A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions

PENGZHEN REN* and YUN XIAO*, Northwest University
XIAOJUN CHANG, Monash University
PO-YAO HUANG, Carnegie Mellon University
ZHIHUI LI, University of New South Wales
XIAOJIANG CHEN and XIN WANG, Northwest University

Deep learning has made major breakthroughs and progress in many fields. This is due to the powerful automatic representation capabilities of deep learning. It has been proved that the design of the network architecture is crucial to the feature representation of data and the final performance. In order to obtain a good feature representation of data, the researchers designed various complex network architectures. However, the design of the network architecture relies heavily on the researchers' prior knowledge and experience. Due to the limitations of human's inherent knowledge, it is difficult for people to jump out of the original thinking paradigm and design an optimal model. Therefore, a natural idea is to reduce human intervention as much as possible and let the algorithm automatically design the architecture of the network. Thus going further to the strong intelligence.

In recent years, a large number of related algorithms for *Neural Architecture Search* (NAS) have emerged. They have made various improvements to the NAS algorithm, and the related research work is complicated and rich. In order to reduce the difficulty for beginners to conduct NAS-related research, a comprehensive and systematic survey on the NAS is essential. Previously related surveys began to classify existing work mainly from the basic components of NAS: search space, search strategy and evaluation strategy. This classification method is more intuitive, but it is difficult for readers to grasp the challenges and the landmark work in the middle. Therefore, in this survey, we provide a new perspective: starting with an overview of the characteristics of the earliest NAS algorithms, summarizing the problems in these early NAS algorithms, and then giving solutions for subsequent related research work. In addition, we conducted a detailed and comprehensive analysis, comparison and summary of these works. Finally, we give possible future research directions.

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

Additional Key Words and Phrases: Neural Architecture Search, AutoDL, Modular Search Strategy, Continuous Search Space, Network Architecture Recycle, Incomplete Training.

ACM Reference Format:
Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2020. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *ACM Comput. Surv.* 37, 4, Article 111 (August 2020), 30 pages. https://doi.org/10.1145/1122445.1122456

*Both authors contributed equally to this research.

Authors' addresses: Pengzhen Ren, pzhren@foxmail.com; Yun Xiao, yxiao@nwu.edu.cn, Northwest University; Xiaojun Chang, Monash University; Po-Yao Huang, Carnegie Mellon University; Zhihui Li, University of New South Wales; Xiaojiang Chen; Xin Wang, Northwest University.

0360-0300/2020/8-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Dlearning has already demonstrated strong learning capabilities in many fields: machine translation [1–3], image recognition [4, 6, 7] and object detection [8–10]. This is mainly due to the powerful automatic feature extraction capabilities of deep learning for unstructured data. Deep learning has transformed the traditional way of manually designing features [13, 14] into automatic extraction [4, 29, 30]. This allows researchers to focus on the design of neural architecture [11, 12, 19]. However, the design of the neural architecture relies heavily on the researchers' prior knowledge and experience, which makes it difficult for beginners to make reasonable modifications to the network architecture according to their actual needs. In addition, human's existing prior knowledge and fixed thinking paradigm are likely to limit the discovery of a new network architecture to a certain extent.

As a result, *Neural architecture search* (NAS) came into being. NAS aims to design a network architecture with the best performance using limited computing resources in an automated way with as little human intervention as possible. The work of NAS-RL [11] and MetaQNN [12] is considered a pioneering work of NAS. The network architecture they obtained using *reinforcement learning* (RL) methods reached the state-of-the-art classification accuracy on the image classification task. This shows that the idea of automated network architecture design is feasible. Subsequently, the work of Large-scale Evolution [15] once again verified the feasibility of this idea, which uses evolutionary learning to achieve similar results. However, they have consumed hundreds of GPU days or even more computing resources in their respective methods. This huge amount of calculation is almost catastrophic for ordinary researchers. Therefore, a lot of work has emerged on how to reduce the amount of calculation and accelerate the search of the network architecture [18–20, 48, 49, 52, 84, 105]. With the improvement of NAS search efficiency, NAS is also quickly applied in the fields of object detection [65, 75, 111, 118], semantic segmentation [63, 64, 120], adversarial learning [53], architectural scaling [114, 122, 124], multi-objective optimization [39, 115, 125], platform-aware [28, 34, 103, 117], data augmentation [121, 123] and so on. In addition, there is some work to consider how to strike a balance between performance and efficiency [116, 119]. Although NAS-related research has been so abundant, it is still difficult to compare and reproduce NAS methods [127]. Because different NAS methods have many differences in search space, hyperparameters, tricks, etc., some work is also devoted to providing a unified evaluation platform for popular NAS methods [78, 126].

With the deepening and rapid development of NAS-related research, some methods previously accepted by researchers have been proved to be imperfect by new research. And soon there was an improved solution. For example, early NAS trained each candidate network architecture from scratch during the architecture search phase, leading to a surge in computation [11, 12]. ENAS [19] proposes to accelerate the process of architecture search by using a parameter sharing strategy. This strategy avoids training each subnet from scratch, but forces all subnets to share weights, thereby greatly reducing the time to obtain the best performing subnet from a large number of candidate networks. Due to the superiority of ENAS in search efficiency, the weight sharing strategy was quickly recognized by a large number of researchers [23, 53, 54]. However, soon new research found that the widely accepted weight sharing strategy is likely to lead to inaccurate ranking of candidate architectures [24]. This will make it difficult for the NAS to select the optimal network architecture from a large number of candidate architectures, thereby further deteriorating the performance of the finally searched network architecture. Shortly afterwards, DNA [21] modularized the large search space of NAS into blocks, so that the candidate architecture was 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

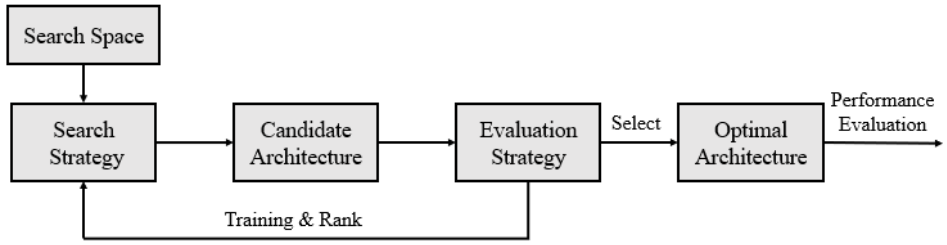


Fig. 1. The general framework of NAS. NAS generally starts with a set of predefined operation sets, and uses search strategies to obtain a large number of candidate network architectures based on the search space formed by the operation sets. The candidate network architecture is trained and ranked. Then, the search strategy is adjusted according to the ranking information of the candidate network architecture, thereby further obtaining a set of new candidate network architectures. When the search is terminated, the most promising network architecture is used as the final optimal network architecture, which is used for the final performance evaluation.

forgetting (when weight sharing is used to sequentially train a new network architecture, the performance of the previous network architecture is reduced) caused by weight sharing during the super network training process.

Similar research clues are very common in the rapidly developing NAS research field, so a comprehensive and systematic survey based on challenges and solutions is very useful for NAS research. Previous related surveys classified existing work mainly according to the basic components of NAS: search space, search strategy and evaluation strategy [26, 27]. This classification method is more intuitive, but it is not conducive to readers to capture the research clues. Therefore, in this survey, we will first summarize the characteristics and corresponding challenges of the early NAS methods. Based on these challenges, we have summarized and categorized existing research in order to show readers a comprehensive and systematic overview based on challenges and solutions. Finally, we will compare the performance of existing research work, and give possible future research directions and some thoughts.

2 CHARACTERISTICS OF EARLY NAS

In this section, we summarize the general framework of early NAS methods, their characteristics, and the challenges facing subsequent NAS research.

We summarize the general framework of NAS as shown in Fig.1. NAS usually starts with a set of predefined operation sets and uses a search strategy to obtain a large number of candidate network architectures based on the search space formed by these operation sets. Then train the candidate network architecture on the training set and rank them according to their accuracy on the validation set. The ranking information of the candidate network architecture is used as feedback information to adjust the search strategy to further obtain a set of new candidate network architectures. When the termination condition is reached, the search will be terminated to select the best network architecture. The network architecture performs performance evaluation on the test set.

Early NAS also roughly followed the above process [11, 12, 15, 16]. The idea of NAS-RL [11] comes from such a simple observation that the architecture of a neural network can be described as a variable-length string. Therefore, a very natural idea is that we can use RNN as a controller to generate such a string, and then use RL to optimize the controller, and finally get a satisfactory network architecture. MetaQNN [12] regards the selection process of the network architecture

as a Markov decision process, and uses Q-learning to record rewards, so as to obtain the optimal network architecture. Large-scale Evolution [15] aims to automatically learn an optimal network architecture using *evolutionary algorithms* (EA) while reducing human intervention as much as possible. It uses the simplest network structure to initialize a large population, and obtains the best network architecture by reproduce, mutating, and selecting the population. GeNet [16] also used EA, it proposes a new neural network architecture coding scheme, which represents the network architecture as a fixed-length binary string. It randomly initializes a group of individuals, uses a predefined set of genetic operations to modify the binary string to generate new individuals, and finally selects competitive individuals as the final network architecture.

These early NAS made the automatically generated network architecture a reality. In order to understand the reasons restricting the widespread use of early NAS, we summarized the common characteristics existing in early NAS work from the perspective of a latecomer as follows:

- **Global search strategy.** It requires the NAS to use a search strategy to search all necessary components of the network architecture. This means that NAS needs to find an optimal network architecture within a huge search space. Obviously, the larger the search space, the higher the corresponding search cost.
- **Discrete search space.** It regards the differences between different network architectures as a limited set of basic operations, that is, by discretely modifying an operation to change the network architecture. This means that we cannot use the gradient strategy to quickly adjust the network architecture.
- **Search from scratch.** The model is built from scratch until the final network architecture is generated. Obviously, this method wastes the existing network architecture design experience and cannot utilize the existing excellent network architecture.
- **Fully trained.** It requires training each candidate network architecture from scratch to convergence. We know that there is a similar network structure between the subsequent network architecture and the previous network architecture, as well as between the network architectures at the same stage. Therefore, training each candidate network architecture from scratch obviously does not fully utilize this relationship. In addition, 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.

The search space is determined by the predefined operation set and the hyperparameters of the network architecture (for example: an architectural template, connection method, the number of channels of the convolutional layer used for feature extraction in the initial stage, etc.). These parameters define which network architectures can be searched by the NAS. Fig.2 shows 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 goes through a series of operations to get the final output. Fig.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

$$z^{(i)} = o_i\{(z^{(i-1)})\}. \tag{1}$$

Fig.2 (right): The example after adding skip connections [11, 15, 16]. At this time, there can be multiple inputs for any feature map $z^{(i)}$, and

$$z^{(i)} = o^{(i)}\left(\left\{z^{(i-1)}\right\} \odot \left\{z^{(k)} | \alpha_{k,i} = 1, k < i-1\right\}\right), \tag{2}$$

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 a sum operation in GeNet [16]. It should be pointed out that NASNet [31]

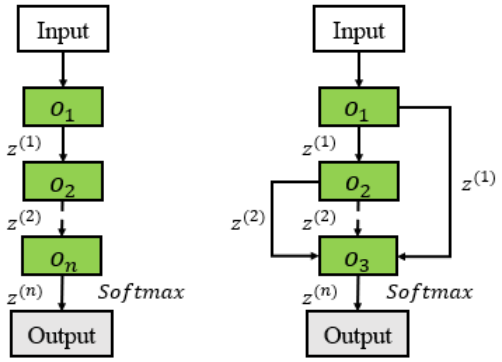


Fig. 2. Two common global search spaces with a chain structure in early NAS work. Left: The simplest example of a chain structure. Right: The example after adding skip connections. o_i is an operation in the candidate set of operations and the i-th operation in the chain structure. The feature map generated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

considered these two operations in the experiment, but the experimental results show that the sum operation is better than the merge operation. Therefore, since then, a lot of work has taken the summation operation as the connection method of the feature map obtained between different branch operations [17, 36, 37]. Similar to the chain structure, Mnasnet [28] suggests searching for a network architecture composed of multiple segments connected in sequence, each segment having its own repeating structure.

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

The common characteristics of these early NAS work are also the collective challenges faced by the automatic generation of network architecture. Based on the above challenges, we will summarize the solutions in the subsequent NAS-related research work in Section 3.

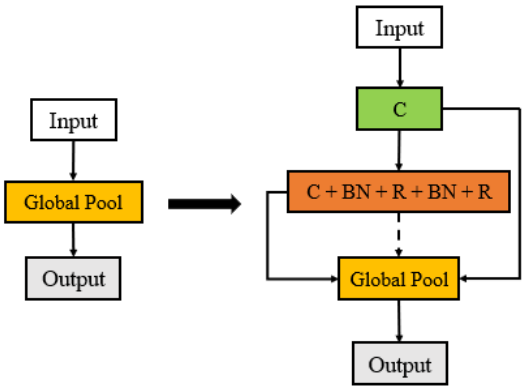


Fig. 3. Take Large-scale Evolution [15] as an example, starting from the simplest network architecture to gradually generate the final network architecture. *C*, *BN* and *R* are the convolution, Batch Normalization and ReLU operations in sequence.

3 OPTIMIZATION STRATEGY

For the characteristics and challenges of early NAS [11, 12, 15, 16], in this section, we will summarize the existing NAS research work through the following four aspects: modular search strategy, continuous search space, network architecture recycle and incomplete training.

3.1 Modular Search Strategy

The effect of the search space design on the final performance of the NAS algorithm is crucial. It not only determines the freedom of network architecture search, but also directly determines the upper limit of the performance of the NAS algorithm to some extent. Therefore, the reconstruction of the search space is necessary.

A common idea is to transform global search into a modular search strategy. Therefore, the cell-based search space is widely 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 only needs to search a few small cell structures, and then repeatedly stack such cells to form the final network architecture. However, the global search space needs to search all the components that build the entire network architecture. In addition, the cell-based search space can be migrated on different data set tasks by stacking different numbers of cells, but the global search space often does not have this feature. Therefore, compared with the global search space, the cell-based search space is more compact and flexible.

This idea mainly stems from the observation of excellent network architectures that have been artificially designed in recent years [4, 29, 30]. These artificial network architectures usually achieve the construction of the overall network architecture by repeatedly stacking a certain unit operation or a small structure. In NAS, this small repeating structure is often called a cell. The construction of cell-based network architecture is based on this idea. The network architecture constructed in this way is not only superior in performance, but also easy to generalize. NASNet [31] is one of the first to explore this idea. It proposes to search for two types of cells, normal cell and reduction cell. Normal cell: It is used to extract advanced features while keeping the spatial resolution unchanged. Reduction cell: It is mainly used to reduce the spatial resolution. Multiple repeated normal cells are followed by a reduction cell, and this connection is repeated multiple times to form the final network architecture. In Fig.4 (left), we show this kind of network architecture based on two cells.

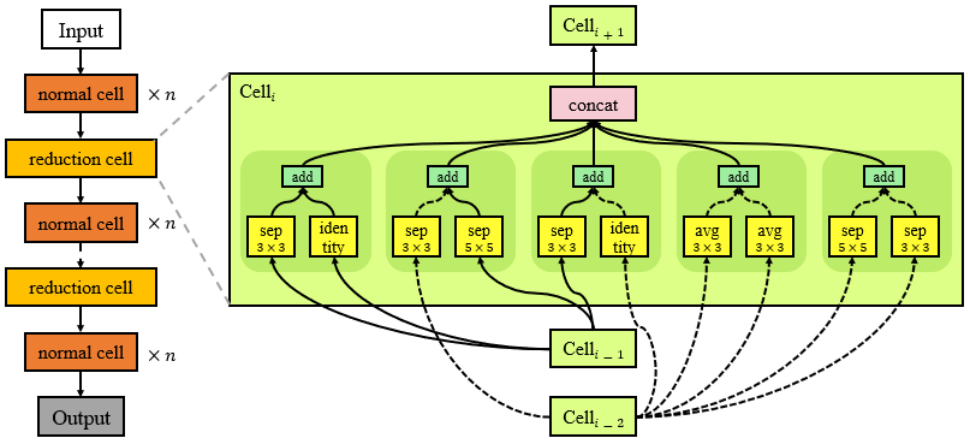


Fig. 4. Left: The structure of the search space example based on two cells in [31]. The normal cell is repeated n times and then connected to a reduction cell. The basic operation of the normal cell has a stride of 1, and the size of the feature map remains unchanged before and after output. The basic operation stride of the reduction cell is 2, and the size of the feature map is halved. Right: The best Normal cell with 5 blocks searched in [31].

In Fig.4 (right), we show the internal structure of an optimal normal cell in NASNet [31]. The structure of the corresponding reduction cell and normal cell is similar, the difference is that the basic operation step of the reduction cell is 2. A lot of follow-up work [17, 42, 43] used the search space similar to NASNet [31].

In ENAS [19], its experiments provide strong evidence for the selection of this similar cell-based search space. Subsequently, this cell-based search space was soon used by other research work. In [32–34, 44], they chose to use some unit operations to replace the reduction cell to complete the downsampling. At this time, the model only needs to search for a normal cell. We show this structure in Fig.5, the curved dotted line indicates the dense connection in Dpp-net [34]. At the same time as Block-QNN [32] of NASNet [31], the pooling operation is used to replace the reduction cell in order to reduce the size of the feature map. Hierarchical-EAS [33] uses a convolution with a kernal size of 3×3 and a stride of 2 instead of the reduction cell to reduce the spatial resolution. In addition, the idea of meta-operation is used to hierarchically build the structure of the cell. Dpp-net [34] is similar to Block-QNN [32], it uses average pooling operation instead of reduction cell. The difference is that Dpp-net [34] draws on the idea of DenseNet [35] to use dense connections including cells to build a network architecture, and proposes to take device into account for multi-objective optimization tasks. In [32–34], their structure of each cell is the same, only need to search for a cell. For video tasks, [44] uses $L \times 3 \times 3$, $stride = 1, 2, 2$ max-pooling instead of reduction cell. And, in order to adapt to the complex task of video and expand the search space, the structure of each cell can be different. AutoDispNet [37] proposes to apply the automatic architecture search technology to optimize large-scale U-Net-like encoder-decoder architectures. Therefore, it searches for three types of cells: normal, reduction, upsampling. In the coding stage, the network architecture consists of alternate connections of normal cells and reduction cells. In the decoding stage, it consists of a stack of multiple upsampling cells. [18] studied the structural commonality of the cells obtained from some popular cell-based search spaces [17, 19, 31, 42, 45]. It defines the width and depth of the cell, and proves from theory and experiment that due to the existence of the common connection mode, the wide and shallow cell is easier to converge during training and easier to be searched, but

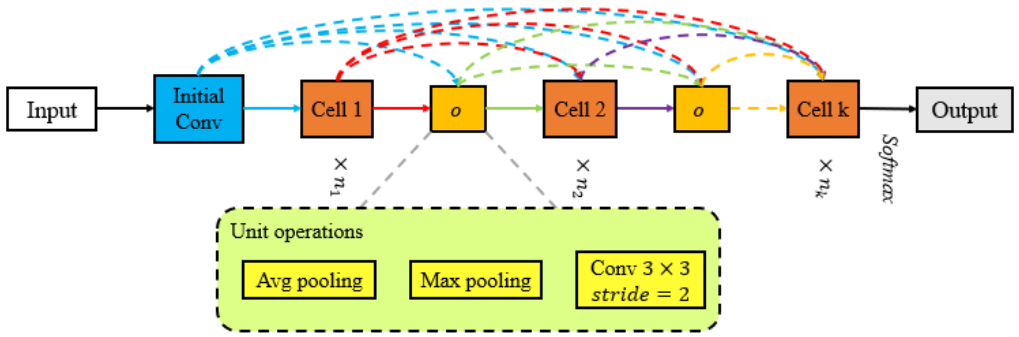


Fig. 5. Use unit operations to replace the reduction cell in [31], while having a densely connected network architecture. The curved dotted line indicates the dense connection in Dpp-net [34]. The initial convolution is to extract low-level features. After the cell is repeated multiple times, a unit operation is used for downsampling. This connection is repeated multiple times to form the final network architecture.

the generalization effect is poor. This provides guidance for us to understand cell-based NAS. In addition, there are many cell-based NAS-related work [53, 91].

In this section, we conduct a comprehensive review of the modular search strategy. Compared with global search, the cell-based modular search strategy effectively reduces the search space and makes NAS more friendly to researchers. Of course, this does not mean that cell-based search can meet all task requirements. Global search still has its unique research value because it gives the network architecture design a higher degree of freedom [38, 103, 152].

3.2 Continuous Search Space

The proposal of NAS is regarded as a revolution in network architecture design. However, NAS has a high demand for computation. For example, NASNet [31] used RL methods to spend 2000 GPU days to obtain the best architecture in CIFAR-10 and ImageNet. Similarly, AmoebaNet [42] spent 3150 GPU days using evolutionary learning. An internal reason for the inefficiency of these mainstream search methods based on RL [11, 12, 31], EA [15, 42], Bayesian optimization [61], SMBO [36] and MCTS [62] is that they regard network architecture search as a black-box optimization problem in a discrete search space.

For this, DAS [68] explores to transform the discrete network architecture space into a continuously differentiable form, and uses gradient optimization techniques to search the network architecture. It mainly focuses on the search of hyperparameters of convolutional layers: filter sizes, number of channels, and grouped convolutions. MaskConnect [69] found that the existing cell-based network architecture generally adopts a predefined fixed connection method between modules, for example, each module only connects its first two modules [29], or connects all the previous modules [35]. This connection method may not be optimal. It uses the modified gradient method to focus on exploring the connection method between modules. In addition, these works [70–72] are also exploring the search for network architecture on continuous domains. However, the search for these network architectures is limited to fine-tuning the specific structure of the network.

In order to solve the above challenges, DARTS [17] appeared. DARTS continuously relaxes the originally discrete search space, making it possible to use gradients to efficiently optimize the search space of the architecture. DARTS follows the cell-based search space of NASNet [31] and further normalizes it. This cell is regarded as a *directed acyclic graph* (DAG), which is formed by sequentially

connecting N nodes. Each cell 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, and the other is the state of the previous step. The output of cell is the concatenation result of all intermediate nodes. Each intermediate node $x^{(j)}$ in the cell is a potential feature representation, and is associated with each previous intermediate node $x^{(i)}$ in the cell through a directed edge operation $o^{(i,j)}$. For a discrete search space, each intermediate node can be expressed as:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}\left(x^{(i)}\right). \tag{3}$$

In DARTS, it makes the discrete search space 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:

$$\bar{o}^{(i,j)}(x) = \sum_{o\in O} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o'\in O}\exp\left(\alpha_{o'}^{(i,j)}\right)} o(x) \tag{4}$$

where O represents a set of candidate operations, $\alpha_o^{(i,j)}$ represents the weight of operation o on directed edge $e^{(i,j)}$. Therefore, the search of the network architecture has evolved into an optimization process for a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$. At the end of the search, the most likely operation $o^{(i,j)}$ on the directed edge $e^{(i,j)}$ will be selected and other operations will be discarded.

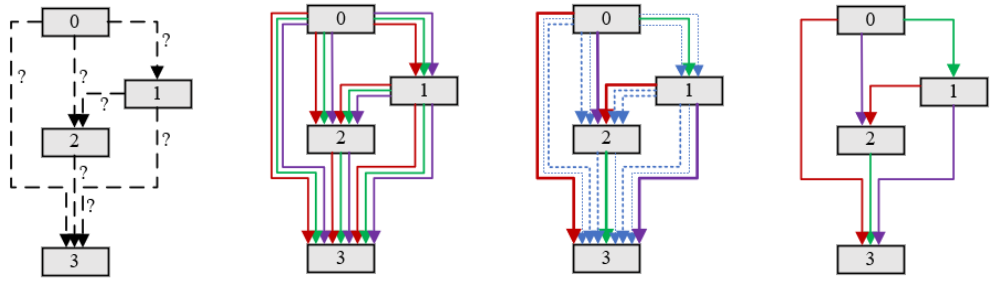
$$o^{(i,j)} = \text{argmax}_{o\in O}\, \alpha_o^{(i,j)} \tag{5}$$

By solving a bilevel optimization problem [66, 67] to jointly optimize the probability of mixed operations (the parameters α of the network architecture) and network weights w:

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}_{val}\left(w^*(\alpha), \alpha\right) \\ \text{s.t.} \quad & w^*(\alpha) = \text{argmin}_w\, \mathcal{L}_{train}(w, \alpha) \end{aligned} \tag{6}$$

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

Compared with DARTS, the search of the network architecture is changed from the selection of discrete candidate operations to the optimization of the probability of continuous mixed operations. During the same period, NAO [73] chose to encode the entire network architecture to map the originally discrete network architecture to continuous embedded encoding. Then, the output of the performance predictor is maximized by the gradient optimization method to obtain the optimal embedded coding. Finally, a decoder is used to discretize the optimal continuous representation (that is, the optimal embedded coding) into the optimal network architecture. In addition, DARTS uses the *argmax* strategy to eliminate the less probable operations among the mixed operations to discretize the network architecture. However, common non-linear problems in network operation introduce bias into the loss function. This bias will exacerbate the performance difference between the derived child networks and the converged parent networks, resulting in the need to retrain the parameters of the derived child networks. Therefore, a NAS solution with less performance deviation between the derived child networks and the converged parent networks is needed. To this end, SNAS [45] starts with the delayed reward of reinforcement learning, and analyzes the reason why delayed reward leads to the slow convergence speed of reinforcement learning when performing architecture search. Therefore, SNAS proposes to remodel the NAS to theoretically bypass the



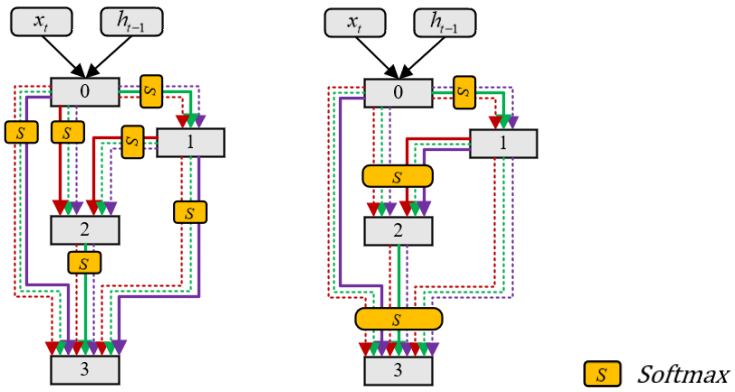
(a) Connection to be determined (b) Continuous relaxation (c) Joint optimization (d) Discrete network architecture

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, 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 network architecture.

problem of delayed rewards for reinforcement learning, and at the same time to continually network parameters, so that network operation parameters and network architecture parameters can be jointly optimized using a gradient method. Based on this, SNAS has proposed a more efficient and automated network architecture search framework, while maintaining the completeness and differentiability of the NAS pipeline.

In the work of SNAS, DARTS and many other NAS [75–78], the feasible paths of the searched network architecture depend on each other and are closely coupled during the search phase. Although SNAS reduces the performance difference between the derived child network and the converged parent network to some extent, SNAS and DARTS still have to choose only one path during the verification phase. This crude decoupling inevitably leads to a gap between network architectures during the search and verification phases. To this end, DATA [74] developed the *Ensemble Gumbel-Softmax* (EGS) estimator. It can decouple the relationship between different paths of the network architecture, and can achieve seamless transfer of gradients between different paths. This solves the problem that the architecture cannot be seamlessly connected between search and verification.

Furthermore, I-DARTS [79] pointed out that the *softmax*-based relaxation constraint between each pair of nodes may cause DARTS to be a "local" model. In DARTS, the middle node of the cell is connected to all the precursor nodes, and when discretizing the network architecture, there is only one directional edge between each pair of nodes. This results in edges from different nodes that cannot 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 limits the size of the DARTS search space. These local decisions caused by the bias problem in graph-based models [80, 81] will limit DARTS to make the optimal architectural choices. Based on this, I-DARTS proposed an interesting and simple idea: use a *softmax* to simultaneously consider all input edges of a given node. We show the cell structure comparison of DARTS and I-DARTS in the recurrent neural network in Fig.7. As shown in Fig.7b, given a node, I-DARTS can determine whether there are edges connected between related nodes according to the importance of all input edges: there are multiple connected edges or no related edges.



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

Fig. 7. Comparison of cell structure of DARTS [17] and I-DARTS [79] in recurrent neural network. (a) In DARTS, edges from different nodes cannot be compared. And when discretizing the network 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 node, I-DARTS can determine whether there are edges connected between related nodes according to the importance of all input edges: there are multiple connected edges or no related edges.

P-DARTS [43] starts with the deep gap between the search and evaluation of network architecture and improves DARTS. In DARTS, due to the limitation of computing resources, DARTS uses a shallow cell stack architecture in the search phase, and in the evaluation phase, stacks more searched cells to process data sets with higher resolution. Therefore, the basic cell of the network architecture used for evaluation is actually designed for shallow architecture, which is different from the deep network architecture used in the evaluation stage. Based on this, P-DARTS proposes to use progressive search to gradually increase the depth of the network during the search phase. And by gradually reducing the candidate operation set according to the weight of the mixed operation in the search process, in order 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 when searching in deep architectures (algorithms are heavily biased towards skip-connect).

Compared with NAS based on RL and EA, although DARTS greatly improves the search efficiency, it is not enough. As shown in Fig.6c, in the search phase, DARTS continuously relaxes the cell's search space and optimizes all parameters in the DAG together. This causes DARTS to occupy too much memory on the device when searching, and the search speed is slow. At the same time, the effects of different operations between the same pair of nodes may be cancelled each other, thereby destroying the entire optimization process. To this end, GDAS [82] proposes to use a differentiable architecture sampler in a training iteration to sample only one subgraph in the DAG, so only one part of the DAG needs to be optimized in one iteration. We show this process in Fig.8. At the same time, the architecture sampler can be optimized using a gradient-based method during the verification phase. Therefore, the search efficiency of GDAS has been further improved.

In order to reduce DARTS's memory usage during search and improve search efficiency, unlike GDAS sampling subgraphs in DAG and training only one subgraph in one iteration, PC-DARTS [83] chooses to start from the channel. In the search process, PC-DARTS samples the channels, and only

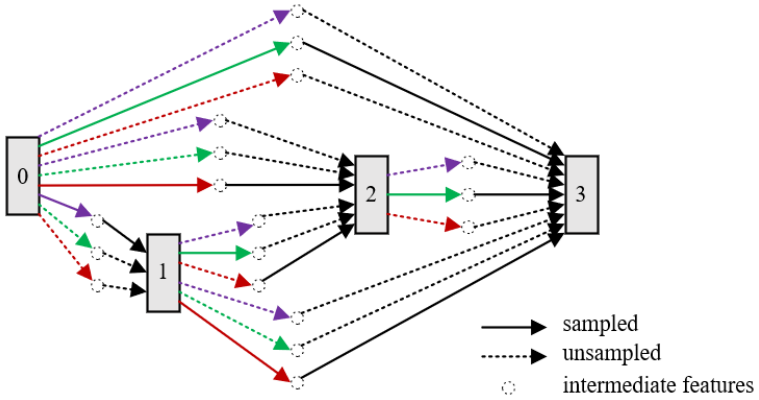


Fig. 8. In GDAS [82], an example of sampling subgraphs in a DAG with 3 intermediate nodes. Colored lines indicate operations in the candidate operation set, and black lines indicate corresponding information flows. The circles indicate the intermediate features after operation processing. Each intermediate node is equal to the sum of all the sampled intermediate features. In one iteration, only one sampled subgraph is trained.

convolves the sampled channel features to achieve efficient gradient optimization. In order to deal with the problem of inconsistent information brought 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. Therefore, PC-DARTS can save memory and is more efficient and stable. [110] recent found that DARTS [17] has a poor test performance for the architecture generated in a wide search space. It believes that when the discovered solutions is consistent with the high verification loss curvature in the architecture space, the discovered architecture is difficult to promote. And by adding various types of regularization to explore how to make DARTS more robust. Finally, [110] proposed several simple variants and achieved good generalization performance. Although we have done many reviews, there are still many improvements based on DARTS [113, 151].

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

3.3 Network Architecture Recycle

The early NAS [11, 12, 15, 16] and much of the subsequent work [17, 38–40] was to search the network architecture from scratch. From a certain perspective, this kind of thinking does increase the freedom of network architecture design, and it is very likely to design 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 network architecture. Because, it does not make full use of the prior knowledge of the existing artificially designed high-performance network architecture. Therefore, a new idea is to use the existing artificially designed high-performance network architecture as a starting point, and use the NAS method to modify or evolve these network architectures, so as to obtain a more promising network architecture at a lower computing cost. This process is generally referred to as network transformation. Net2Net [46] conducted a detailed study of network transformation technology and proposed the function-preserving transformations to

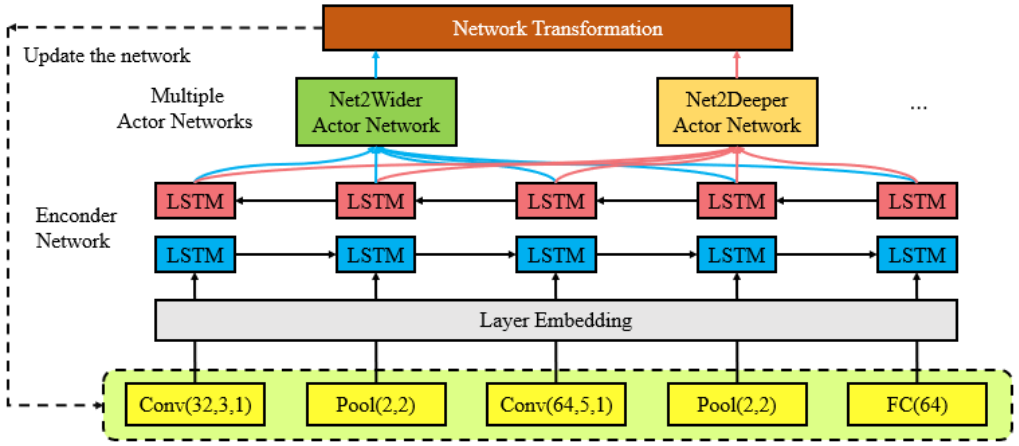


Fig. 9. In EAS [50], architecture search based on network transformation. After the existing network layer is encoded by the layer encoder, Bi-LSTM [47] is used as a meta-controller to learn the low-dimensional feature representation of the network architecture. The multi-action network combines these features to decide whether to adopt the corresponding network transformation operation (deepening layer or widening layer). Finally, reinforcement learning is used to update the meta-controlled parameters.

achieve reuse of model parameters after transformation. It can effectively accelerate the training of new and larger networks.

Based on this idea, [50] proposes *efficient architecture search* (EAS), which use the encoder network as a meta-controller to learn the low-dimensional representation of the existing network architecture, and refer to the multiple actor networks in Net2Net [46] to decide whether to make corresponding adjustments to the network architecture at the layer level (deepening or widening layer). In addition, it uses reinforcement learning strategies to update the parameters in the meta-controller. EAS considers that the network transformation at the layer level needs to combine the information of the entire network architecture, so a *bidirectional recurrent network* (Bi-LSTM) [47] is used as the network encoder. Since EAS is a network transformation on an existing network, the reuse of models and weights can greatly reduce the amount of calculation. We show the overall network architecture of EAS in Fig.9. In Fig.10, we also showed the internal structure of two action networks: Net2Wider and Net2Deeper. In Net2Wider, the action network shares the same *sigmoid* classifier, and decides whether to widen the layer according to each hidden state of the encoder. In Net2Deeper, the action network inputs the state of the final hidden layer of Bi-LSTM into the recurrent network, and the recurrent network decides where to insert the layer and the parameters of the inserted layer.

Contrary to widening or deepening the layers of the existing network in EAS [50], N2N learning [51] considers compressing the teacher network by removing or shrinking the layers. It compresses the teacher network through a two-stage operation selection: first, the layer removal is performed on the macro level, and then the layer shrinkage is performed on the micro level. Use reinforcement learning to explore the search space, and use knowledge distillation [55] to train each generated network architecture. Then learn a locally optimal student network. Using this method, under the condition of similar performance, a compression ratio of more than 10× is achieved for networks such as ResNet-34 [29]. Unlike EAS [50] and N2N learning [51], which can only deepen (remove) and widen (shrink) the network at the layer level, Path-level EAS [56] realizes a network transformation at the path level. The impulse for this idea stems from the performance gains of the multi-branch

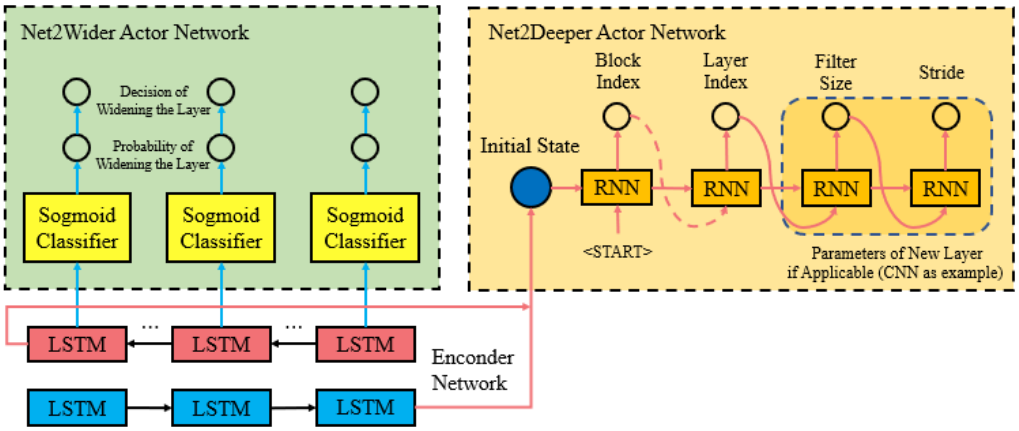


Fig. 10. In EAS [50], the internal structure of two action networks: Net2Wider and Net2Deeper. In Net2Wider, the action network shares the same sigmoid classifier, and decides whether to widen the layer according to each hidden state of the encoder. In Net2Deeper, the action network inputs the state of the final hidden layer of Bi-LSTM into the recurrent network, and the recurrent network decides where to insert the layer and the parameters of the inserted layer.

network architecture included in the manually designed network [29, 30, 57, 58]. It achieves network path-level transformation by replacing a single layer with multi-branch operations with allocation and merge strategies. Allocation strategies include: *replication* and *split*. Merge strategies include: *add* and *concatenation*. We show an example of the process of implementing a path-level network transformation by using a multi-branch operation instead of a single layer in Fig.11. Another similar work NASH-Net [84], which further proposes four network morphism types based on Net2Net [46]. NASH-Net can start from a pre-trained network, apply network morphism to generate a set of sub-networks, and get the best sub-network after a short period of training. Then, starting from the best sub-network, iterate this process using the *Neural Architecture Search by Hill-climbing* (NASH) to get the best network architecture.

For complex tasks such as semantic segmentation or object detection, previous work often used the network designed for image classification as the backbone network. In fact, performance gains can be obtained by designing networks specifically for complex target tasks. Although there are some works [63–65] using NAS to design backbone networks for semantic segmentation or object detection tasks, pre-training still exists and the computational cost is high. *Fast Neural Network Adaptation* (FNA) [59] proposes a method that can adapt the architecture and parameters of a network to new tasks at almost zero cost. It starts from a seed network (manually designed high-performance network), expands it into a super network in its operation set, and then uses the NAS method [17, 19, 60] to adapt the network architecture to obtain the target architecture. And use 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 on the target task. We show this process in Fig.12. It is precisely because of the low cost of FNA in network transformation that it is possible for NAS to design a special network architecture for large-scale tasks such as detection and segmentation.

In this section, we give a comprehensive overview of NAS based on architecture recycle. This method makes it possible to utilize a large number of previously artificially designed high-performance networks. It saves the NAS from having to search the network architecture from

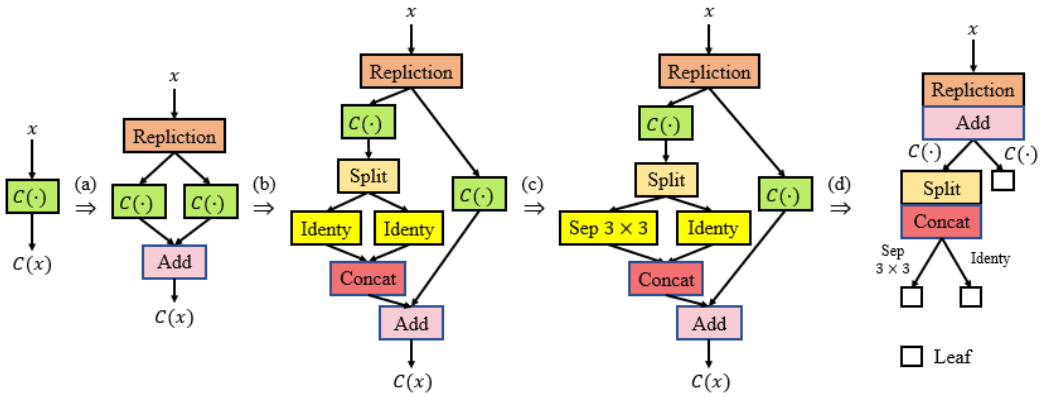


Fig. 11. In Path-level EAS [56], an example of a process for implementing path-level network transformation by using a multi-branch operation instead of a single layer. (a) Use the *replication-add* strategy to add branches to a single layer. (b) Use the *split-concat* strategy to further add branches to the network. (c) Replace identity mapping with a 3×3 depthwise-separable convolution. (d) The tree structure of the network architecture in (c).

scratch, reducing a large number of unnecessary random searches in the massive search space. Compared with other optimization strategies, there are relatively few studies on NAS based on network architecture recycling.

3.4 Incomplete Training

The key technology of NAS is to use a search strategy to find the best network architecture by comparing the performance of a large number of candidate network architectures. Therefore, the performance ranking of candidate network architectures is extremely important. Early NAS [11, 12, 15, 16] usually fully trained the candidate network architecture, and then obtained the ranking of the candidate network architecture based on its performance on the validation set. This method is too time-consuming because there are too many candidate network architectures to compare.

Although they also used some methods to accelerate the ranking of candidate network architectures. For example: NAS-RL [11] uses parallel and asynchronous updates [41] to accelerate the training of candidate network architectures. MetaQNN [12] compares the performance of the candidate network architecture after the first epoch 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 network architecture to inherit the weight of the parent as much as possible, thereby reducing the burden of retraining the candidate network architecture. However, there are still a large number of child networks whose structural changes cannot inherit the weight of their parents after mutation, and these candidate networks will be forced to retrain. Although the above methods also accelerate the training of candidate network architectures to a certain extent, they still require a lot of computing power and the acceleration effect is relatively limited. Therefore, some research that can be used to further accelerate the training of candidate network architectures to obtain their relative ranking is necessary.

3.4.1 Training from scratch? Do we really need to train every candidate network architecture from scratch? The answer is negative.

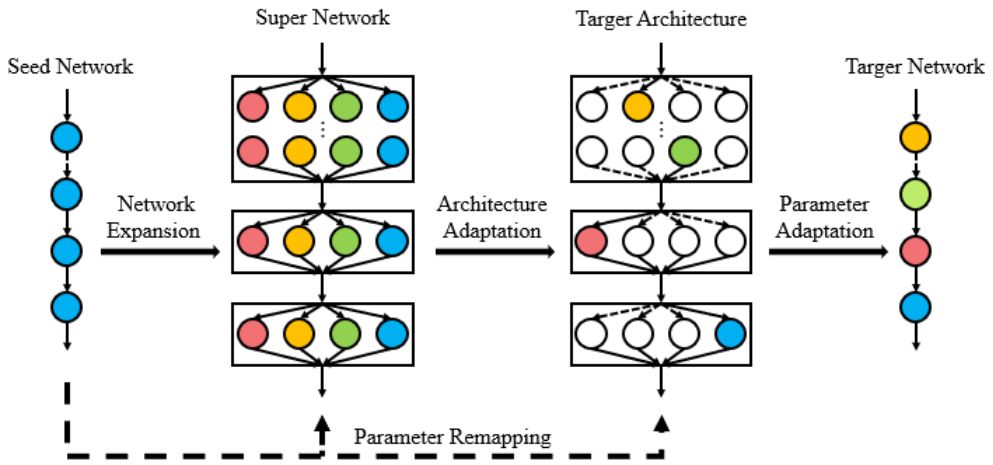


Fig. 12. The overall framework of FNA [59]. It starts from a seed network, expands it into a super network in its operation set, and then uses the NAS method to adapt the network architecture to obtain the target architecture. And use the seed network to map the parameters to the super network and the target network to initialize the parameters. Finally, the target network is obtained by fine-tuning on the target task.

By treating the candidate network architecture as an independent individual, each candidate network architecture is trained from scratch, and the candidate network architecture is ranked according to their performance on the validation set. This may provide a more accurate ranking, as other work has done [11, 31, 33, 36]. In this process, the parameters of each trained candidate network architecture are directly discarded. Obviously these trained parameters have not been fully utilized. So a new idea of parameter sharing has emerged.

ENAS [19] is the first NAS work to explicitly propose parameter sharing. ENAS noted that the candidate network architecture in NAS can be regarded as a directed acyclic subgraph sampled in a supercomputing graph constructed by the search space. We show this sampling process in Fig.13. Based on this observation, ENAS uses LSTM as a controller to search the optimal subgraph on a large computation graph to obtain the neural network architecture. In transfer learning and multi-task learning, the weights obtained by training a model designed for a specific task on a data set are also applicable to other models designed for other tasks [85–87]. Encouraged by this, ENAS proposed to force the sharing of parameters among all different child models (candidate architecture). Through this mechanism, the child models can obtain empirical performance with each other, thereby avoiding the complete training of each child model from scratch. We show an example of different subgraphs sharing weights in Fig.13. The supercomputing graph can be expressed as a DAG. The nodes in the graph are defined as local calculations, and the edges represent the flow of information. Each node has its own corresponding weight parameter, as shown in the upper right of Fig.13. However, the corresponding parameters can only be activated when a specific edge is sampled. The ENAS mechanism allows all subgraphs (that is, candidate network architectures) to share parameters. Therefore, EANS has greatly improved its search efficiency compared to [11, 31, 33, 36]. Subsequently, CAS [54] explored a multi-task architecture search based on ENAS. It extends NAS to transfer learning across data sources, and introduces a novel continuous architecture search to solve this forgetting problem in the continuous learning process. This allows CAS to inherit the experience gained from the previous task when training a new task, so that the model parameters can be continuously trained. This is very beneficial for NAS

research on multi-tasking. Moreover, AutoGAN [53] first introduced NAS into *generative adversarial networks* (GANs) [92] and used *Inception score* (IS) [93] as the reward value of RL to accelerate the search process through parameter sharing ENAS [19] and dynamic-resetting. And use the progressive GAN training [94] to introduce *multi-level architecture search* (MLAS) in AutoGAN, and gradually implement NAS. Compared with the most advanced manual GANs [94–97], AutoGAN is very competitive. The parameter sharing mechanism is also used to accelerate the deployment research of the NAS architecture model in multiple devices and multiple constrained environments. At the kernel-level, OFA [99] uses a elastic kernel mechanism to meet the application needs of multi-platform deployment and the diversity of visual needs of different platforms. Small kernels share the weight of large kernels. In order to avoid the repeated use of centering sub-kernels (centering sub-kernels is used as 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, and the smaller network sharing the weight of the larger network before fine-tuning. The weight of the large network can provides a good initialization for the small network, which greatly accelerates the training efficiency.

In addition, the one-shot based method also uses the idea of parameter sharing. SMASH [23] proposes to train an auxiliary HyperNet [88] and use the auxiliary HyperNet to generate weights for other candidate network architectures. In addition, SMASH also uses the early training performance of different networks according to the research in Hyperband [89] to provide meaningful guidance suggestions for the ranking of candidate network architectures. Parameter sharing is mainly reflected in hypernetwork and between candidate network architectures. The use of the auxiliary HyperNet avoids the complete training of each candidate network architecture. By comparing the performance of the candidate network architectures with weights generated by the HyperNet on the verification set, their relative rankings are obtained. This allows SMASH to quickly obtain the optimal network architecture at the cost of a single training session. Understanding One-Shot Models [22] conducted a comprehensive analysis of the rationality of the parameter sharing ideas 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, and pointed out that a good enough result can be obtained without the hypernetwork and RL controller. Unlike SMASH, which encodes an architecture into a three-dimensional tensor through a memory channel scheme, Graph HyperNetwork (GHN) [90] recommends using computation graphs to represent the network architecture, and then using graph neural networks to perform architecture searches. Compared to SMASH, which can only use hypernetwork to predict some weights, GHN can predict all the free weights through a graph model. Therefore, GHN based on network topology modeling can predict network performance faster and more accurately than SMASH.

A typical one-shot NAS needs to randomly sample a large number of candidate architectures from the hypernetwork with parameter sharing and evaluate them to find the best network architecture [22, 23]. SETN [91] noted that it is extremely difficult to find the best architecture from these sampled candidate architectures. Because, in the relevant NAS [22, 23, 82], the shared parameters are closely coupled with the learnable architectural parameters. These deviations introduced into the template parameters will cause some of the learnable architectural parameters to be more biased towards simple networks (these networks have fewer layers and are more lightweight). Because they converge faster than more complex networks, which leads to a simplified search architecture. At the same time, it also makes the candidate architectures sampled have a very low good rate. To this end, SETN adopts a uniformly stochastic training strategy to treat each candidate architecture fairly, so that they are fully trained to obtain more accurate verification performance. In addition, SETN is also equipped with an estimator for the template architecture. Unlike the

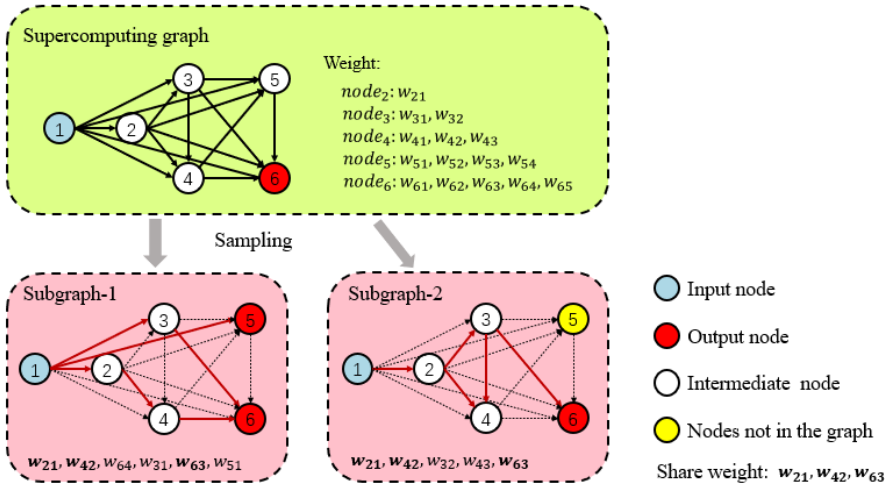


Fig. 13. Parameter sharing mechanism in ENAS [19]. The candidate network architecture in NAS can be viewed as a directed acyclic subgraph sampled in a supercomputing graph constructed by the search space. The nodes in the graph represent local calculations, and the edges represent information flow. Each node has its own corresponding weight parameter, as shown 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 (that is, candidate network architectures) to share parameters.

random sampling methods previously used in Understanding One-Shot Models [22] and SMASH [23], the estimator in SETN can be used to learn the probability that the candidate architecture has a lower verification loss, and 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], so that it is more likely to find the optimal architecture.

[24] through the evaluation of the effectiveness of the NAS search strategy found that the weight sharing strategy in ENAS [19] resulted in inaccurate performance evaluation of the candidate architecture, making it difficult for the NAS to search for the best architecture. In addition, the research of Fairnas [100] and [101] also shows that candidate network architectures based on these parameter sharing often cannot be adequately trained, which will lead to inaccurate ranking of candidate network architectures. In NAS work based on gradient optimization [17, 102, 103], the joint optimization of supernet weights and architectural parameters will also introduce bias between sub-models. To this end, DNA [21] proposed a NAS's large-scale search space for modular, by ensuring that the candidate architecture is adequately trained to reduce the representation shift caused by shared parameters. In addition, DNA [21] also uses block-wise search to evaluate all candidate architectures within the block. Use these methods to evaluate candidate architectures more accurately. GDAS-NSAS [25] also thought and improved the weight sharing mechanism in one-shot NAS, it proposed a NSAS loss function to solve the problem of multi-model forgetting (when weight sharing is used to sequentially train a new network architecture, the performance of the previous network architecture is reduced) caused by weight sharing during the super network training process. Finally, GDAS-NSAS [25] applies the proposed method to RandomNAS [104] and

GDAS [82], which effectively suppresses the multi-model forgetting problem and improves the training quality of the supernet.

Differentiable neural architecture search also uses similar parameter sharing ideas, such as DARTS-like work [17, 43, 79, 83]. For details, refer to Sec.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 phase and only need to optimize the subgraphs to be sampled in each iteration. The difference is that the DARTS-like method chooses to directly optimize a super network, 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. In addition, optimization strategies based on network architecture recycle can often be initialized with the help of function-preserving [46] to inherit the parameters of the template network, thereby avoiding retraining of the sub-network architecture. For details, refer to Sec.3.3. For example: EAS [50], Path-level EAS [56] and N2N learning [51], etc.

3.4.2 Training to convergence? Do we really need to train each candidate network architecture to convergence? The answer is negative.

In order to quickly analyze the effectiveness of the current model in deep learning, human experts can often judge whether the current model is necessary to continue training according to the learning curve of the model. Therefore, the model training with no potential will be terminated as soon as possible, instead of waiting for its convergence to save 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, and for more promising architectures, more adequate training can be obtained.

Early termination of training is not a new idea, many researchers have done a lot of related research. For example: [106] uses the probabilistic method to simulate the learning curve of the deep neural network, and terminates the training of the poorly running model in advance. However, it requires a long early training to accurately simulate and predict the learning curve. [107] extends [106], in [107], 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 are also used to solve hyperparameter optimization problems [89, 108].

The above methods are based on the partially observed early performance to predict the learning curve, and the corresponding machine learning model is designed. In order to imitate human experts so that in NAS search can also automatically identify and terminate training early below standard candidate architectures, [20] combines learning curve prediction with NAS tasks for the first time. It builds a set of standard frequentist regression models, and obtains the corresponding simple features from the network architecture, hyperparameters and early learning curve. Use these features to train the frequentist regression model, and then predict the final verification set performance of the network architecture with early training experience. Performance prediction is also used in PNAS [36]. In order to avoid training and evaluation for all child networks, it learns a predictor function, which can be trained based on the observable early performance of the cell. Then use the predictor to evaluate all candidate cells, and select top-k cells, and repeat this process until a sufficient number of blocks of cells are found. As shown by the black curve in Fig.14, we show an example of such performance prediction based on early performance observations.

NAO [73] uses a performance predictor similar to previous work [20, 36, 109]. Unlike PNAS [36] that uses a performance predictor to evaluate and select the generated network architecture to speeds up the search process. In NAO [73], after the encoder completes the continuous representation of the network architecture, the performance predictor is taken as the optimization goal of gradient ascent. By maximizing the output of the performance predictor f, the continuous

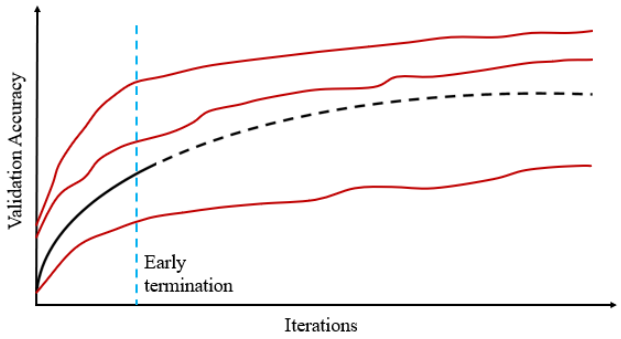


Fig. 14. Example of early termination of training strategy. The black curve shows an attempt to use an observable early performance learning curve (solid line) to predict the final performance of the network architecture on the validation set [20, 36, 109]. The three red curves show examples of hypothesis that the performance of the network architecture during early training and convergence is consistent with respect to ranking [49]. Using performance prediction and early performance ranking assumptions can terminate the training of network architectures that lack potential as early as human experts, and use limited computing resources to explore more potential network architectures. This speeds up the NAS architecture search process.

representation of the best network architecture is obtained. Finally, use the decoder to get the final discrete network architecture. Unlike previous NAS based on performance prediction [20, 36, 109], in multinomial distribution learning for NAS, MdeNAS [49] proposed a performance ranking hypothesis, that is, the relative performance ranking of the network architecture at each training stage is consistent. In other words, the network architecture that performed well in the early days still has good performance when the training converges. MdeNAS [49] has done a lot of experiments to verify this hypothesis, according to which the early performance of candidate architectures can be used to quickly and easily obtain their relative performance rankings, thereby speeding up the search process of the network architecture. We show an early performance ranking hypothesis in Fig.14 (three red curves).

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

4 PERFORMANCE COMPARISON

NAS is a very promising study. In Section 2, we analyzed the common characteristics of early NAS and summarized the challenges facing early NAS to be widely used. In Section 3, we conduct a comprehensive and systematic review of the solutions that our existing work has taken to address these challenges. In this section, we will classify and compare the performance of existing NAS based on mainstream search methods [26, 27], and at the same time report the optimization strategies they use according to Section 3. These search methods mainly include: *reinforcement learning* (RL), *evolutionary algorithm* (EA), *gradient optimization* (GO), *random search* (RS) and *sequential model-based optimization* (SMBO). We look forward to obtaining their similarities and differences from these summaries.

Things are not as simple as we think. In fact, it is relatively difficult to compare NAS performance because NAS lacks some baselines. In addition, the preprocessing, hyperparameters, search space,

trick, etc. between different NAS are different, which exacerbates the difficulty of NAS performance comparison. For example, learning rate decay, regularization (for example, DropPath [31, 134]), augmenting techniques (for example, Cutout [129]), etc. The random search strategy is considered to be a strong baseline. For example, in [24], random search found the best RNN cells compared to other strategies. The findings in [127], [33] and [104] also prove this. Therefore, [127] proposes to use the average architecture of the random search strategy as the baseline for comparison.

We summarize the performance of the state-of-the-art NAS and mainstream artificial networks on the CIFAR-10 dataset in Table 1. At the same time, based on Section 3, we also reported the optimization strategy used in NAS. Similarly, we report the performance comparison on the ImageNet dataset in Table 2. Because the optimization strategies used in the same NAS method are the same, we therefore omit the reporting of the corresponding optimization strategies in Table 2.

By observing Table 1 and 2, we clearly know that in popular NAS, the use of modular search strategies is extremely extensive. This is mainly because compared to global search, modular search greatly reduces the complexity of the search space. However, as summarized in Section 3.1, this does not mean that the modular search strategy must be better than the global search. Moreover, we can say with certainty that incomplete training is also widely used, which can effectively accelerate the ranking of candidate network architectures, thereby reducing the search duration. In addition, among many optimization strategies, the gradient optimization based on the continuous search space can greatly reduce the search cost, and has a very rich research. We also observe that NAS based on random search strategy has also achieved extremely competitive performance. However, it is clear that NAS research on random search strategies is relatively inadequate. The optimization strategy of architecture recycle also has relatively 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 (ImageNet). Therefore, the search time costs used in the two datasets shown in Table 1 and 2 are the same [17, 31, 38, 45]. These search tasks in advance on small datasets are often called agent tasks. ProxylessNAS [103] also studies the problem of searching directly on large target tasks without agents.

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

5 FUTURE DIRECTIONS

The emergence of NAS is exciting, it is expected to end the tedious trial and error process of manually designing network architecture. And, it is also hoped to design a network structure that is completely different from the artificial network, so as to break through the existing human mindset. Artificially designed networks have made breakthroughs in many fields, including image recognition [4, 7, 112], machine translation [1, 3, 141, 149], semantic segmentation [142, 143, 148], object detection [8, 146, 147], and video understanding [144, 145] and so on. Although the research related to NAS has been quite rich, compared with the rapid development of artificial network architecture design in various fields, NAS is still in the preliminary stage of research. The current NAS is mainly focused on improving the classification accuracy of images, compressing the search time of the network architecture, and making it as civilian as possible. In this process, a proper baseline is crucial, which helps to avoid NAS search strategy research being masked in 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

Table 1. On CIFAR-10, the performance comparison between the state-of-the-art NAS and mainstream artificial networks. For 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 search strategy used in the NAS was reported based on Section 3. Cutout is an augment technology in [129].

Search method	Reference	Venue	Optimization Strategy				Error Acc (%)	Params (Millions)	GPU Days
			Modular search strategy	Continuous search space	Architecture recycle	Incomplete training			
Human	WRN [131]	CVPR16					3.87	36.2	-
	Shark [132]	CoRR17					3.55	2.9	-
	PyramidSepDrop + ShakeDrop[133]	CoRR16					2.67	26.2	-
	ResNet [134]	ECCV16					6.41	1.7	-
	Fractalnet [135]	ICLR17					5.22	38.6	-
	DenseNet-BC [35]	CVPR17					3.46	25.6	-
RL	NAS-RL [11]	ICLR17					3.65	37.4	22,400
	MetaQNN [12]	ICLR17					6.92	11.18	100
	EAS [50]	AAAI18			✓	✓	4.23	23.4	10
	NASNet-A [31]	CVPR18	✓				3.41	3.3	2,000
	NASNet-A + Cutout [31]	CVPR18	✓				2.65	3.3	2,000
	Block-QNN [32]	CVPR18	✓				3.54	39.8	96
	Path-level EAS [56]	ICML18			✓	✓	2.99	5.7	200
	Path-level EAS + Cutout [56]	ICML18			✓	✓	2.49	5.7	200
	N2N learning [51]	ICLR18			✓	✓	6.46	3.87	2.1
	ProxylessNAS-R + Cutout [103]	ICLR19				✓	2.30	5.8	N/A
	FPNAS + Cutout [38]	ICCV19	✓			✓	3.01	5.76	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]	GECCO17					5.98	1.7	14.9
	Hierarchical-EAS [33]	ICLR18	✓				3.75	15.7	300
	NASH-Net [84]	ICLR18			✓	✓	5.20	19.7	1
	Neuro-Cell-based Evolution + Cutout [128]	ECML PKDD18	✓		✓	✓	3.57	5.8	0.5
	AmoebaNet [42]	AAAI19	✓				3.34	3.2	3,150
GO	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
	DARTS (1^{st} order) + Cutout [17]	ICLR19	✓	✓		✓	3.0	3.3	1.5
	DARTS (2^{nd} order) + Cutout [17]	ICLR19	✓	✓		✓	2.76	3.3	4
	SNAS + Cutout [45]	ICLR19	✓	✓		✓	2.85	2.8	1.5
	PARSEC + Cutout [130]	CoRR19	✓	✓		✓	2.81	3.7	1
	GHN [90]	ICLR19	✓	✓		✓	2.84	5.7	0.84
	ProxylessNAS-G + Cutout [103]	ICLR19		✓		✓	2.08	5.7	N/A
	BayesNAS [136]	ICML19	✓	✓		✓	2.81	3.4	0.2
	P-DARTS + Cutout [43]	ICCV19	✓	✓		✓	2.50	3.4	0.3
	DATA + Cutout [74]	NeurIPS19	✓	✓		✓	2.59	3.4	1
	SGAS [151]	CVPR20	✓	✓		✓	2.66	3.7	0.25
	GDAS-NSAS [25]	CVPR20	✓	✓		✓	2.73	3.54	0.4
	PC-DARTS + Cutout [83]	CVPR20	✓	✓		✓	2.57	3.6	0.1
RS	Hierarchical-EAS Random [33]	ICLR18	✓				3.91	N/A	300
	NAO Random-WS [73]	NeurIPS18	✓			✓	3.92	3.9	0.3
	NASH-Net Random [84]	ICLR18					6.5	4.4	0.2
	DARTS Random [17]	ICLR19	✓				3.29	3.2	4
	RandomNAS + Cutout [104]	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
	PNAS [36]	ECCV18	✓			✓	3.41	3.2	225
	NAO [73]	NeurIPS18	✓	✓		✓	2.98	28.6	200
	NAO-WS [73]	NeurIPS18	✓	✓		✓	3.53	2.5	0.3
	NAO-Cutout [73]	NeurIPS18	✓	✓		✓	2.11	128	200

Table 2. On ImageNet, the performance comparison between the state-of-the-art NAS and mainstream artificial networks. The search strategy adopted by the corresponding NAS is consistent with Table 1. Cutout is an augment technology in [129].

Search mothed	Reference	Venue	Top 1/Top 5 Acc(%)	Params (Millions)	Image Size (squarred)	GPU Days
Human	Mobilenets [6]	CoRR17	70.6/89.5	4.2	224	-
	ResNeXt [138]	CVPR17	80.9/95.6	83.6	320	-
	Polynet [139]	CVPR17	81.3/95.8	92.0	331	-
	DPN [140]	NIPS17	81.5/95.8	79.5	320	-
	Shufflenet [137]	CVPR18	70.9/89.8	5.0	224	-
RL	NASNet [31]	CVPR18	82.7/96.2	88.9	331	2,000
	NASNet-A [31]	CVPR18	74.0/91.6	5.3	224	2,000
	Block-QNN [32]	CVPR18	77.4/93.5	N/A	224	96
	Path-level EAS [56]	ICML18	74.6/91.9	594	224	200
	FPNAS [38]	ICCV19	73.3/N/A	3.41	224	0.8
EA	GeNet [16]	ICCV17	72.1/90.4	156	224	17
	Hierarchical-EAS [33]	ICLR18	79.7/94.8	64.0	299	300
	AmoebaNet [42]	AAAI19	82.8/96.1	86.7	331	3,150
	AmoebaNet [42]	AAAI19	83.9/96.6	469	331	3,150
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
	PARSEC [130]	CoRR19	74.0/91.6	5.6	N/A	1
	DARTS [17]	ICLR19	73.3/91.3	4.7	224	4
	SNAS [45]	ICLR19	72.7/90.8	4.3	224	1.5
	ProxylessNAS [103]	ICLR19	75.1/92.5	N/A	224	8.33
	GHN [90]	ICLR19	73.0/91.3	6.1	224	0.84
	SGAS [151]	CVPR20	75.62/92.6	5.4	224	0.25
	PC-DARTS (CIFAR10) [83]	ICLR20	74.9/92.2	5.3	224	0.1
	PC-DARTS (ImageNet) [83]	ICLR20	75.8/92.7	5.3	224	3.8
RS	Hierarchical-EAS Random [33]	ICLR18	79.0/94.8	N/A	299	300
SMBO	PNAS [36]	ECCV18	74.2/91.9	5.1	224	225
	PNAS [36]	ECCV18	82.9/96.2	86.1	331	225

supernets, and there are many theoretical deficiencies in related research. Therefore, research related to it is of great benefit to the development of NAS.

Early NAS was very close to people's expectations for automatic network architecture design. For example, Large-scale Evolution [15] uses an EA as a NAS search strategy, and emphasizes that once network evolution begins, there is no need for manual participation. In the case of reducing human interference as much as possible, let the algorithm autonomously determine the evolution direction of the network. This is a good start, although the search performance at the time was not very prominent. This is mainly because Large-scale Evolution emphasizes to reduce artificial restrictions as much as possible, and evolve from the simplest single-layer network (as shown in Fig.3), which requires that the population contains a sufficient number of individuals to evolve satisfactory results. This means that the evolution process consumes a lot of computing resources, which relatively increases the possibility of evolving a network structure that is free from human inherent thinking. However, due to the limitations of evolutionary efficiency and huge search space, it is difficult to evolve a network structure with outstanding performance. Therefore, subsequent NAS began to discuss how to reduce the search space as much as possible while improving network performance. NASNet [31] benefited from the idea of artificial network architecture design [4, 29, 30], and proposed a modular search strategy that was widely adopted later. But obviously this comes at the expense of the freedom of network architecture design. Although this modular search strategy does greatly reduce the search cost, in reality we are not sure whether a better network architecture has been ignored in the process of turning global search to modular search. This is also the main reason why it is not certain that modular search is better than global search, and related research is also lacking. In addition, the freedom of network architecture design and the search cost are

a contradiction. How to balance the two and obtain good performance is an important research direction in the future.

In addition, RobNet [150] proposes to use the NAS method to generate a large number of network architectures by analyzing the differences between strong and poorly performing architectures to find out which network structures help to obtain a robust network architecture. Inspired by this, a feasible idea is to use a similar idea to RobNet in a search space with a higher degree of freedom, analyze the common structural features between promising architectures and increase their structural proportion in the search space. On the contrary, it reduces the proportion of the common structural features of the architecture with poor performance in the search space, so as to gradually reduce the search space in stages. Let the algorithm autonomously decide which network structures to remove or add instead of artificially imposing constraints.

As analyzed in Section 4, another widely criticized issue 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. Although RS has proven to be a strong baseline [24, 33, 104, 127], relevant research is still insufficient. [24] pointed out that the performance of the current optimal NAS algorithm is even only similar to the random strategy, which should arouse researchers' vigilance. Therefore, more ablation experiments are needed, and researchers need to pay more attention to which part of the NAS design leads to performance gains. Blindly stacking tricks to increase performance should be criticized. Therefore, relevant theoretical analysis and reflection are crucial to the future development of NAS. Another issue that needs attention is the widely used parameter sharing strategy [19], although it effectively improves NAS search efficiency. But more and more evidence shows that the parameter sharing strategy is likely to result in a sub-optimal inaccurate candidate architecture ranking [21, 24, 25, 99]. This will make it almost impossible for NAS to find the optimal network architecture in the search space. Therefore, relevant theoretical research and improvements are expected. In addition, the current NAS is mainly focused on improving the accuracy of image classification and reducing the search cost. NAS will play a greater role in areas that require complex network architecture design, such as multi-object architecture search, network transformation using NAS based on existing models, model compression, target detection and segmentation and so on.

In short, the emergence of NAS is exciting. At present, NAS is still in the initial stage of development, and more theoretical guidance and experimental analysis are needed. Finding out which NAS designs lead to improved performance is critical to improving NAS. Want to use NAS to completely replace the design of artificial network architecture requires more research and a more solid theoretical basis. It takes a long way.

ACKNOWLEDGMENTS

This work was partially supported by Australian Research Council Discovery Early Career Researcher Award (DE190100626).

REFERENCES

[1] Hochreiter, S., & Schmidhuber, J. (1997). Lonort-term memory. Neural computation, 9(8), 1735-1780.
[2] Chen, M. X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., ... & Wu, Y. (2018). The best of both worlds: Combining recent advances in neural machine translation. arXiv preprint arXiv:1804.09849.
[3] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... & Klingner, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144.
[4] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
[5] Suganuma, M., Shirakawa, S., & Nagao, T. (2017, July). A genetic programming approach to designing convolutional neural network architectures. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 497-504).

[6] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.
[7] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).
[8] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).
[9] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.
[10] Lin, T. Y., Goyal, P., Girshick, R., He, K., & DollÃąr, P. (2017). Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision (pp. 2980-2988).
[11] Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.
[12] Baker, B., Gupta, O., Naik, N., & Raskar, R. (2016). Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167.
[13] Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05) (Vol. 1, pp. 886-893). IEEE.
[14] Lowe, D. G. (1999, September). Object recognition from local scale-invariant features. In Proceedings of the seventh IEEE international conference on computer vision (Vol. 2, pp. 1150-1157). Ieee.
[15] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., ... & Kurakin, A. (2017, August). Large-scale evolution of image classifiers. In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 2902-2911). JMLR. org.
[16] Xie, L., & Yuille, A. (2017). Genetic cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1379-1388).
[17] Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055.
[18] Shu, Y., Wang, W., & Cai, S. (2019). Understanding Architectures Learnt by Cell-based Neural Architecture Search. arXiv preprint arXiv:1909.09569.
[19] Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., & Dean, J. (2018). Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268.
[20] Baker, B., Gupta, O., Raskar, R., & Naik, N. (2017). Accelerating neural architecture search using performance prediction. arXiv preprint arXiv:1705.10823.
[21] Li, C., Peng, J., Yuan, L., Wang, G., Liang, X., Lin, L., & Chang, X. (2019). Blockwisely Supervised Neural Architecture Search with Knowledge Distillation. arXiv preprint arXiv:1911.13053.
[22] Bender, G. (2019). Understanding and simplifying one-shot architecture search.
[23] Brock, A., Lim, T., Ritchie, J. M., & Weston, N. (2017). Smash: one-shot model architecture search through hypernetworks. arXiv preprint arXiv:1708.05344.
[24] Sciuto, C., Yu, K., Jaggi, M., Musat, C., & Salzmann, M. (2019). Evaluating the search phase of neural architecture search. arXiv preprint arXiv:1902.08142.
[25] Zhang, M., Li, H., Pan, S., Chang, X., & Su, S. Overcoming Multi-Model Forgetting in One-Shot NAS with Diversity Maximization.
[26] Elsken, T., Metzen, J. H., & Hutter, F. (2018). Neural architecture search: A survey. arXiv preprint arXiv:1808.05377.
[27] Wistuba, M., Rawat, A., & Pedapati, T. (2019). A survey on neural architecture search. arXiv preprint arXiv:1905.01392.
[28] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2820-2828).
[29] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
[30] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).
[31] Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 8697-8710).
[32] Zhong, Z., Yan, J., Wu, W., Shao, J., & Liu, C. L. (2018). Practical block-wise neural network architecture generation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2423-2432).
[33] Liu, H., Simonyan, K., Vinyals, O., Fernando, C., & Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436.
[34] Dong, J. D., Cheng, A. C., Juan, D. C., Wei, W., & Sun, M. (2018). Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 517-531).

[35] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).
[36] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L. J., ... & Murphy, K. (2018). Progressive neural architecture search. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 19-34).
[37] Saikia, T., Marrakchi, Y., Zela, A., Hutter, F., & Brox, T. (2019). AutoDispNet: Improving Disparity Estimation With AutoML. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1812-1823).
[38] Cui, J., Chen, P., Li, R., Liu, S., Shen, X., & Jia, J. (2019). Fast and Practical Neural Architecture Search. In Proceedings of the IEEE International Conference on Computer Vision (pp. 6509-6518).
[39] Xiong, Y., Mehta, R., & Singh, V. (2019). Resource Constrained Neural Network Architecture Search: Will a Submodularity Assumption Help?. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1901-1910).
[40] Ryoo, M. S., Piergiovanni, A. J., Tan, M., & Angelova, A. (2019). Assemblenet: Searching for multi-stream neural connectivity in video architectures. arXiv preprint arXiv:1905.13209.
[41] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., ... & Le, Q. V. (2012). Large scale distributed deep networks. In Advances in neural information processing systems (pp. 1223-1231).
[42] Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019, July). Regularized evolution for image classifier architecture search. In Proceedings of the aaai conference on artificial intelligence (Vol. 33, pp. 4780-4789).
[43] Chen, X., Xie, L., Wu, J., & Tian, Q. (2019). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1294-1303).
[44] Piergiovanni, A. J., Angelova, A., Toshev, A., & Ryoo, M. S. (2019). Evolving space-time neural architectures for videos. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1793-1802).
[45] Xie, S., Zheng, H., Liu, C., & Lin, L. (2018). SNAS: stochastic neural architecture search. arXiv preprint arXiv:1812.09926.
[46] Chen, T., Goodfellow, I., & Shlens, J. (2015). Net2net: Accelerating learning via knowledge transfer. arXiv preprint arXiv:1511.05641.
[47] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. IEEE transactions on Signal Processing, 45(11), 2673-2681.
[48] Bashivan, P., Tensen, M., & DiCarlo, J. J. (2019). Teacher guided architecture search. In Proceedings of the IEEE International Conference on Computer Vision (pp. 5320-5329).
[49] Zheng, X., Ji, R., Tang, L., Zhang, B., Liu, J., & Tian, Q. (2019). Multinomial Distribution Learning for Effective Neural Architecture Search. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1304-1313).
[50] Cai, H., Chen, T., Zhang, W., Yu, Y., & Wang, J. (2018, April). Efficient architecture search by network transformation. In Thirty-Second AAAI conference on artificial intelligence.
[51] Ashok, A., Rhinehart, N., Beainy, F., & Kitani, K. M. (2017). N2n learning: Network to network compression via policy gradient reinforcement learning. arXiv preprint arXiv:1709.06030.
[52] Mei, J., Li, Y., Lian, X., Jin, X., Yang, L., Yuille, A., & Yang, J. (2019). AtomNAS: Fine-Grained End-to-End Neural Architecture Search. arXiv preprint arXiv:1912.09640.
[53] Gong, X., Chang, S., Jiang, Y., & Wang, Z. (2019). Autogan: Neural architecture search for generative adversarial networks. In Proceedings of the IEEE International Conference on Computer Vision (pp. 3224-3234).
[54] Pasunuru, R., & Bansal, M. (2019). Continual and Multi-Task Architecture Search. arXiv preprint arXiv:1906.05226.
[55] Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.
[56] Cai, H., Yang, J., Zhang, W., Han, S., & Yu, Y. (2018). Path-level network transformation for efficient architecture search. arXiv preprint arXiv:1806.02639.
[57] Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017, February). Inception-v4, inception-resnet and the impact of residual connections on learning. In Thirty-first AAAI conference on artificial intelligence.
[58] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).
[59] Fang, J., Sun, Y., Peng, K., Zhang, Q., Li, Y., Liu, W., & Wang, X. (2020). Fast Neural Network Adaptation via Parameter Remapping and Architecture Search. arXiv preprint arXiv:2001.02525.
[60] Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019, July). Regularized evolution for image classifier architecture search. In Proceedings of the aaai conference on artificial intelligence (Vol. 33, pp. 4780-4789).
[61] Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., & Xing, E. P. (2018). Neural architecture search with bayesian optimisation and optimal transport. In Advances in Neural Information Processing Systems (pp. 2016-2025).
[62] Negrinho, R., & Gordon, G. (2017). Deeparchitect: Automatically designing and training deep architectures. arXiv preprint arXiv:1704.08792.
[63] Liu, C., Chen, L. C., Schroff, F., Adam, H., Hua, W., Yuille, A. L., & Fei-Fei, L. (2019). Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 82-92).

[64] Zhang, Y., Qiu, Z., Liu, J., Yao, T., Liu, D., & Mei, T. (2019). Customizable architecture search for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 11641-11650).
[65] Chen, Y., Yang, T., Zhang, X., Meng, G., Pan, C., & Sun, J. (2019). Detnas: Neural architecture search on object detection. arXiv preprint arXiv:1903.10979.
[66] Anandalingam, G., & Friesz, T. L. (1992). Hierarchical optimization: An introduction. Annals of Operations Research, 34(1), 1-11.
[67] Colson, B., Marcotte, P., & Savard, G. (2007). An overview of bilevel optimization. Annals of operations research, 153(1), 235-256.
[68] Shin, R., Packer, C., & Song, D. (2018). Differentiable neural network architecture search.
[69] Ahmed, K., & Torresani, L. (2018). Maskconnect: Connectivity learning by gradient descent. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 349-365).
[70] Saxena, S., & Verbeek, J. (2016). Convolutional neural fabrics. In Advances in Neural Information Processing Systems (pp. 4053-4061).
[71] Ahmed, K., & Torresani, L. (2017). Connectivity learning in multi-branch networks. arXiv preprint arXiv:1709.09582.
[72] Veniat, T., & Denoyer, L. (2018). Learning time/memory-efficient deep architectures with budgeted super networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3492-3500).
[73] Luo, R., Tian, F., Qin, T., Chen, E., & Liu, T. Y. (2018). Neural architecture optimization. In Advances in neural information processing systems (pp. 7816-7827).
[74] Chang, J., Guo, Y., MENG, G., XIANG, S., & Pan, C. (2019). DATA: Differentiable ArchiTecture Approximation. In Advances in Neural Information Processing Systems (pp. 874-884).
[75] Ghiasi, G., Lin, T. Y., & Le, Q. V. (2019). Nas-fpn: Learning scalable feature pyramid architecture for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 7036-7045).
[76] Tran, D., Ray, J., Shou, Z., Chang, S. F., & Paluri, M. (2017). Convnet architecture search for spatiotemporal feature learning. arXiv preprint arXiv:1708.05038.
[77] Chen, L. C., Collins, M., Zhu, Y., Papandreou, G., Zoph, B., Schroff, F., ... & Shlens, J. (2018). Searching for efficient multi-scale architectures for dense image prediction. In Advances in neural information processing systems (pp. 8699-8710).
[78] Ying, C., Klein, A., Real, E., Christiansen, E., Murphy, K., & Hutter, F. (2019). Nas-bench-101: Towards reproducible neural architecture search. arXiv preprint arXiv:1902.09635.
[79] Jiang, Y., Hu, C., Xiao, T., Zhang, C., & Zhu, J. (2019, November). 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) (pp. 3576-3581).
[80] Lafferty, J., McCallum, A., & Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.
[81] Koller, D., & Friedman, N. (2009). Probabilistic graphical models: principles and techniques. MIT press.
[82] Dong, X., & Yang, Y. (2019). Searching for a robust neural architecture in four gpu hours. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1761-1770).
[83] Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G. J., Tian, Q., & Xiong, H. (2019, September). PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In International Conference on Learning Representations.
[84] Elsken, T., Metzen, J. H., & Hutter, F. (2017). Simple and efficient architecture search for convolutional neural networks. arXiv preprint arXiv:1711.04528.
[85] Sharif Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops (pp. 806-813).
[86] Zoph, B., Yuret, D., May, J., & Knight, K. (2016). Transfer learning for low-resource neural machine translation. arXiv preprint arXiv:1604.02201.
[87] Luong, M. T., Le, Q. V., Sutskever, I., Vinyals, O., & Kaiser, L. (2015). Multi-task sequence to sequence learning. arXiv preprint arXiv:1511.06114.
[88] Ha, D., Dai, A., & Le, Q. V. (2016). Hypernetworks. arXiv preprint arXiv:1609.09106.
[89] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2016). Hyperband: Bandit-based configuration evaluation for hyperparameter optimization.(2016).
[90] Zhang, C., Ren, M., & Urtasun, R. (2018). Graph hypernetworks for neural architecture search. arXiv preprint arXiv:1810.05749.
[91] Dong, X., & Yang, Y. (2019). One-shot neural architecture search via self-evaluated template network. In Proceedings of the IEEE International Conference on Computer Vision (pp. 3681-3690).
[92] Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784.

[93] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training gans. In Advances in neural information processing systems (pp. 2234-2242).
[94] Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196.
[95] Tran, N. T., Bui, T. A., & Cheung, N. M. (2018). Dist-gan: An improved gan using distance constraints. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 370-385).
[96] Wang, W., Sun, Y., & Halgamuge, S. (2018). Improving mmd-gan training with repulsive loss function. arXiv preprint arXiv:1812.09916.
[97] Hoang, Q., Nguyen, T. D., Le, T., & Phung, D. (2018). MGAN: Training generative adversarial nets with multiple generators.
[98] Ghiasi, G., Lin, T. Y., & Le, Q. V. (2019). Nas-fpn: Learning scalable feature pyramid architecture for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 7036-7045).
[99] Cai, H., Gan, C., & Han, S. (2019). Once for all: Train one network and specialize it for efficient deployment. arXiv preprint arXiv:1908.09791.
[100] Chu, X., Zhang, B., Xu, R., & Li, J. (2019). Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. arXiv preprint arXiv:1907.01845.
[101] Li, X., Lin, C., Li, C., Sun, M., Wu, W., Yan, J., & Ouyang, W. (2019). Improving One-shot NAS by Suppressing the Posterior Fading. arXiv preprint arXiv:1910.02543.
[102] Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., ... & Keutzer, K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 10734-10742).
[103] Cai, H., Zhu, L., & Han, S. (2018). Proxylessnas: Direct neural architecture search on target task and hardware. arXiv preprint arXiv:1812.00332.
[104] Li, L., & Talwalkar, A. (2019). Random search and reproducibility for neural architecture search. arXiv preprint arXiv:1902.07638.
[105] Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., & Sun, J. (2019). Single path one-shot neural architecture search with uniform sampling. arXiv preprint arXiv:1904.00420.
[106] Domhan, T., Springenberg, J. T., & Hutter, F. (2015, June). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Twenty-Fourth International Joint Conference on Artificial Intelligence.
[107] Klein, A., Falkner, S., Springenberg, J. T., & Hutter, F. (2016). Learning curve prediction with Bayesian neural networks.
[108] Chandrashekaran, A., & Lane, I. R. (2017, September). 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 (pp. 477-492). Springer, Cham.
[109] Deng, B., Yan, J., & Lin, D. (2017). Peephole: Predicting network performance before training. arXiv preprint arXiv:1712.03351.
[110] Zela, A. , Elsken, T. , Saikia, T. , Marrakchi, Y. , Brox, T. , & Hutter, F. . (2019). Understanding and robustifying differentiable architecture search.
[111] Peng, J., Sun, M., ZHANG, Z. X., Tan, T., & Yan, J. (2019). Efficient Neural Architecture Transformation Search in Channel-Level for Object Detection. In Advances in Neural Information Processing Systems (pp. 14290-14299).
[112] Zhu, Y., Zhuang, F., Wang, J., Ke, G., Chen, J., Bian, J., ... & He, Q. (2020). Deep Subdomain Adaptation Network for Image Classification. IEEE Transactions on Neural Networks and Learning Systems.
[113] Nayman, N., Noy, A., Ridnik, T., Friedman, I., Jin, R., & Zelnik, L. (2019). Xnas: Neural architecture search with expert advice. In Advances in Neural Information Processing Systems (pp. 1975-1985).
[114] Cao, S., Wang, X., & Kitani, K. M. (2019). Learnable embedding space for efficient neural architecture compression. arXiv preprint arXiv:1902.00383.
[115] Elsken, T., Metzen, J. H., & Hutter, F. (2018). Efficient multi-objective neural architecture search via lamarckian evolution. arXiv preprint arXiv:1804.09081.
[116] Li, X., Zhou, Y., Pan, Z., & Feng, J. (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 (pp. 9145-9153).
[117] Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., ... & Vajda, P. (2019). Chamnet: Towards efficient network design through platform-aware model adaptation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 11398-11407).
[118] Liang, F., Lin, C., Guo, R., Sun, M., Wu, W., Yan, J., & Ouyang, W. (2019). Computation Reallocation for Object Detection. arXiv preprint arXiv:1912.11234.
[119] Fedorov, I., Adams, R. P., Mattina, M., & Whatmough, P. (2019). Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. In Advances in Neural Information Processing Systems (pp. 4978-4990).

[120] Nekrasov, V., Chen, H., Shen, C., & Reid, I. (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 (pp. 9126-9135).
[121] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2019). Autoaugment: Learning augmentation strategies from data. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 113-123).
[122] Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946.
[123] Zhang, X., Wang, Q., Zhang, J., & Zhong, Z. (2019). Adversarial AutoAugment. arXiv preprint arXiv:1912.11188.
[124] Dong, X., & Yang, Y. (2019). Network pruning via transformable architecture search. In Advances in Neural Information Processing Systems (pp. 759-770).
[125] Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., & Banzhaf, W. (2018). Nsga-net: A multi-objective genetic algorithm for neural architecture search.
[126] Dong, X., & Yang, Y. (2020). NAS-Bench-102: Extending the Scope of Reproducible Neural Architecture Search. arXiv preprint arXiv:2001.00326.
[127] Yang, A., EsperanÃğa, P. M., & Carlucci, F. M. (2019). NAS evaluation is frustratingly hard. arXiv preprint arXiv:1912.12522.
[128] Wistuba, M. (2018, September). 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 (pp. 243-258). Springer, Cham.
[129] DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552.
[130] Casale, F.P., Gordon, J., & Fusi, N. (2019). Probabilistic Neural Architecture Search. ArXiv, abs/1902.05116.
[131] Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. arXiv preprint arXiv:1605.07146.
[132] Gastaldi, X. (2017). Shake-shake regularization. arXiv preprint arXiv:1705.07485.
[133] Yamada, Y., Iwamura, M., & Kise, K. (2016). Deep pyramidal residual networks with separated stochastic depth. arXiv preprint arXiv:1612.01230.
[134] Huang, G., Sun, Y., Liu, Z., Sedra, D., & Weinberger, K. Q. (2016, October). Deep networks with stochastic depth. In European conference on computer vision (pp. 646-661). Springer, Cham.
[135] Larsson, G., Maire, M., & Shakhnarovich, G. (2016). Fractalnet: Ultra-deep neural networks without residuals. arXiv preprint arXiv:1605.07648.
[136] Zhou, H., Yang, M., Wang, J., & Pan, W. (2019). Bayesnas: A bayesian approach for neural architecture search. arXiv preprint arXiv:1905.04919.
[137] Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 6848-6856).
[138] Xie, S., Girshick, R., DollÃąr, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1492-1500).
[139] Zhang, X., Li, Z., Change Loy, C., & Lin, D. (2017). Polynet: A pursuit of structural diversity in very deep networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 718-726).
[140] Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., & Feng, J. (2017). Dual path networks. In Advances in neural information processing systems (pp. 4467-4475).
[141] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
[142] Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3431-3440).
[143] Sun, W., Huang, Z., Liang, M., Shao, T., & Bi, H. (2020). Cocoon Image Segmentation Method Based on Fully Convolutional Networks. In Proceedings of the Seventh Asia International Symposium on Mechatronics (pp. 832-843). Springer, Singapore.
[144] Vecchio, G., Palazzo, S., Giordano, D., Rundo, F., & Spampinato, C. (2020). MASK-RL: Multiagent Video Object Segmentation Framework Through Reinforcement Learning. IEEE Transactions on Neural Networks and Learning Systems.
[145] Ji, Z., Zhao, Y., Pang, Y., Li, X., & Han, J. (2020). Deep Attentive Video Summarization With Distribution Consistency Learning. IEEE transactions on neural networks and learning systems.
[146] Zhang, D., Han, J., Zhao, L., & Zhao, T. (2020). From Discriminant to Complete: Reinforcement Searching-Agent Learning for Weakly Supervised Object Detection. IEEE Transactions on Neural Networks and Learning Systems.
[147] Shih, K. H., Chiu, C. T., Lin, J. A., & Bu, Y. Y. (2019). Real-Time Object Detection With Reduced Region Proposal Network via Multi-Feature Concatenation. IEEE transactions on neural networks and learning systems.

[148] Zhou, Y., Yen, G. G., & Yi, Z. (2019). Evolutionary Compression of Deep Neural Networks for Biomedical Image Segmentation. IEEE transactions on neural networks and learning systems.
[149] Zhang, B., Xiong, D., Xie, J., & Su, J. (2020). Neural Machine Translation With GRU-Gated Attention Model. IEEE Transactions on Neural Networks and Learning Systems.
[150] Guo, M., Yang, Y., Xu, R., & Liu, Z. (2019). When NAS Meets Robustness: In Search of Robust Architectures against Adversarial Attacks. ArXiv, abs/1911.10695.
[151] Li, G., Qian, G., Delgadillo, I. C., MÃijller, M., Thabet, A., & Ghanem, B. (2019). SGAS: Sequential Greedy Architecture Search. arXiv preprint arXiv:1912.00195.
[152] Fang, J., Sun, Y., Zhang, Q., Li, Y., Liu, W., & Wang, X. (2019). Densely connected search space for more flexible neural architecture search. arXiv preprint arXiv:1906.09607.