecture. At the same time, it also results in the sampled candidate architectures having a very low good rate. To this end, SETN adopts a uniformly stochastic training strategy to treat each candidate architecture fairly, meaning that they are fully trained to obtain more accurate verification performance. In addition, SETN is also equipped with a template architecture estimator. Unlike the random sampling methods previously used in Understanding One-Shot Models [22] and SMASH [23], the estimator in SETN can be used to determine the probability that the candidate architecture has a lower verification loss, as well as to ensure that the low verification loss architecture with a higher probability will be selected for one-shot evaluation. At the same time, the estimator is trained on the validation set. Therefore, SETN improves the excellent rate of the sampling candidate architecture compared to Understanding One-Shot Models [22] and SMASH [23], making it more likely to find the optimal architecture.

[24], through evaluating the effectiveness of the NAS search strategy, find that the weight sharing strategy in ENAS [19] resulted in inaccurate performance evaluation of the candidate architecture, making it difficult for the NAS to identify the best architecture. In addition, the research of Fairnas [101] and [102] also demonstrates that candidate neural architectures based on these parameter sharing methods also cannot be adequately trained, which lead to an inaccurate ranking of candidate neural architectures. In NAS works based on gradient optimization [17, 103, 104], the joint optimization of supernet weights and architectural parameters also introduces bias between sub-models. In light of this, DNA [21] proposes to modularize the NAS's large-scale search space to ensure that the candidate architecture is adequately trained to reduce the representation shift caused by parameter sharing. In addition, DNA [21] also uses block-wise search to evaluate all candidate architectures within the block. These methods are used to evaluate candidate architectures more accurately. GDAS-NSAS [25] also considered and improved the weight sharing mechanism in one-shot NAS, proposing an NSAS loss function to solve the problem of multi-model forgetting (when weight sharing is used to sequentially train a new neural architecture, the performance of the previous neural architecture is reduced) that arises due to weight sharing during the super network training process. Finally, GDAS-NSAS [25] applies the proposed method to RandomNAS [105] and GDAS [83]; this approach effectively suppresses the multi-model forgetting problem and consequently improves the training quality of the supernet,

Differentiable NAS also employs similar parameter sharing ideas. Examples include DARTS-like work [17, 44, 80, 84]; for details, refer to Section 3.2. In ENAS, a controller is used to sample subgraphs in a supercomputing graph. Subgraphs with the same information flow share parameters in the search phrase, so only the sampled subgraphs need to be optimized in each iteration. The difference is that the DARTS-like method chooses to optimize a super network directly, and the best sub-network is decoupled from the super network according to the learned mixed operation weights. Parameters are shared among different sub-networks in the super network. Moreover, optimization strategies based on neural architecture recycling can often be initialized with the help of function preservation [47] to inherit the parameters of the template network, thereby avoiding the retraining of sub-neural architecture. More detailed content can be found in Section 3.3, for example, EAS [51], Path-level EAS [57], N2N learning [52], and so forth.

3.4.2 Training to Convergence? Can we only train each candidate neural architecture to convergence? This may ignore the guiding role of early performance curves in predicting the potential of neural architecture. Early termination brings us a new possibility.

To quickly analyze the effectiveness of a specific model in deep learning, human experts often judge whether this model is necessary to continue training according to the model's learning curve. Therefore, the training of models with no potential will be terminated as soon as possible;

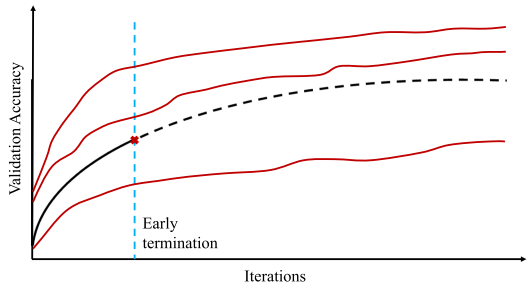


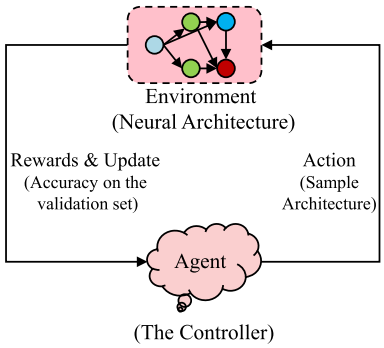
Fig. 14. Assuming learning curve for performance prediction. The black curve represents an attempt to use an observable early performance learning curve (solid line) to predict the neural architecture's final performance on the validation set [20, 37, 110]. The three red curves indicate the hypothesis that the performance of the neural architecture during early training and convergence is consistent with ranking [50]. Using performance prediction and early performance ranking assumptions can facilitate the termination of the training of neural architectures that lack potential as early as human experts, as well as the use of minimal computing resources to explore more potential neural architectures. This speeds up the NAS architecture search process.

this means that there is no need to wait for this model to converge, which saves resources for new exploration. Similar strategies can also be used to rank the performance of NAS candidate architectures: for candidate architectures with no potential, training can be terminated early, while more adequate training can be obtained for more promising architectures.

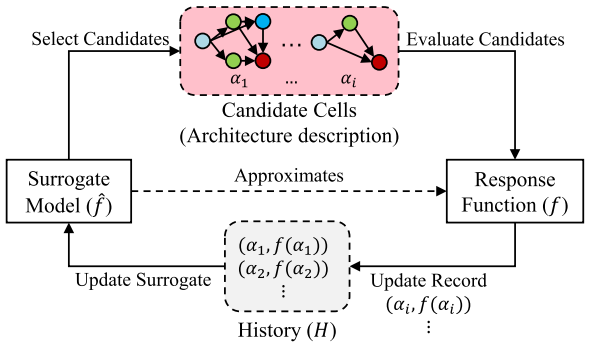
Early termination of training is not a new idea; many researchers have done a large amount of related research on this subject. For example, [107] uses the probabilistic method to simulate the learning curve of the deep neural network and terminates the training of poorly running models in advance. However, this approach requires a long early training period to accurately simulate and predict the learning curve. [108] extends [107]; in [108], the probability model of the learning curve can be set across hyperparameters, and a mature learning curve is used to improve the performance of the Bayesian neural network. Similar strategies have also been used to solve hyperparameter optimization problems [90, 109].

The above methods are based on the use of partially observed early performance to predict the learning curve and design the corresponding machine learning model. To imitate human experts to the extent that NAS search can also automatically identify below-standard candidate architectures and terminate their training early, [20] combines learning curve prediction with NAS tasks for the first time. This approach builds a set of standard frequentist regression models and obtains the corresponding simple features from the neural architecture, hyperparameters, and early learning curve. These features are then used to train the frequentist regression model and are then used to predict the final verification set performance of the neural architecture with early training experience. Performance prediction is also used in **Progressive Neural Architecture Search (PNAS)** [37]. To avoid the need to train and evaluate all child networks, it learns a predictor function that can be trained based on the observable early performance of the cell. The predictor is used to evaluate all candidate cells, then the top-k cells are selected, and this process is repeated until a sufficient number of blocks of cells are found. The black curve in Figure 14 shows the learning curve prediction trying to predict the final performance from the premature learning curve.

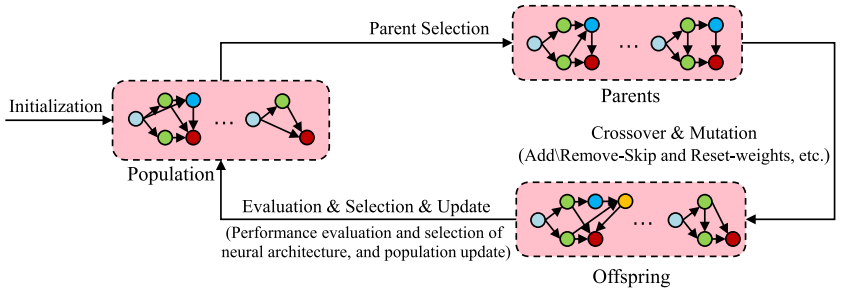
NAO [74] uses a performance predictor similar to previous work [20, 37, 110]. This is dissimilar to PNAS [37], which uses a performance predictor to evaluate and select the generated neural architecture to speed up the search process. In NAO [74], after the encoder completes the continuous representation of the neural architecture, the performance predictor is taken as the optimization goal of the gradient ascent. By maximizing the output of the performance predictor f, the



(a) The general framework of RL in the context of NAS.



(b) The general framework of SMBO in the context of NAS.



(c) The general framework of EA in the context of NAS.

Fig. 15. Comparison of RL, SMBO, and EA general frameworks in the context of NAS.

continuous representation of the best neural architecture can be obtained. Finally, the decoder is used to get the final discrete neural architecture. Unlike previous NAS based on performance prediction [20, 37, 110], in multinomial distribution learning for NAS, MdeNAS [50] put forward a performance ranking hypothesis; that is, the relative performance ranking of the neural architecture at each training stage is consistent. In other words, the neural architecture that performed well in the early days still maintains good performance when the training converges. MdeNAS [50] has conducted a large number of experiments to verify this hypothesis; according to these results, the early performance of candidate architectures can be used to quickly and easily obtain their relative performance rankings, thereby speeding up the NAS process. We illustrate an early performance ranking hypothesis in Figure 14 (as shown by the three red curves).

In this section, we focus on the challenge of fully trained candidate architectures, starting with the necessity of two aspects (training from scratch and training to convergence), and comprehensively and systematically summarizing the existing work. Compared with other optimization strategies, this aspect of the research work is relatively small, but still very necessary.

4 PERFORMANCE COMPARISON

NAS is a highly promising study. In this section, we classify and compare the performance of existing NAS based on mainstream search methods [27, 28], while also reporting the optimization strategies they use according to Section 3. These search methods primarily include the following: RL, EA, ***gradient optimization*** (**GO**), ***random search*** (**RS**), and ***sequential model-based optimization*** (**SMBO**) [162]. We summarized in Figure 15 the general framework comparison of RL,

SMBO, and EA in the context of NAS. Figure 7(a) shows the general framework of RL in the context of NAS. The agent (controller) makes corresponding actions (sampling neural framework) to modify the system state (neural architecture), and the environment gives the agent corresponding rewards (accuracy on the verification set) and updates the agent accordingly (adjusting algorithm parameters). Figure 7(b) shows the general framework of SMBO in the context of NAS. The surrogate model $\hat{f}$ (performance predictor) evaluates all candidate cells α (the structure description of the neural architecture) [37] and selects promising candidate cells, and then evaluates the performance of the neural architecture composed of the candidate cells on the validation set and gets the response value $f(\alpha)$ (loss value) of the corresponding response function f [161]. The cell α_i and the corresponding response value $f(\alpha_i)$ are added as a new meta instance $(\alpha_i, f(\alpha_i))$ to the history record H of cell measurement performance, and the surrogate model is updated again according to H until convergence. Through iteration, the surrogate model value $\hat{f}(\alpha)$ of the cell α is continuously approximated to the corresponding response value $f(\alpha)$ to avoid time-consuming training steps. Figure 7(c) shows the general framework of EA in the context of NAS. The algorithm first initializes the population and selects parents, then crosses or mutates based on the parent to generate a new individual offspring (neural architecture) and evaluate the adaptability of the offspring (the performance evaluation of the new neural architecture), and select a group of individuals with the strongest adaptability (the best performing neural architecture) to update the population. GO has been described in detail in Figure 6, so it is no longer repeated here.

We intend to obtain their similarities and differences from these summaries. The situation is not as simple as it might appear. In fact, it is relatively difficult to compare NAS performance, because NAS lacks certain baselines. In addition, different types of NAS have great differences in preprocessing, hyperparameters, search space, and trick, which increase the difficulty of NAS performance comparison. These tricks include learning rate decay, regularization (e.g., DropPath [32, 136]), augmenting techniques (e.g., Cutout [131]), and so forth. The random search strategy is considered to be a strong baseline. For example, in [24], the random search identifies the best RNN cells compared to other strategies. The findings presented in [129], [34], and [105] also prove this. Therefore, [129] uses the average architecture of the random search strategy as the baseline for comparison.

The performance of the state-of-the-art NAS and mainstream artificial networks on the CIFAR-10 dataset are summarized in Table 2. Moreover, based on Section 3, we simultaneously report the optimization strategy used in NAS. Similarly, we present a performance comparison on the ImageNet dataset in Table 3. Because the optimization strategies used in the same NAS method are identical, we therefore omit the reporting of the corresponding optimization strategies in Table 3.

From observing Tables 2 and 3, we can clearly see that in popular NAS, the use of modular search strategies is highly extensive. This is mainly because the modular search greatly reduces the complexity of the search space compared to a global search. However, as summarized in Section 3.1, this does not necessarily imply that the modular search space is better than the global search. Moreover, we can conclude with certainty that incomplete training is also widely used; this can effectively accelerate the ranking of candidate neural architectures, thereby reducing the search duration. In addition, among the many optimization strategies considered, the gradient optimization based on the continuous search strategy can most significantly reduce the search cost and has a very rich body of research to support it. We also observe that random search strategy-based NAS has also achieved highly competitive performance. However, it is clear that existing NAS research on random search strategies is relatively inadequate. The optimization strategy of architecture recycling also exhibits comparatively excellent performance, but there are relatively few related studies. On the other hand, transfer learning is a widely used technique in NAS tasks; that is, the architecture searched on a small dataset (CIFAR-10) will be transferred to a large dataset

Table 2. The Performance Comparison between the State-of-the-Art NAS and Mainstream Artificial Networks on CIFAR-10

Search method	Reference	Venue	Optimization Strategy				Error Acc (%)	Params (Millions)	GPU Days
			Modular search space	Continuous search strategy	Architecture recycle	Incomplete training			
Human	WRN [133]	CVPR16					3.87	36.2	-
	Shark [134]	CoRR17					3.55	2.9	-
	PyramidSepDrop [135]	CoRR16					2.67	26.2	-
	ResNet [136]	ECCV16					6.41	1.7	-
	Fractalnet [137]	ICLR17					5.22	38.6	-
	DenseNet-BC [36]	CVPR17					3.46	25.6	-
RL	NAS-RL [11]	ICLR17					3.65	37.4	22,400
	MetaQNN [12]	ICLR17					6.92	11.2	100
	EAS [51]	AAAI18			✓	✓	4.23	23.4	10
	NASNet-A [32]	CVPR18	✓				3.41	3.3	2,000
	NASNet-A + Cutout [32]	CVPR18	✓				2.65	3.3	2,000
	Block-QNN [33]	CVPR18	✓				3.54	39.8	96
	Path-level EAS [57]	ICML18			✓	✓	2.99	5.7	200
	Path-level EAS + Cutout [57]	ICML18			✓	✓	2.49	5.7	200
	N2N learning [52]	ICLR18			✓	✓	6.46	3.9	2.1
	ProxylessNAS-R + Cutout [104]	ICLR19				✓	2.30	5.8	N/A
	FPNAS + Cutout [39]	ICCV19	✓			✓	3.01	5.8	0.8
EA	Large-scale Evolution [15]	ICML17				✓	5.40	5.4	2,600
	GeNet [16]	ICCV17					5.39	N/A	17
	Genetic Programming CNN [5]	GECC17					5.98	1.7	14.9
	Hierarchical-EAS [34]	ICLR18	✓				3.75	15.7	300
	NASH-Net [85]	ICLR18			✓	✓	5.20	19.7	1
	Neuro-Cell-based Evolution [130]	ECML-KDD18	✓		✓	✓	3.57	5.8	0.5
	AmoebaNet-A [43]	AAAI19	✓				3.34	3.2	3,150
	Single-Path One-Shot NAS [106]	CoRR19	✓			✓	N/A	N/A	N/A
	ENAS + micro [19]	ICML18	✓	✓		✓	3.54	4.6	0.5
	ENAS + micro + Cutout [19]	ICML18	✓	✓		✓	3.54	4.6	0.5
	ENAS + macro [19]	ICML18		✓		✓	4.23	21.3	0.32
	SMASH [23]	ICLR18		✓		✓	4.03	16	1.5
	Understanding One-Shot Models [22]	ICML18	✓	✓		✓	4.00	5.0	N/A
	Maskconnect [70]	ECCV18	✓	✓		✓	3.27	N/A	N/A
	DARTS (1st order) + Cutout [17]	ICLR19	✓	✓		✓	3.00	3.3	1.5
	DARTS (2nd order) + Cutout [17]	ICLR19	✓	✓		✓	2.76	3.3	4
	SNAS + Cutout [46]	ICLR19	✓	✓		✓	2.85	2.8	1.5
	GHN [91]	ICLR19	✓	✓		✓	2.84	5.7	0.84

(Continued)

Table 2. Continued

Search method	Reference	Venue	Optimization Strategy				Error Acc (%)	Params (Millions)	GPU Days
			Modular search space	Continuous search strategy	Architecture recycle	Incomplete training			
GO	ProxylessNAS-G + Cutout [104]	ICLR19		✓		✓	2.08	5.7	N/A
	PARSEC + Cutout [132]	CoRR19	✓	✓		✓	2.81	3.7	1
	BayesNAS [138]	ICML19	✓	✓		✓	2.81	3.4	0.2
	P-DARTS + Cutout [44]	ICCV19	✓	✓		✓	2.50	3.4	0.3
	SETN [92]	ICCV19	✓	✓		✓	2.69	N/A	1.8
	DATA + Cutout [75]	NeurIPS19	✓	✓		✓	2.59	3.4	1
	TAS [126]	NeurIPS19		✓	✓	✓	6.82	N/A	0.3
	XNAS [115]	NeurIPS19	✓	✓		✓	1.60	7.2	0.3
GDAS [83]	CVPR19	✓✓		✓	2.82	2.5	0.17		
	FBNet-C [103]	CVPR19	✓	✓		✓	N/A	N/A	N/A
	SGAS [153]	CVPR20	✓	✓		✓	2.66	3.7	0.25
	GDAS-NSAS [25]	CVPR20	✓	✓		✓	2.73	3.5	0.4
	PC-DARTS + Cutout [84]	CVPR20	✓	✓		✓	2.57	3.6	0.1
	R-DARTS [111]	ICLR20	✓	✓		✓	2.93	4.1	N/A
RS	Hierarchical-EAS Random [34]	ICLR18	✓				3.91	N/A	300
	NAO Random-WS [74]	NeurIPS18	✓			✓	3.92	3.9	0.3
	NASH-Net Random [85]	ICLR18					6.50	4.4	0.2
	DARTS Random [17]	ICLR19	✓				3.29	3.2	4
	RandomNAS + Cutout [105]	UAI19	✓			✓	2.85	4.3	2.7
	RandomNAS-NSAS [25]	CVPR20	✓				2.64	3.08	0.7
SMBO	NASBOT [61]	NeurIPS18					8.69	N/A	1.7
	NAO [74]	NeurIPS18	✓	✓		✓	2.98	28.6	200
	NAO-WS [74]	NeurIPS18	✓	✓		✓	3.53	2.5	0.3
	NAO-Cutout [74]	NeurIPS18	✓	✓		✓	2.11	128	200
	PNAS [37]	ECCV18	✓			✓	3.41	3.2	225
	DPP-Net [35]	ECCV18	✓			✓	5.84	0.45	2

In the interests of clarity, we classify NAS according to mainstream search methods: reinforcement learning (RL), evolutionary algorithm (EA), gradient optimization (GO), random search (RS), and sequential model-based optimization (SMBO). At the same time, the optimization strategy used in NAS is reported based on the analysis in Section 3. Cutout is an augmentation technology used in [131].

(ImageNet). Therefore, the search time costs involved in the two datasets shown in Tables 2 and 3 are the same [17, 32, 39, 46]. These search tasks conducted in advance on small datasets are often called agent tasks. ProxylessNAS [104] also considers the problem of searching on large target tasks directly and without agents.

In this part, we compare and summarize the popular NAS method with mainstream, artificially designed networks in multiple dimensions in terms of both performance and parameter amount. It is, however, worth noting that the performance gains obtained using NAS are limited compared to those achieved using manually designed networks.

Table 3. The Performance Comparison between the State-of-the-Art NAS and Mainstream Artificial Networks on ImageNet

Search method	Reference	Venue	Top 1/Top 5 Acc (%)	Params (Millions)	Image Size (squared)	GPU Days
Human	Mobilenets [6]	CoRR17	70.6/89.5	4.2	224	-
	ResNeXt [140]	CVPR17	80.9/95.6	83.6	320	-
	Polynet [141]	CVPR17	81.3/95.8	92.0	331	-
	DPN [142]	NIPS17	81.5/95.8	79.5	320	-
	Shufflenet [139]	CVPR18	70.9/89.8	5.0	224	-
RL	NASNet [32]	CVPR18	82.7/96.2	88.9	331	2,000
	NASNet-A [32]	CVPR18	74.0/91.6	5.3	224	2,000
	Block-QNN [33]	CVPR18	77.4/93.5	N/A	224	96
	N2N learning [52]	ICLR18	69.8/N/A	3.34	32	11.3
	Path-level EAS [57]	ICML18	74.6/91.9	594	224	200
	FPNAS [39]	ICCV19	73.3/N/A	3.41	224	0.8
EA	GeNet [16]	ICCV17	72.1/90.4	156	224	17
	Hierarchical-EAS [34]	ICLR18	79.7/94.8	64.0	299	300
	AmoebaNet-A *(N=6, F=190)* [43]	AAAI19	82.8/96.1	86.7	331	3,150
	AmoebaNet-A *(N=6, F=448)* [43]	AAAI19	83.9/96.6	469	331	3,150
	Single-Path One-Shot NAS [106]	CoRR19	74.7/N/A	N/A	224	<1
GO	Understanding One-Shot Models [22]	ICML18	75.2/N/A	11.9	224	N/A
	SMASH [23]	ICLR18	61.4/83.7	16.2	32	3
	Maskconnect [70]	ECCV18	79.8/94.8	N/A	224	N/A
	PARSEC [132]	CoRR19	74.0/91.6	5.6	N/A	1
	DARTS [17]	ICLR19	73.3/91.3	4.7	224	4
	SNAS [46]	ICLR19	72.7/90.8	4.3	224	1.5
	ProxylessNAS [104]	ICLR19	75.1/92.5	N/A	224	8.33
	GHN [91]	ICLR19	73.0/91.3	6.1	224	0.84
	SETN [92]	ICCV19	74.3/92.0	N/A	224	1.8
	TAS [126]	NeurIPS19	69.2/89.2	N/A	224	2.5
	XNAS [115]	NeurIPS19	76.1/N/A	5.2	224	0.3
	GDAS [83]	CVPR19	72.5/90.9	4.4	224	0.17
	FBNet-C [103]	CVPR19	74.9/N/A	5.5	224	9
	SGAS [153]	CVPR20	75.6/92.6	5.4	224	0.25
	PC-DARTS *(CIFAR10)* [84]	ICLR20	74.9/92.2	5.3	224	0.1
	PC-DARTS *(ImageNet)* [84]	ICLR20	75.8/92.7	5.3	224	3.8
RS	Hierarchical-EAS Random [34]	ICLR18	79.0/94.8	N/A	299	300
SMBO	PNAS *(Mobile)* [37]	ECCV18	74.2/91.9	5.1	224	225
	PNAS *(Large)* [37]	ECCV18	82.9/96.2	86.1	331	225
	DPP-Net [35]	ECCV18	75.8/92.9	77.2	224	2

The search strategy adopted by the corresponding NAS is consistent with Table 2. Cutout is an augmentation technology used in [131].

5 FUTURE DIRECTIONS

The emergence of NAS is exciting, as it is expected to end the tedious trial and error process of manual neural architecture design. Moreover, it is also hoped that it can design a network structure that is completely different from the artificial network, thereby breaking through the existing human mindset. Artificially designed networks have made breakthroughs in many fields, including image recognition [4, 7, 113], machine translation [1, 3, 143, 151], semantic segmentation [144, 145, 150], object detection [8, 148, 149], video understanding [146, 147], and so on. Although the NAS-related research has been quite rich, compared with the rapid development of artificial neural architecture design in various fields, NAS is still in the preliminary research stage. The current NAS is primarily focused on improving image classification accuracy, compressing the time required to search for neural architecture, and making it as light and easy to promote as possible. In this process, a proper baseline is crucial, as this helps to avoid NAS search strategy research being masked by various powerful augmenting technologies, regularization, and search space design tricks. In addition, the search strategies currently used by NAS are relatively concentrated, especially GO based on supernets, and there are many theoretical deficiencies in the related research. Therefore, improving the related research background is of great benefit to the development of NAS.

More specifically, from the original intention of NAS design, early NAS was very close to people's expectations for automatic neural architecture design. For example, Large-scale Evolution [15] uses an EA as a NAS search strategy, and emphasizes that there is no need for manual participation once network evolution begins. In the case of reducing human interference as much as possible, the algorithm is allowed to autonomously determine the evolution direction of the network. This is a good start, although the search performance at the time was insufficient. This is mainly because Large-scale Evolution emphasizes the reduction of artificial restrictions to the greatest extent possible, and also evolves from the simplest single-layer network (as shown in Figure 3). This means that the population must contain a sufficient number of individuals to evolve satisfactory results. In addition, due to the limitations of evolutionary efficiency and the huge search space, it is difficult to evolve a network structure that can achieve outstanding performance. Therefore, subsequent NAS works began to discuss how to reduce the search space as much as possible while also improving network performance. NASNet [32] benefited from the idea of artificial neural architecture design [4, 30, 31], and proposes a modular search space that was later widely adopted. However, this comes at the expense of the freedom of neural architecture design. Although this modular search space does significantly reduce the search cost, it is unclear in reality whether a better neural architecture has been ignored in the process of turning global search into modular search. This is also the main reason why it is unclear whether the modular search is better than a global search (related research is also lacking). In addition, the freedom of neural architecture design and the search cost presents a contradiction. How to balance the two and obtain good performance remains an important future research direction.

Also, there are two issues worthy of vigilance. From the perspective of the baseline, as analyzed in Section 4, a widely criticized problem of NAS is the lack of a corresponding baseline and sharable experimental protocol, which makes it difficult to compare NAS search algorithms with each other. While RS has proven to be a strong baseline [24, 34, 105, 129], there is still insufficient research on this subject. [24] pointed out that the performance of the current optimal NAS algorithm is similar to the random strategy, which should arouse researchers' concerns. Therefore, more ablation experiments are necessary, and researchers should pay more attention to which part of the NAS design leads to performance gains. Blindly stacking certain techniques to increase performance should also be criticized. Therefore, relevant theoretical analysis and reflection are crucial to the future development of NAS.

Another issue that requires vigilance is the widely used parameter sharing strategy [19]; although it effectively improves NAS search efficiency, an increasing body of evidence shows that the parameter sharing strategy is likely to result in a sub-optimal inaccurate candidate architecture ranking [21, 24, 25, 100]. This would make it almost impossible for NAS to find the optimal neural architecture in the search space. Therefore, relevant theoretical research and improvements are expected in the future. In addition, the current NAS works are mainly focused on improving the accuracy of image classification and reducing the search cost. In the future, NAS will play a greater role in areas that require complex neural architecture design, such as multi-object architecture search, network transformation that employs NAS based on existing models, model compression, target detection, segmentation, and so on.

RobNet [152] uses the NAS method to generate a large number of neural architectures by analyzing the differences between strongly and poorly performing architectures; this will enable the identification of which network structures help to obtain a robust neural architecture. Inspired by this, the once feasible idea would be to use a similar concept to RobNet in a search space with a higher degree of freedom, analyze the structural features common to promising architectures, and increase their structural proportion in the search space. In addition, it enables a reduction in the proportion of the common structural features of poorly performing architectures in the search space, to gradually reduce the search space in stages. The algorithm should be allowed to autonomously decide which network structures should be removed or added rather than artificially imposing constraints.

On the other hand, in the design of deep neural networks, in addition to the design of the neural architecture that consumes a lot of researchers' energy, another headache is the setting of non-architecture hyperparameters (e.g., optimizer type, initial learning rate, weight decay, mixup ratio [159], drop out ratio, and stochastic depth drop ratio [160]), because the impact of these hyperparameters on network performance is also crucial. Moreover, the number of possible related non-architectural hyperparameter groups and architectural combinations is often very large, and it is difficult and inefficient to manually design all possible combinations. Therefore, it is also a very promising research direction to consider how to integrate the hyperparameters in the neural architecture into the search space and allow the algorithm to automatically adjust the settings of the hyperparameters. Although some early methods also explored the joint search of hyperparameters and architectures, they mainly focused on small datasets and small search spaces. Fortunately, the experiments of FBNetV3 [157] and AutoHAS [158] show that this direction has great potential, and it is expected to find higher-accuracy architecture-recipe combinations.

In short, the emergence of NAS is an exciting development. At present, NAS is still in its initial stages, and additional theoretical guidance and experimental analysis are required. Determining which NAS designs lead to improved performance is a critical element of improving NAS. If NAS is to completely replace the design of artificial neural architecture, more research, and a more solid theoretical basis are required. There is still a long way to go.

6 REVIEW THREATS

We made the above summary and discussion based on the open NAS-related work. We objectively state the viewpoints in some unpublished papers that need to be further confirmed. Due to limited time and energy, we directly applied the results in the references without reproducing their results.

7 SUMMARY AND CONCLUSIONS

This survey provides an insightful observation of the early NAS work and summarizes the four characteristics shared by the early NAS. In response to these challenges, we conducted a comprehensive and systematic analysis and comparison of existing work in turn. For the first time, we

reviewed all NAS-related work from the perspective of challenges and corresponding solutions. And the performance of existing NAS work was comprehensively compared and discussed. Finally, we conducted a comprehensive discussion on the possible future direction of NAS and raised two questions worthy of vigilance. This is very useful for future NAS research.

REFERENCES

[1] S. Hochreiter and J. Schmidhuber. 1997. Lonort-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[2] M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar, M. Schuster, Z. Chen, Y. Wu, and M. Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. arXiv:1804.09849. Retrieved from https://arxiv.org/pdf/1804.09849.pdf.

[3] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv:1609.08144. https://arxiv.org/pdf/1609.08144.pdf.

[4] K. Simonyan and A. Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. *ICLR*.

[5] M. Suganuma, S. Shirakawa, and T. Nagao. 2017. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 497–504.

[6] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861. https://arxiv.org/pdf/1704.04861.pdf.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.

[8] S. Ren, K. He, R. Girshick, and J. Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*. 91–99.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg. 2016. SSD: Single shot multibox detector. In *European Conference on Computer Vision*. Springer, Cham., 21–37.

[10] T. Y. Lin, P. Goyal, R. Girshick, K. He, and Dollár, P. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*. 2980–2988.

[11] B. Zoph and Q. V. Le. 2017. Neural architecture search with reinforcement learning. *ICLR*.

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

[13] N. Dalal and B. Triggs. 2005. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. IEEE, Vol. 1, 886–893.

[14] D. G. Lowe. 1999. Object recognition from local scale-invariant features. In *Proceedings of the 7th IEEE International Conference on Computer Vision*. IEEE, Vol. 2, 1150–1157.

[15] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, ... and A. Kurakin. 2017. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning*, Volume 70, 2902–2911. JMLR. org.

[16] L. Xie and A. Yuille. 2017. Genetic CNN. In *Proceedings of the IEEE International Conference on Computer Vision*. 1379–1388.

[17] H. Liu, K. Simonyan, and Y. Yang. 2018. Darts: Differentiable architecture search. arXiv:1806.09055.

[18] Y. Shu, W. Wang, and S. Cai. 2019. Understanding architectures learnt by cell-based neural architecture search. arXiv:1909.09569.

[19] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. 2018. Efficient neural architecture search via parameter sharing. arXiv:1802.03268.

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

[21] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang. 2020. Block-wisely supervised neural architecture search with knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1989–1998.

[22] G. Bender, P. Kindermans, B. Zoph, V. Vasudevan, and Q. Le. 2018. Understanding and simplifying one-shot architecture search. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 80:550–559.

[23] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. 2017. Smash: One-shot model architecture search through hypernetworks. arXiv:1708.05344.

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

[25] M. Zhang, H. Li, S. Pan, X. Chang, and S. Su. 2020 Overcoming multi-model forgetting in one-shot NAS with diversity maximization. In *Advances in Neural Information Processing Systems*.
[26] X. Cheng, Y. Zhong, M. Harandi, Y. Dai, X. Chang, H. Li, T. Drummond, and Z. Ge. 2020. Hierarchical neural architecture search for deep stereo matching. In *Advances in Neural Information Processing Systems*.
[27] T. Elsken, J. H. Metzen, and F. Hutter. 2018. Neural architecture search: A survey. arXiv:1808.05377.
[28] M. Wistuba, A. Rawat, and T. Pedapati. 2019. A survey on neural architecture search. arXiv:1905.01392.
[29] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. 2019. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2820–2828.
[30] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.
[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, ... and A. Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.
[32] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8697–8710.
[33] Z. Zhong, J. Yan, W. Wu, J. Shao, and C. L. Liu. 2018. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2423–2432.
[34] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. 2017. Hierarchical representations for efficient architecture search. arXiv:1711.00436.
[35] J. D. Dong, A. C. Cheng, D. C. Juan, W. Wei, and M. Sun. 2018. DPP-Net: Device-aware progressive search for Pareto-optimal neural architectures. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 517–531.
[36] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4700–4708.
[37] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. J. Li, ... and K. Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 19–34.
[38] T. Saikia, Y. Marrakchi, A. Zela, F. Hutter, and T. Brox. 2019. AutoDispNet: Improving disparity estimation with AutoML. In *Proceedings of the IEEE International Conference on Computer Vision*. 1812–1823.
[39] J. Cui, P. Chen, R. Li, S. Liu, X. Shen, and J. Jia. 2019. Fast and practical neural architecture search. In *Proceedings of the IEEE International Conference on Computer Vision*. 6509–6518.
[40] Y. Xiong, R. Mehta, and V. Singh. 2019. Resource constrained neural network architecture search: Will a submodularity assumption help? In *Proceedings of the IEEE International Conference on Computer Vision*. 1901–1910.
[41] M. S. Ryoo, A. J. Piergiovanni, M. Tan, and A. Angelova. 2019. AssembleNet: Searching for multi-stream neural connectivity in video architectures. arXiv:1905.13209.
[42] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, ... and Q. V. Le. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*. 1223–1231.
[43] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, 4780–4789.
[44] X. Chen, L. Xie, J. Wu, and Q. Tian. 2019. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*. 1294–1303.
[45] A. J. Piergiovanni, A. Angelova, A. Toshev, and M. S. Ryoo. 2019. Evolving space-time neural architectures for videos. In *Proceedings of the IEEE International Conference on Computer Vision*. 1793–1802.
[46] S. Xie, H. Zheng, C. Liu, and L. Lin. 2018. SNAS: Stochastic neural architecture search. arXiv:1812.09926.
[47] T. Chen, I. Goodfellow, and J. Shlens. 2015. Net2net: Accelerating learning via knowledge transfer. arXiv:1511.05641.
[48] M. Schuster and K. K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681.
[49] P. Bashivan, M. Tensen, and J. J. DiCarlo. 2019. Teacher guided architecture search. In *Proceedings of the IEEE International Conference on Computer Vision*. 5320–5329.
[50] X. Zheng, R. Ji, L. Tang, B. Zhang, J. Liu, and Q. Tian. 2019. Multinomial distribution learning for effective neural architecture search. In *Proceedings of the IEEE International Conference on Computer Vision*. 1304–1313.
[51] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. 2018. Efficient architecture search by network transformation. In *32nd AAAI Conference on Artificial Intelligence*. 2787–2794.
[52] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani. 2017. N2N learning: Network to network compression via policy gradient reinforcement learning. arXiv:1709.06030.
[53] J. Mei, Y. Li, X. Lian, X. Jin, L. Yang, A. Yuille, and J. Yang. 2019. AtomNAS: Fine-grained end-to-end neural architecture search. arXiv:1912.09640.
[54] X. Gong, S. Chang, Y. Jiang, and Z. Wang. 2019. AutoGAN: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 3224–3234.

[55] R. Pasunuru and M. Bansal. 2019. Continual and multi-task architecture search. arXiv:1906.05226.
[56] G. Hinton, O. Vinyals, and J. Dean. 2015. Distilling the knowledge in a neural network. arXiv:1503.02531.
[57] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu. 2018. Path-level network transformation for efficient architecture search. arXiv:1806.02639.
[58] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. 2017. Inception-v4, inception-ResNet and the impact of residual connections on learning. In *31st AAAI Conference on Artificial Intelligence*.
[59] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2818–2826.
[60] J. Fang, Y. Sun, K. Peng, Q. Zhang, Y. Li, W. Liu, and X. Wang. 2020. Fast neural network adaptation via parameter remapping and architecture search. arXiv:2001.02525.
[61] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing. 2018. Neural architecture search with Bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*. 2016–2025.
[62] R. Negrinho and G. Gordon. 2017. DeepArchitect: Automatically designing and training deep architectures. arXiv:1704.08792.
[63] C. Liu, L. C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei. 2019. Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 82–92.
[64] S. Ding, T. Chen, X. Gong, W. Zha, and Z. Wang. 2020. AutoSpeech: Neural architecture search for speaker recognition. arXiv:2005.03215.
[65] Y. Zhang, Z. Qiu, J. Liu, T. Yao, D. Liu, and T. Mei. 2019. Customizable architecture search for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 11641–11650.
[66] Y. Chen, T. Yang, X. Zhang, G. Meng, C. Pan, and J. Sun. 2019. DetNAS: Neural architecture search on object detection. arXiv:1903.10979.
[67] G. Anandalingam and T. L. Friesz. 1992. Hierarchical optimization: An introduction. *Annals of Operations Research* 34, 1 (1992), 1–11.
[68] B. Colson, P. Marcotte, and G. Savard. 2007. An overview of bilevel optimization. *Annals of Operations Research* 153, 1 (2007), 235–256.
[69] R. Shin, C. Packer, and D. Song. 2018. Differentiable Neural Network Architecture Search. *ICLR*.
[70] K. Ahmed and L. Torresani. 2018. MaskConnect: Connectivity learning by gradient descent. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 349–365.
[71] S. Saxena and J. Verbeek. 2016. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*. 4053–4061.
[72] K. Ahmed and L. Torresani. 2017. Connectivity learning in multi-branch networks. arXiv:1709.09582.
[73] T. Veniat and L. Denoyer. 2018. Learning time/memory-efficient deep architectures with budgeted super networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3492–3500.
[74] R. Luo, F. Tian, T. Qin, E. Chen, and T. Y. Liu. 2018. Neural architecture optimization. In *Advances in Neural Information Processing Systems*. 7816–7827.
[75] J. Chang, Y. Guo, G. Meng, S. Xiang, and C. Pan. 2019. DATA: Differentiable ArchiTecture approximation. In *Advances in Neural Information Processing Systems*. 874–884.
[76] G. Ghiasi, T. Y. Lin, and Q. V. Le. 2019. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7036–7045.
[77] D. Tran, J. Ray, Z. Shou, S. F. Chang, and M. Paluri. 2017. ConvNet architecture search for spatiotemporal feature learning. arXiv:1708.05038.
[78] L. C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, ... and J. Shlens. 2018. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems*. 8699–8710.
[79] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter. 2019. NAS-bench-101: Towards reproducible neural architecture search. arXiv:1902.09635.
[80] Y. Jiang, C. Hu, T. Xiao, C. Zhang, and J. Zhu. 2019. Improved differentiable architecture search for language modeling and named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP'19)*. 3576–3581.
[81] J. Lafferty, A. McCallum, and F. C. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *ICML* 2001 (2001), 282–289.
[82] D. Koller and N. Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.
[83] X. Dong and Y. Yang. 2019. Searching for a robust neural architecture in four GPU hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1761–1770.
[84] Y. Xu, L. Xie, X. Zhang, X. Chen, G. Qi, Q. Tian, and H. Xiong. 2019. PC-DARTS: Partial channel connections for memory-efficient architecture search. arXiv:abs/1907.05737.

[85] T. Elsken, J. H. Metzen, and F. Hutter. 2017. Simple and efficient architecture search for convolutional neural networks. arXiv:1711.04528.
[86] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. 2014. CNN features off-the-shelf: An astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 806–813.
[87] B. Zoph, D. Yuret, J. May, and K. Knight. 2016. Transfer learning for low-resource neural machine translation. arXiv:1604.02201.
[88] M. T. Luong, Q. V. Le, I. Sutskever, O. Vinyals, and L. Kaiser. 2015. Multi-task sequence to sequence learning. arXiv:1511.06114.
[89] D. Ha, A. Dai, and Q. V. Le. 2016. Hypernetworks. arXiv:1609.09106.
[90] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. 2017. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *International Conference on Learning Representations 2017 (ICLR'17)*.
[91] C. Zhang, M. Ren, and R. Urtasun. 2018. Graph hypernetworks for neural architecture search. arXiv:1810.05749.
[92] X. Dong and Y. Yang. 2019. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE International Conference on Computer Vision*. 3681–3690.
[93] M. Mirza and S. Osindero. 2014. Conditional generative adversarial nets. arXiv:1411.1784.
[94] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. 2016. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*. 2234–2242.
[95] T. Karras, T. Aila, S. Laine, and J. Lehtinen. 2017. Progressive growing of GANs for improved quality, stability, and variation. arXiv:1710.10196.
[96] N. T. Tran, T. A. Bui, and N. M. Cheung. 2018. Dist-GAN: An improved GAN using distance constraints. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 370–385.
[97] W. Wang, Y. Sun, and S. Halgamuge. 2018. Improving MMD-GAN training with repulsive loss function. arXiv:1812.09916.
[98] Q. Hoang, T. D. Nguyen, T. Le, and D. Phung. 2018. MGAN: Training generative adversarial nets with multiple generators. In *ICLR*.
[99] G. Ghiasi, T. Y. Lin, and Q. V. Le. 2019. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7036–7045.
[100] H. Cai, C. Gan, and S. Han. 2019. Once for all: Train one network and specialize it for efficient deployment. arXiv:1908.09791.
[101] X. Chu, B. Zhang, R. Xu, and J. Li. 2019. FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search. arXiv:1907.01845.
[102] X. Li, C. Lin, C. Li, M. Sun, W. Wu, J. Yan, and W. Ouyang. 2019. Improving one-shot NAS by suppressing the posterior fading. arXiv:1910.02543.
[103] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, ... and K. Keutzer. 2019. FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 10734–10742.
[104] H. Cai, L. Zhu, and S. Han. 2018. ProxylessNAS: Direct neural architecture search on target task and hardware. arXiv:1812.00332.
[105] L. Li and A. Talwalkar. 2019. Random search and reproducibility for neural architecture search. arXiv:1902.07638.
[106] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun. 2019. Single path one-shot neural architecture search with uniform sampling. arXiv:1904.00420.
[107] T. Domhan, J. T. Springenberg, and F. Hutter. 2015. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *24th International Joint Conference on Artificial Intelligence*.
[108] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. 2016. Learning curve prediction with Bayesian neural networks. In *International Conference on Learning Representation*. 184–194.
[109] A. Chandrashekaran and I. R. Lane. 2017. Speeding up hyper-parameter optimization by extrapolation of learning curves using previous builds. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, Cham, 477–492.
[110] B. Deng, J. Yan, and D. Lin. 2017. Peephole: Predicting network performance before training. arXiv:1712.03351.
[111] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter. 2020. Understanding and robustifying differentiable architecture search. In *ICLR*.
[112] J. Peng, M. Sun, Z. X. Zhang, T. Tan, and J. Yan. 2019. Efficient neural architecture transformation search in channel-level for object detection. In *Advances in Neural Information Processing Systems*. 14290–14299.
[113] Y. Zhu et al. [n.d.]. Deep subdomain adaptation network for image classification. In *IEEE Transactions on Neural Networks and Learning Systems*. DOI: 10.1109/TNNLS.2020.2988928.

[114] Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, Chuan Zhou, Zongyuan Ge, and Steven W. Su. One-shot neural architecture search: Maximising diversity to overcome catastrophic forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. DOI: 10.1109/TPAMI.2020.3035351

[115] N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik. 2019. XNAS: Neural architecture search with expert advice. In *Advances in Neural Information Processing Systems*. 1975–1985.

[116] S. Cao, X. Wang, and K. M. Kitani. 2019. Learnable embedding space for efficient neural architecture compression. arXiv:1902.00383.

[117] T. Elsken, J. H. Metzen, and F. Hutter. 2018. Efficient multi-objective neural architecture search via Lamarckian evolution. arXiv:1804.09081.

[118] X. Li, Y. Zhou, Z. Pan, and J. Feng. 2019. Partial order pruning: For best speed/accuracy trade-off in neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9145–9153.

[119] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, ... and P. Vajda. 2019. ChamNet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 11398–11407.

[120] F. Liang, C. Lin, R. Guo, M. Sun, W. Wu, J. Yan, and W. Ouyang. 2019. Computation reallocation for object detection. arXiv:1912.11234.

[121] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough. 2019. Sparse: Sparse architecture search for CNNs on resource-constrained microcontrollers. In *Advances in Neural Information Processing Systems*. 4978–4990.

[122] V. Nekrasov, H. Chen, C. Shen, and I. Reid. 2019. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9126–9135.

[123] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. 2019. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 113–123.

[124] M. Tan and Q. V. Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. arXiv:1905.11946.

[125] X. Zhang, Q. Wang, J. Zhang, and Z. Zhong. 2019. Adversarial AutoAugment. arXiv:1912.11188.

[126] X. Dong and Y. Yang. 2019. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*. 759–770.

[127] Z. Lu, I. Whalen, V. Boddeti, Y. D. Dhebar, K. Deb, E. D. Goodman, and W. Banzhaf. 2018. NSGA-NET: A multi-objective genetic algorithm for neural architecture search. *Computer Vision and Pattern Recognition*.

[128] X. Dong, L. Liu, K. Musial, and B. Gabrys. 2020. NATS-Bench: Benchmarking NAS algorithms for architecture topology and size. arXiv:2009.00437.

[129] A. Yang, P. M. Esperança, and F. M. Carlucci. 2019. NAS evaluation is frustratingly hard. arXiv:1912.12522.

[130] M. Wistuba. 2018. Deep learning architecture search by neuro-cell-based evolution with function-preserving mutations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, Cham, 243–258.

[131] T. DeVries and G. W. Taylor. 2017. Improved regularization of convolutional neural networks with cutout. arXiv:1708.04552.

[132] F. P. Casale, J. Gordon, and N. Fusi. 2019. Probabilistic neural architecture search. arXiv:abs/1902.05116.

[133] S. Zagoruyko and N. Komodakis. 2016. Wide residual networks. arXiv:1605.07146.

[134] X. Gastaldi. 2017. Shake-shake regularization. arXiv:1705.07485.

[135] Y. Yamada, M. Iwamura, and K. Kise. 2016. Deep pyramidal residual networks with separated stochastic depth. arXiv:1612.01230.

[136] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. 2016. Deep networks with stochastic depth. In *European Conference on Computer Vision*. Springer, Cham, 646–661.

[137] G. Larsson, M. Maire, and G. Shakhnarovich. 2016. FractalNet: Ultra-deep neural networks without residuals. arXiv:1605.07648.

[138] H. Zhou, M. Yang, J. Wang, and W. Pan. 2019. BayesNAS: A Bayesian approach for neural architecture search. arXiv:1905.04919.

[139] X. Zhang, X. Zhou, M. Lin, and J. Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6848–6856.

[140] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. 2017. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1492–1500.

[141] X. Zhang, Z. Li, C. Change Loy, and D. Lin. 2017. PolyNet: A pursuit of structural diversity in very deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 718–726.

[142] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng. 2017. Dual path networks. In *Advances in Neural Information Processing Systems*. 4467–4475.

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

[144] J. Long, E. Shelhamer, and T. Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3431–3440.

[145] W. Sun, Z. Huang, M. Liang, T. Shao, and H. Bi. 2020. Cocoon image segmentation method based on fully convolutional networks. In *Proceedings of the 7th Asia International Symposium on Mechatronics*. Springer, Singapore, 832–843.

[146] G. Vecchio, S. Palazzo, D. Giordano, F. Rundo, and C. Spampinato. [n.d.]. MASK-RL: Multiagent video object segmentation framework through reinforcement learning. In *IEEE Transactions on Neural Networks and Learning Systems*. DOI: 10.1109/TNNLS.2019.2963282.

[147] Z. Ji, Y. Zhao, Y. Pang, X. Li and J. Han. [n.d.]. Deep attentive video summarization with distribution consistency learning. In *IEEE Transactions on Neural Networks and Learning Systems*. DOI: 10.1109/TNNLS.2020.2991083.

[148] D. Zhang, J. Han, L. Zhao, and T. Zhao. [n.d.]. From discriminant to complete: Reinforcement searching-agent learning for weakly supervised object detection. In *IEEE Transactions on Neural Networks and Learning Systems*. DOI: 10.1109/TNNLS.2020.2969483.

[149] K. Shih, C. Chiu, J. Lin, and Y. Bu. 2020. Real-time object detection with reduced-region proposal network via multi-feature concatenation. In *IEEE Transactions on Neural Networks and Learning Systems*. 31, 6 (June 2020, pp. 2164–2173. DOI: 10.1109/TNNLS.2019.2929059.

[150] Y. Zhou, G. G. Yen, and Z. Yi. [n.d.]. Evolutionary compression of deep neural networks for biomedical image segmentation. In *IEEE Transactions on Neural Networks and Learning Systems*. DOI: 10.1109/TNNLS.2019.2933879.

[151] B. Zhang, D. Xiong, J. Xie, and J. Su. [n.d.]. Neural machine translation with GRU-gated attention model. In *IEEE Transactions on Neural Networks and Learning Systems*. DOI: 10.1109/TNNLS.2019.2957276.

[152] M. Guo, Y. Yang, R. Xu, and Z. Liu. 2019. When NAS meets robustness: In search of robust architectures against adversarial attacks. arXiv:abs/1911.10695.

[153] G. Li, G. Qian, I. C. Delgadillo, M. Müller, A. Thabet, and B. Ghanem. 2019. SGAS: Sequential greedy architecture search. arXiv:1912.00195.

[154] J. Fang, Y. Sun, Q. Zhang, Y. Li, W. Liu, and X. Wang. 2019. Densely connected search space for more flexible neural architecture search. arXiv:1906.09607.

[155] D. So, C. Liang, and Q. V. Le. 2019. The evolved transformer. arXiv:abs/1901.11117.

[156] A. Zela, J. Siems, and F. Hutter. 2020. NAS-Bench-1Shot1: Benchmarking and dissecting one-shot neural architecture search. arXiv:2001.10422.

[157] X. Dai, A. Wan, P. Zhang, B. Wu, Z. He, Z. Wei, ... and J. E. Gonzalez. 2020. FBNetV3: Joint architecture-recipe search using neural acquisition function. arXiv:2006.02049.

[158] X. Dong, M. Tan, A. W. Yu, D. Peng, B. Gabrys, and Q. V. Le. 2020. AutoHAS: Differentiable hyper-parameter and architecture search. arXiv:2006.03656.

[159] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. arXiv:1710.09412.

[160] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. 2016. Deep networks with stochastic depth. In *European Conference on Computer Vision*. Springer, Cham, 646–661.

[161] D. R. Jones. 2001. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization* 21, 4 (2001), 345–383.

[162] F. Hutter, H. H. Hoos, and K. Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*. Springer, Berlin, 507–523.

Received June 2020; revised January 2021; accepted January 2021