Cognitive Computation (2023) 15:429–439
https://doi.org/10.1007/s12559-022-10046-y

DLW-NAS: Differentiable Light-Weight Neural Architecture Search

Shu Li[1] · Yuxu Mao[1] · Fuchang Zhang[1] · Dong Wang[1] · Guoqiang Zhong[1]

Received: 26 May 2022 / Accepted: 13 July 2022 / Published online: 8 August 2022

Abstract
In recent years, the method of automatically constructing convolutional neural networks based on neural architecture search has attracted wide attention, and greatly reduces the manual intervention and the cost of manual design of neural networks. However, most neural architecture search methods focus on the performance of the model, but ignore the complexity of the model, which makes it difficult to deploy this method on devices with limited resources. In this paper, a novel differentiable light-weight architecture search method named DLW-NAS is proposed, which aims to search convolutional neural networks (CNNs) with remarkable performance as well as a small amount of parameters and floating point operations (FLOPs). Concretely, in order to limit the parameters and FLOPs from the source of neural architecture search (NAS), we build a light-weight search space containing effective light-weight operations. Moreover, we design a differentiable NAS strategy with computation complexity constraints. In addition, we propose a neural architecture optimization method, which makes the topology of the searched architecture sparse. The experimental results show that DLW-NAS achieves 2.73% error rate on CIFAR-10, which is comparable to the state-of-the-art (SOTA) methods. However, it only needs 2.3M parameters and 334M FLOPs, which reduces that of the SOTA DARTS by 30% and 36% in parameters and FLOPs, respectively. The searched model on CIFAR-100 uses only 2.47M parameters and 376M FLOPs with an error rate of only 17.12%. On ImageNet, compared with the SOTA MobileNet, DLW-NAS achieves 3.3% better top-1 accuracy with much fewer parameters and FLOPs.

Keywords Neural architecture search · Light-weight search space · Differentiable architecture search

Introduction

Convolutional neural networks (CNNs) inspired by the cognitive mechanism of biological natural vision have been successfully applied to many fields in recent years, such as object recognition [1, 2], image segmentation [3, 4] and information retrieval [5, 6]. Generally, CNNs are manually designed for the specific applications. However, manual design methods are heavily dependent upon the knowledge of domain experts, and the design process requires lots of time and efforts.

To automatically design CNNs, Zoph and Le propose the first neural architecture search (NAS) algorithm [7]. Since then, the research on NAS has attracted more and more attention [8, 9]. As one of the most popular NAS algorithms, differentiable architecture search (DARTS) [8] is usually taken as a benchmark framework. DARTS searches CNNs based on a cell structure. Each cell can be seen as a directed acyclic graph (DAG) and has N nodes and E edges. Then, the searched cell is stacked in sequence to form a complete deep CNNs. In the search stage, DARTS assigns a learnable parameter α to each edge via the softmax function, where the value of α represents the contribution of the edge in the cell. Nevertheless, there exist two shortcomings in the DARTS algorithm. On the one hand, the operations in the cell have a major number of parameters and FLOPs. On the other hand, based on the architecture parameters, DARTS simply retains the two operations corresponding to the two largest α at each node, which results in relatively redundant connections in the cell. These two issues cause that DARTS can only search for CNNs with complex

✉ Dong Wang
wangdong@ouc.edu.cn

✉ Guoqiang Zhong
gqzhong@ouc.edu.cn

Shu Li
15689930257@163.com

Yuxu Mao
1369627028@qq.com

Fuchang Zhang
zfc_1527@163.com

[1] Ocean University of China, Qingdao 266100, China

architectures. However, the amount of model parameters and FLOPs is crucial for the deployment of CNNs on devices with limited memory and computing resources.

This paper puts forward a differentiable light-weight architecture search method, named DLW-NAS, to automatically build high-performance and light-weight CNNs. Specifically, the core ideas of this work include three aspects. First, as the search space can directly affect the accuracy and complexity of the searched model, we rebuild a new search space involving several effective light-weight operations. Second, we design a novel differentiable architecture search strategy with computation complexity constraints. Last but not the least, we propose a neural architecture optimization strategy, with which we can keep as few operations as possible in the cell while maintaining the model performance.

We have conducted extensive experiments on the CIFAR-10, CIFAR-100 and ImageNet datasets to evaluate DLW-NAS. DLW-NAS obtains 2.73% test error rate on CIFAR-10 with only 2.3M parameters and 334M FLOPs. On CIFAR-100, it uses only 2.47M parameters and 376M FLOPs with an error rate of 17.12%. When transferred from CIFAR-10 to ImageNet for mobile settings, using 3.8M parameters and 397M FLOPs, DLW-NAS produces top-1 and top-5 error rates of 26.1% and 8.3%, respectively, which are the state-of-the-art (SOTA) results but with fewer parameters and FLOPs than SOTA approaches.

Our major contributions can be summarized as below:

- We present a differentiable light-weight neural architecture search algorithm called DLW-NAS. Especially, we rebuild a light-weight search space to limit the amount of parameters and calculations of the searched model from the source of NAS.
- We design a differentiable NAS strategy with computation complexity constraints, which can be used to search for light-weight architectures.
- We propose a novel and effective neural architecture optimization strategy to greatly sparsify the cell structure.
- On standard image classification datasets, including CIFAR-10, CIFAR-100 and ImageNet, we obtain the SOTA results with the searched light-weight models.

This paper is extended based on our conference version [10] with significant improvements. Firstly, we conduct a more comprehensive review of related approaches on the basis of our conference version. Secondly, we design a new differentiable search strategy with computation complexity constraints to automatically search light-weight CNNs. Thirdly, we have added more comparisons on the CIFAR-10 and ImageNet datasets, and performed more experiments on the CIFAR-100 dataset. Experiments show that the proposed DLW-NAS in this paper outperforms its previous version. Additionally, we have conducted ablation study to verify the effect of each component of DLW-NAS.

The rest of this paper is organized as follows. In "Related Work", we briefly review some related work. In "The Proposed DLW-NAS", we describe the proposed method, including the design of the search space, the new search strategy, the neural architecture optimization strategy and computational complexity analysis. The experimental results are reported and analyzed in "Experiments and Results". The conclusion of this paper is presented in "Conclusion".

Related Work

There are mainly two ways for the construction of light-weight CNNs, namely handcrafted and automatic design. We briefly review these two kinds of approaches in the following.

Handcrafted Handcrafted methods usually change traditional convolutional operations or model construction rules to compress the amount of parameters and FLOPs. For instance, SqueezeNet [11] adjusts the number of feature channels via *expanding* and *squeezing* convolutional layers. To reduce the computational complexity of convolutional operations, Howard et al. propose the depthwise separable convolution to perform convolutional operations channel by channel [12]. ShuffleNet [13] achieves the reduction of computation complexity using group convolution and channel shuffle. Recently, based on a set of original feature maps, Han et al. have generated numerous "ghost" feature maps by using a serial of linear transformations, which is effective in digging out the required information from original features [1]. Although manually designed CNNs show extraordinarily considerable performance, the design methods are heavily dependent upon the knowledge of domain experts.

Automatic Design The auto-design of convolutinal networks usually refers to learning the optimal network structure based on NAS algorithms. As one of the most popular NAS algorithms, DARTS converts the discrete operations search into a differentiable optimization problem through the Softmax function, which effectively reduces the search time. Recently, some differentiable NAS methods used for automatically designing light-weight models on embedded devices are proposed [14, 15]. For example, Cai et al. propose a latency regularization loss to reduce inference latency [16] in the search process. Wu et al. present an algorithm for searching a continuous structure with sparsity constraints, which is known as the mixed-path NAS algorithm. These hardware-aware NAS methods are effective in reducing the inference delay of deep neural networks. However, they do not specifically explore how to limit model parameters and FLOPs. To search for CNNs with low computational complexity, quite a few methods are proposed by either constraining the number of transformation operations or redesigning the search space.

For example, both BayesNAS [17] and DSO-NAS [18] use the ℓ_1-norm to enforce the connections sparse in the cell topology. In CNAS [19], Weng et al. build a new search space by exploring some light-weight operations. In this work, we synthetically consider the differentiable light-weight architecture search problem. In particular, we design a novel search space containing only light-weight operations and propose a new strategy to optimize the neural architectures.

The Proposed DLW-NAS

This section describes our proposed differentiable light-weight neural structure search (DLW-NAS) method in detail. Specifically, the new light-weight search space is first presented in "Light-Weight Search Space", followed by the neural architecture search algorithm in "The Architecture Search Algorithm". Section "Neural Architecture Optimization" shows our proposed effective neural architecture optimization strategy. Finally, we quantitatively analyze the complexity of DLW-NAS in "Complexity Analysis".

Light-Weight Search Space

Search space determines the accuracy and complexity of the constructed CNNs to a great extent. It is the foundation of neural architecture search. To rebuild a new light-weight search space, we comprehensively consider candidate operations in the cell topology from three aspects, namely accuracy, parameters and FLOPs. Our target is to construct deep architectures with as high accuracy and as few parameters and FLOPs as possible. Hence, we mainly improve the search space of DARTS [8]. Except for five light-weight operations in DARTS search space, i.e., max pooling, depthwise separable convolution, identity, dilated convolution and zero, we explore three more awesome light-weight operations (i.e., GhostConv, SKConv and ShuConv). Therefore, there are eight light-weight operations in the search space of DLW-NAS. In the following, we briefly introduce the new three light-weight operations.

(i) GhostConv [1] is a plug-and-play light-weight operation. It can generate the same number of feature maps as traditional convolution through a series of simple linear transformation operations (e.g., depthwise separable convolution and 1×1 convolution). However, different from conventional convolution, the number of parameters and FLOPs of GhostConv is only $1/r$ of that of conventional convolution operation, where r represents the number of the used convolutional kernels.

(ii) SKConv [20] makes the input feature maps segmented into several branches along the channel dimension and uses convolutional kernels to procure features of different sizes of receptive fields.

(iii) ShuConv [13] adopts the point-wise group convolutions as well as channel shuffle, so that they can significantly reduce the computational overhead.

Besides, we constrain the light-weight search space of DLW-NAS from the following two aspects:

1. **Limiting the size of convolutional kernels to 3×3**. Previous methods usually include convolution kernels of size 5×5 or even 7×7. Whereas, these convolutional kernels retain larger receptive field with a quantity of parameters and FLOPs. Therefore, we abandon convolutional operations with receptive field larger than 3×3. In addition, it is desirable to replace stacked convolutional layers and have large receptive field with dilated convolution. For example, with the dilation rate 2, a 3×3 dilated convolution can be regarded as a 5×5 convolutional operation.
2. **Droping redundant operations.** In [21], Li et al. prove the existence of the multi-collinearity problem among the candidate operations of previous differentiable NAS approaches [8]. That is, some operations (such as average pooling and max pooling) have high linear correlation, which may split the contributions of the candidate operations during the model searching. To avoid this problem, we preserve only one operation with multi-collinearity in the search space of DLW-NAS. This guarantees that the candidate operations are not redundant.

The Architecture Search Algorithm

This subsection introduces the proposed differentiable neural architecture search strategy. Following previous work [8], we search a cell and stack it to build the deep architecture. The structure of the initial cell is shown in Fig. 1a. It can be abstracted as a directed acyclic graph (DAG). In the cell, the nodes represent layers of the network, while each directional edge $E_{(i,j)}$ represents the information flow which is from node i to node j. During the search process, information flows that input into a node are summarized to:

$$x^{(j)} = \sum_{i<j} \overline{o}^{(i,j)}(x^{(i)}), \tag{1}$$

where $x^{(i)}$ represents the output of the i-th node, and $\overline{o}^{(i,j)}$ is a set of candidate operations performed between node i and node j (e.g., 3×3 dilated convolution and max pooling). For changing the discrete search to a continuous and differentiable optimization problem, like DARTS [8], we utilize the softmax function to compute the contribution of each operation.

$$\overline{o}^{(i,j)} = \sum_{o \in O} \frac{exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} exp(\alpha_{o'}^{(i,j)})} o(x), \quad i < j, \tag{2}$$

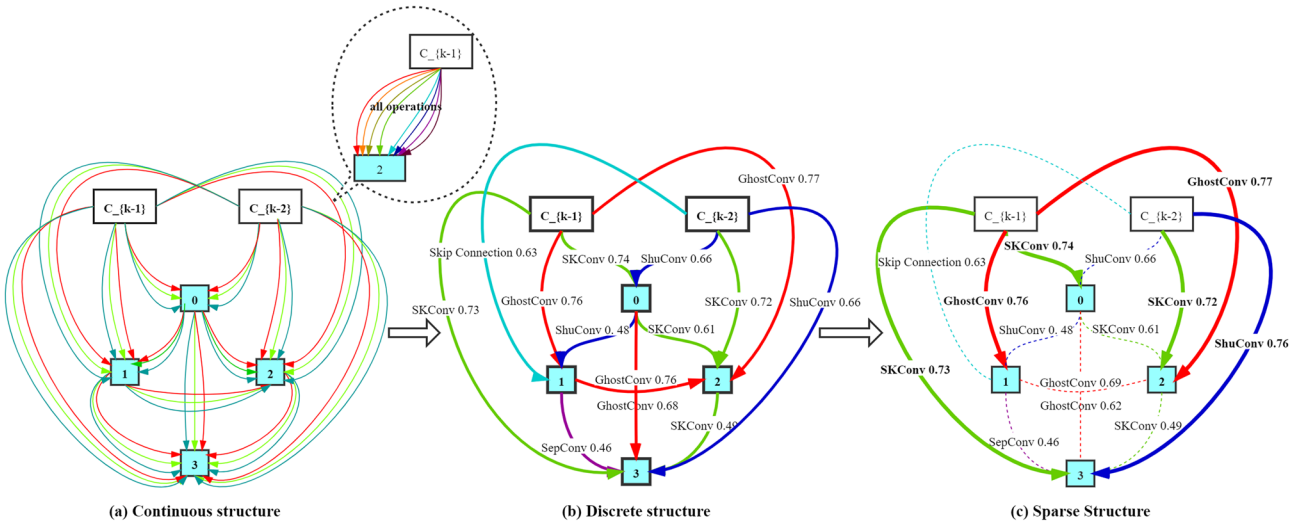

Fig. 1 The optimization process of neural architecture. **a** A cell consisting of nodes and directional edges. Two white boxes denote the input nodes, while the blue boxes denote the inner nodes. The directional edges between two nodes denote the candidate operations. **b** In accordance with the learned architecture weights, only the operation contributes the most is retained. **c** The ultimate sparse architecture obtained based on our proposed architecture optimization strategy

where O denotes the set of candidate light-weight operations in the search space, and o represents an operation between node i and node j with its architecture weight $\alpha_o^{(i,j)}$. Specifically, we apply a vector $\alpha^{(i,j)}$ with dimension $|O|$ to parameterize the candidate operations between node i and node j. In addition, the value of $\alpha_o^{(i,j)}$ measures how much a candidate operation has contributed in the feature map transformation.

To learn light-weight architectures, we add computational complexity constraints on the differentiable optimization algorithm. We use $P(o^{(i,j)})$ and $F(o^{(i,j)})$ to compute the number of parameters and FLOPs of operation $o^{(i,j)}$, respectively. To maintain the differentiability of the objective function, the weighted parameter number and FLOPs between a pair of nodes can be computed as:

$$\mathbb{E}_{[params^{(i,j)}]} = \sum_{o \in O} \alpha_o^{(i,j)} \times P(o^{(i,j)}), \tag{3}$$

$$\mathbb{E}_{[flops^{(i,j)}]} = \sum_{o \in O} \alpha_o^{(i,j)} \times F(o^{(i,j)}). \tag{4}$$

To the end, the architecture parameters can be updated using the gradients.

$$\begin{cases} \frac{\partial \mathbb{E}_{[params^{(i,j)}]}}{\partial \alpha_o^{(i,j)}} = P(o^{(i,j)}), \\ \frac{\partial \mathbb{E}_{[flops^{(i,j)}]}}{\partial \alpha_o^{(i,j)}} = F(o^{(i,j)}). \end{cases} \tag{5}$$

The parameter number and FLOPs of a cell are obtained by summarizing up the parameter numbers and FLOPs of all the internal operations:

$$\mathbb{E}_{[params_n]} = \sum_{i \in n} \sum_{j > i} \mathbb{E}_{[params^{(i,j)}]}, \tag{6}$$

$$\mathbb{E}_{[flops_n]} = \sum_{i \in n} \sum_{j > i} \mathbb{E}_{[flops^{(i,j)}]}. \tag{7}$$

After the search stage, the constructed deep architecture contains several cells, and its parameter number and FLOPs are the sum of all the cells:

$$\mathbb{E}_{[params]} = \sum_{n} \mathbb{E}_{[params_n]}, \tag{8}$$

$$\mathbb{E}_{[flops]} = \sum_{n} \mathbb{E}_{[flops_n]}. \tag{9}$$

Based on the above analysis, we design the objective function as follows:

$$\mathcal{L} = CE + \lambda_1 \mathbb{E}_{[params]} + \lambda_2 \mathbb{E}_{[flops]}, \tag{10}$$

where CE is the cross entropy loss to evaluate the deep architecture and the latter two terms are used to restrict the amount of parameters and FLOPs, respectively. Specifically, λ_1 and λ_2 are trade-off factors among the accuracy, parameter number and FLOPs.

Neural Architecture Optimization

As shown in Fig. 1, the optimization process of DLW-NAS mainly includes two stages. In the first stage, the architecture weights between each pair of nodes in the cell (as shown in

Fig. 1a) are learned and sorted, and only the operation corresponding to the maximum architecture weight is reserved. Then, the original cell becomes a discrete structure (as shown in Fig. 1b). Here we emphasize that, although most of the operations in the original cell have been discarded, as shown in Fig. 1b, there is a connection between any nodes in the discrete structure.

In the second stage, while keeping the model accuracy, we further sparsify the discrete structure and reduce its parameter number and FLOPs. Specifically, the proposed strategy which is based on the optimized spanning tree ensures us to obtain the sparse cell topology. Concretely, we first abstract the discrete structure into a weighted and undirected graph, and the weight value on each edge is set to the inverse of the corresponding architecture weight. Then, in the undirected graph, we find a minimum spanning tree T. Since the construction of the minimum spanning tree T starts from the edges with the least weight, the edges contained in T correspond to the operations with the relatively large architecture weights. Nevertheless, some inner nodes may have only output edge but no input edge. In this case, the information flows will be influenced. To solve this problem, we traverse every node in the cell. As long as a node encounters such a problem, the most weighted operations having input to this node are added to T. By this step, we complete the transformation from the discrete structure to the sparse structure (as shown in Fig. 1c).

As shown in Fig. 2, through the above two stages, the proposed architecture optimization strategy yields relatively sparse cells compared to that of DARTS with 8 connections and that of SparseNAS with 12 connections. Whereas, there still exists one issue that whether or not the reserved connections are suitable for the learning tasks. In Fig. 3, we show some experimental results obtained on the CIFAR-10 and CIFAR-100 datasets. From Fig. 3, we can see that 6 is an elbow point of the curves about validation error. This indicates that the object recognition accuracy of the constructed evaluation model with only five connections in the cell is much lower than that with six connections. Moreover, although the performance of the evaluation models with 7 or 8 connections in the searched cell is comparable to that with 6 connections, they need much more parameters. This

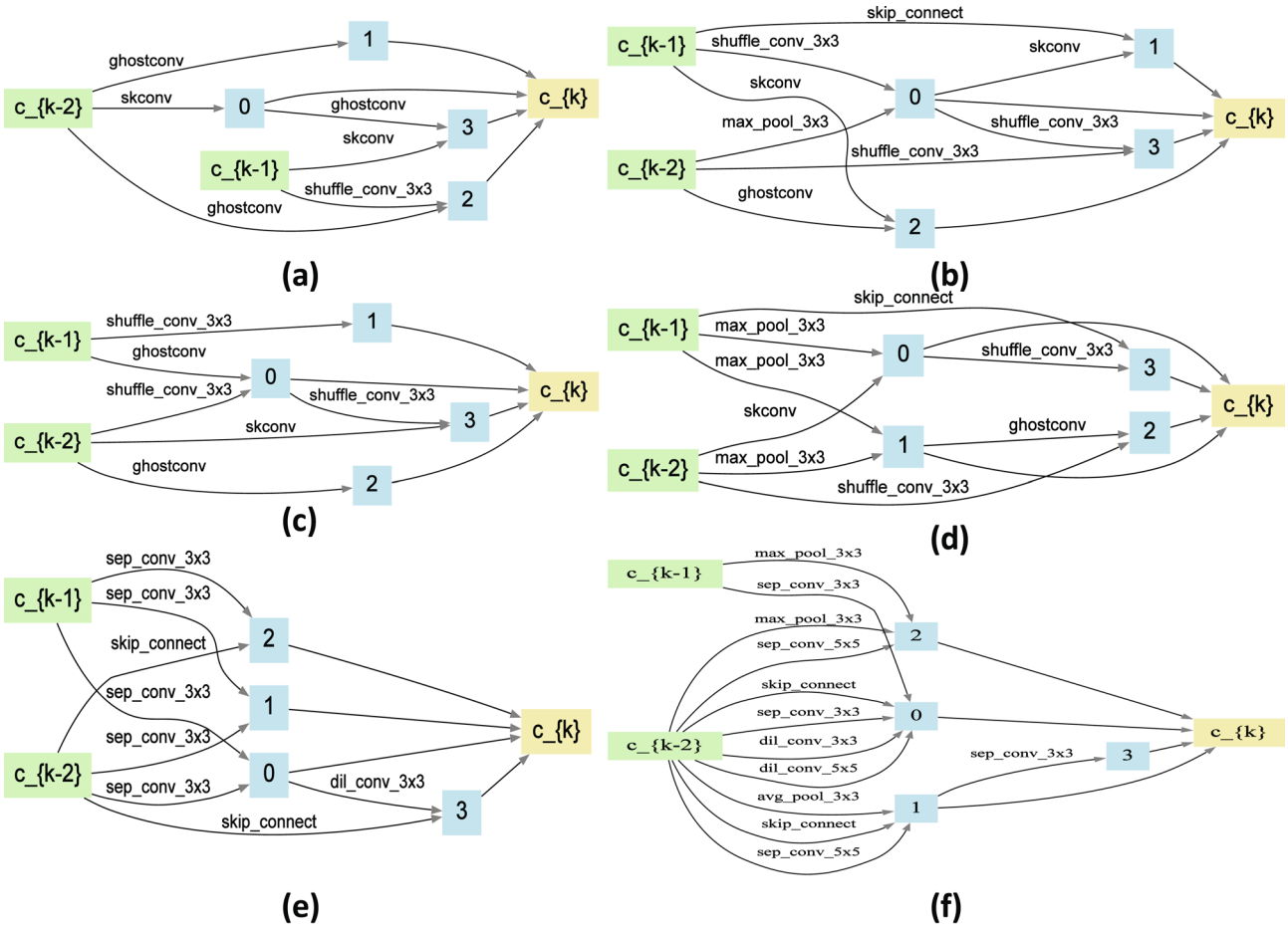

Fig. 2 Topological structure of normal cells and reduction cells. **a** Normal cell searched by DLW-NAS on CIFAR-10. **b** Reduction cell searched by DLW-NAS on CIFAR-10. **c** Normal cell searched by DLW-NAS on CIFAR-100. **d** Reduction cell searched by DLW-NAS on CIFAR-100. **e** Normal cell searched by DARTS [8] on CIFAR-10. **f** Normal cell searched by SparseNAS [22] on CIFAR-10

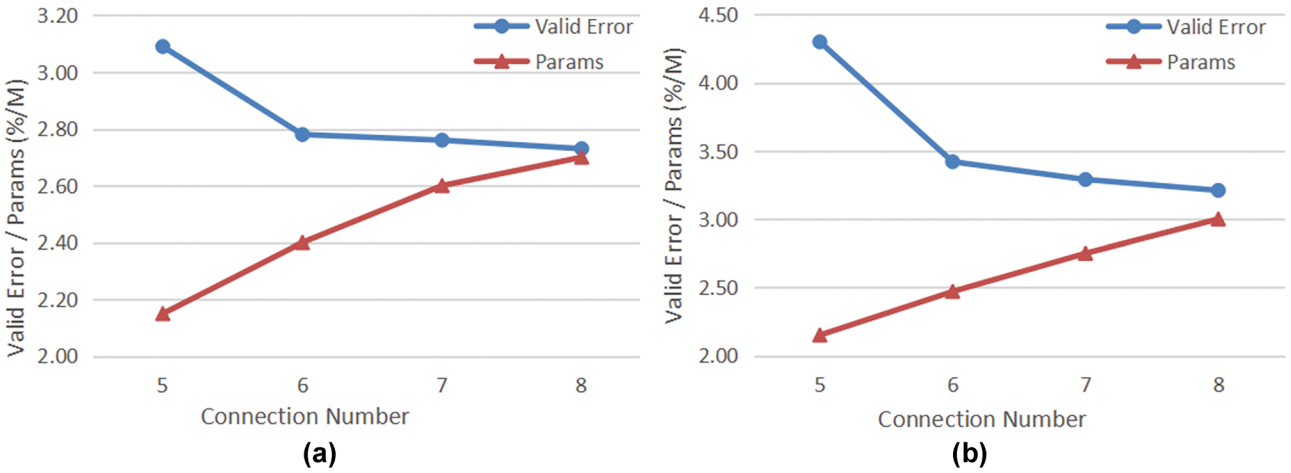

Fig. 3 **a** The effect of the connections number contained in the searched cell on model parameters and evaluation accuracy on CIFAR-10. **b** The effect of the connections number contained in the searched cell on model parameters and evaluation accuracy on CIFAR-100

motivates us to restrict the number of operations in the cell to a constant $M = 6$. As the retained operations number N is less than M, the $M - N$ operations, with the largest architecture weights, will be selected. Subsequently, these M operations are used to construct the final cell.

Algorithm 1 describes the overall process of the neural architecture optimization strategy proposed in this section. The structural weight of the initial neural structure block is the trained parameter, and the input α is an N × 8 parameter matrix, where 8 represents the eight operations

Algorithm 1 Neural Architecture Optimization Algorithm.

Input:
The number of nodes in nerve block, N;
The threshold of internal connection number of nerve block, M;
The learned structural parameters, $\alpha[[\alpha_0^{<0,1>}, ..., \alpha_7^{<0,1>}], ..., [\alpha_0^{<N-2,N-1>}, ..., \alpha_7^{<N-2,N-1>}]]$;
The empty list, $w \leftarrow [\,]$, $w' \leftarrow [\,]$
Output:
The sparse neural structure with M connections, T;

for $i = 1$ *to* $N - 2$ **do**
  for $j = i + 1$ *to* $N - 1$ **do**
    $\alpha^{<i,j>} \leftarrow sort[\alpha_0^{<i,j>}, \alpha_1^{<i,j>}, ..., \alpha_7^{<i,j>}]$;
    $w^{<i,j>} \leftarrow argmax\alpha^{<i,j>}$;
    $W.append(w^{<i,j>})$;
  end for
end for
$W' \leftarrow [\frac{1}{w^{<0,1>}}, \frac{1}{w^{<0,2>}}, ..., \frac{1}{w^{<N-2,N-1>}}]$;
$T \leftarrow Kruskal(W')$;
Connection_num ← sum(T(←));
for t *to* T **do**
  if ((←)t==none && (t(←))!= none) **then**
    $T.append(max(w^{<0,t>}, ..., w^{<t-1,t>}))$;
    Connection_num += 1;
  else
    continue;
  end if
end for
if Connection_num<M **then**
  for $i = 1$ *to* $M - Connection\_num$ **do**
    T.append(max(W-T));
  end for
end if

contained in the search space. Lines 1 to 7 describe the process of transforming the initial neural structure into discrete neural structure blocks. Lines 8 to 18 describe how to realize the transformation from discrete structures to sparse structures. Lines 9 to 23 introduce the edge number regularization rules used to balance the accuracy and complexity of neural networks. Finally, a sparse and light-weight neural structure block with excellent performance is output.

Complexity Analysis

In this subsection, we analyze the computational complexity of DLW-NAS from two aspects, i.e., the search space and the whole deep architecture.

Existing differentiable NAS methods are mainly implemented based on the DARTS search space. Hence, we compare the proposed light-weight search space and that of DARTS. For convenience, taking a pair of nodes as an example, we analyze the computational complexity of the search space. Here, we assume that, the size of the input feature maps is 32×32, both input and output have 32 channels $C_{in} = C_{out} = 32$, while the kernel size and convolutional stride are 3×3 and 1, respectively. Regardless of the bias, Table 1 compares the parameters and FLOPs of the search space of DLW-NAS and DARTS. For clarity, we calculate the mean values of parameters and FLOPs for all the operations between a pair of nodes as follows.

$$\begin{aligned} Params &= \frac{1}{|O|}\sum_{o\in O} p_o, \quad p_o > 0, \\ FLOPs &= \frac{1}{|O|}\sum_{o\in O} m_o, \quad m_o > 0, \end{aligned} \tag{11}$$

where O denotes the set of the candidate operations, $|O|$ is the size of O, while p_o and m_o are the parameter number and FLOPs of the operation o, respectively. Compared with DARTS, the average parameters and FLOPs of DLW-NAS are smaller, which is only 0.126M and 0.84M, respectively. It demonstrates that the computational complexity of the search space of DLW-NAS is lower than that of DARTS.

To analyze the complexity of architecture, we take the evaluation model that stacks 20 cells as an example. The architecture complexity is evaluated by the number of transformations of feature maps inside the cells. DLW-NAS and DARTS contain 6 and 8 connections in a cell, respectively. Thus, an image only needs $20 \times 6 = 120$ transformations in DLW-NAS from input to output, while DARTS has to pass $20 \times 8 = 160$ transformations. It is easy to calculate that our proposed architecture optimization method reduces the number of operations of the evaluation model by 25%. This shows that the architecture learned by DLW-NAS is more light-weight than that learned by DARTS.

Experiments and Results

In this section, we report the experimental settings and results in detail.

Datasets and Implementation Issues

To evaluate the proposed DLW-NAS method, we have conducted experimental comparison on three standard image classification datasets, including CIFAR-10, CIFAR-100 and ImageNet (mobile setting). We briefly introduce them as follows.

CIFAR-10 and CIFAR-100 [23] These two datasets consist of 50K training images and 10K testing images separately, with image size of 32×32×3. The images in these two datasets belong to 10 and 100 classes, respectively. During the architecture search, half of the training data are applied to train the architecture weights, and the remaining half are used to adjust the parameters of the searched architecture.

ImageNet [24] It is composed of 1.3M images for training and 50K images for test, and the images belong to 1,000

Table 1 Comparison with DARTS on candidate operations of the search space. The best results are highlighted with boldface

DLW-NAS	Params (K)	FLOPs (M)	DARTS	Params (K)	FLOPs (M)
Identity	-	-	Identity	-	-
3×3 SepConv	1.41	1.34	3×3 SepConv	1.41	1.34
3×3 DilConv	1.41	1.34	5×5 SepConv	1.93	1.64
3×3 ShuConv	0.30	0.20	3×3 DilConv	1.41	1.34
SKConv	2.44	0.65	5×5 DilConv	1.93	1.64
GhostConv	0.74	0.67	3×3 Max Pooling	-	-
3×3 Max Pooling	-	-	3×3 Avg Pooling	-	-
Zero	-	-	Zero	-	-
Avg	**1.26**	**0.84**	Avg	1.67	1.49

classes. The size of the images is 224×224×3. The mobile setting is used for test.

Following the optimization algorithm in [25], we learn the architecture weights and network parameters. After the architecture search stage, we utilize the proposed architecture optimization strategy to transform the searched cell to sparse structure, which contains $M = 6$ connections. Moreover, λ_1 and λ_2 are set to 0.01 and 0.005, respectively. When evaluating the architecture, we stack 20 cells and train the deep architecture on CIFAR-10 and CIFAR-100 from scratch. In addition, on ImageNet, we stack 14 cells to test the deep architecture. Particularly, our experiments are conducted using two NVIDIA GTX 1080Ti GPUs.

The Searched Architectures

We conduct the architecture search on CIFAR10 and CIFAR100. Figure 2 presents the architectures searched by DLW-NAS. Compared with SOTA DARTS [8] and SparseNAS [22], we can see that the normal cells searched by DLW-NAS are more sparse, and the number of internal connections is only 3/4 of DARTS and 1/2 of SparseNAS. Such results are attributed to the architecture optimization strategy of DLW-NAS. Furthermore, the up-to-date lightweight operations in the search space ensure to deliver a quite efficient deep architecture.

Architecture Evaluation

In the following, we report the evaluation results on the searched architectures.

Experiments on CIFAR-10 and CIFAR-100

Following previous work, we construct the evaluation model with 20 searched cells, including 18 normal cells and 2 reduction cells. With batch size 96, we train the evaluation model for 600 epochs from scratch. The standard SGD optimizer is used, and we set the initial learning rate to 0.025, the momentum to 0.9 and the weight decay to 3×10^{-4} on CIFAR-10 and 5×10^{-5} on CIFAR-100. Auxiliary towers [26] of weight 0.4 and cutout regularization [27] of length 16 are also applied.

In Table 2, we list the comparison results between our method and other NAS methods, where "–" indicates that the relevant results are not given in the research work listed. The second column shows the recognition error rate of CNN

Table 2 Comparison with state-of-the-art methods on CIFAR-10. The best results are highlighted with boldface

Architecture	Test Error. (%)	Params (M)	FLOPs (M)	Search Cost (GPU-days)	Search Method
DenseNet-BC [28]	3.46	25.6	-	-	manual
MobileNet [12]	15.60	3.3	-	-	manual
Ghost-VGG-16 [1]	6.30	7.7	-	-	manual
RandomNAS [29]	2.85	4.3	-	2.7	RS
DARTS-random + cutout [8]	3.29	3.2	-	4	RS
AmodebaNet-A + cutout [30]	3.12	3.2	-	3150	EA
Hierarchical-EAS [31]	3.75	15.7	-	300	EA
CARS-E + cutout [32]	2.86	3.0	-	0.4	EA
MetaQNN + cutout [33]	6.92	-	-	100	RL
BlockQNN [34]	3.54	39.8	-	96	RL
ENAS + cutout [35]	2.89	4.6	626	0.5	RL
Path-level EAS [36]	2.99	5.7	-	200	RL
PNAS [37]	3.41	3.2	-	225	SMBO
EPNAS [38]	3.71	6.6	-	1.8	SMBO
GHN [39]	2.84	5.7	-	0.84	SMBO
DARTS (first order) + cutout [8]	3.00	3.3	-	1.5	GD
DARTS (second order) + cutout [8]	2.76	3.3	528	4	GD
SNAS + mild [40]	2.98	2.9	422	1.5	GD
CNAS + cutout [19]	4.23	2.95	-	2.5	GD
GDAS + cutout [41]	2.93	3.4	519	0.21	GD
DSO-NAS-full [18]	2.95	3	-	1	GD
SparseNAS + HAPG [22]	**2.73**	3.8	538	1	GD
BayesNAS + cutout [17]	2.90	3.1	-	**0.2**	GD
DLW-NAS + cutout (ours)	**2.73**	**2.3**	**334**	0.45	GD

models constructed by different methods on CIFAR-10. The smaller the value is, the higher the recognition accuracy of the model for objects is. The third and fourth columns are important indicators to measure the complexity of the model, showing the number of model parameters and the amount of calculation of different methods. Using "M" as a statistical unit, the smaller the value is, the lower the complexity of the model is, which means the model is lighter. In addition, the last two columns in the table show the CPU time and model construction methods required for automatic construction of convolutional neural networks, in which RL (Reinforcement Learning), EA (Evolutionary Algorithm) and SMBO (Sequential model-based optimization) are all methods for automatic model construction. Thus, we can see that DLW-NAS achieves a 2.73% error rate with only 2.3M parameters and 334M FLOPs. Compared with the SOTA NAS approaches, such as BayesNAS [17] and DSO-NAS [18], the classification accuracy obtained by DLW-NAS is higher and the number of model parameters is greatly reduced. The test error of SparseNAS [22] is comparable with DLW-NAS. However, there are more connections in the searched cell, such that SparseNAS has more parameters and FLOPs. The parameters and FLOPs of DLW-NAS are about 37% less than those of SparseNAS.

Table 3 shows the performance of different methods on CIFAR-100. The evaluation indices in the experiments are consistent with those in Table 2. As shown in Table 3, the architecture searched by DLW-NAS on CIFAR-100 uses only 2.47M parameters and 376M FLOPs and delivers an error rate of 17.12%. Comparing with the SOTA DARTS, DLW-NAS surpasses DARTS on classification performance but using fewer parameters and FLOPs. This further demonstrates the advantages of DLW-NAS over SOTA approaches. In a word, compared with the manual design of neural network, evolutionary algorithm and reinforcement learning method, the network structure searched by our method has better performance, faster search speed, fewer parameters and less computation. In comparison with other differentiable neural architecture search methods, our method also has distinct advantages. The results obtained by DLW-NAS are comparable with faster search speed in the case of smaller parameters and less computation with that of the SOTA methods.

Experiments on ImageNet

In this experiment, the architectures searched on CIFAR-10 are transferred to ImageNet for mobile setting test. According to the common comparison method on the ImageNet dataset, Top-1 and Top-5 error rates are used to evaluate the compared methods, and the complexity evaluation indices are the same as CIFAR-10 and CIFAR-100 datasets. Following the conventions [8], we construct the evaluation model with 14 cells, including 12 normal cells as well as 2 reduction ones. Moreover, we train the evaluation model for 250 epochs with batch size 256. Bsides we set the initial value of the learning rate of the SGD optimizer to 0.1.

It can be seen from Table 4 that the results obtained by DLW-NAS are obviously superior to the manual design of neural network methods and other differentiable methods on ImageNet. With only 3.8M parameters and 397M FLOPs, DLW-NAS achieves a 26.1% error rate and outperforms DARTS on either model complexity or classification accuracy. In addition, DLW-NAS performs slightly better than DSO-NAS [18] which is directly searched on ImageNet, with even less parameters and FLOPs. This demonstrates the effectiveness of DLW-NAS on architecture search. In terms of accuracy, SparseNAS performs slightly better than DLW-NAS, but its number of parameters is much more than that of DLW-NAS.

Ablation Study

DLW-NAS mainly includes three innovations: the lightweight search space (LWSS), search strategy with complexity constraints (SSCC) and architecture optimization strategy (AOS). In the experiment, we also conduct ablation study to evaluate the contribution of these components to the performance of DLW-NAS on image classification.

Table 5 shows the experimental results on CIFAR-10. It can be seen that each component has certain impact on

Table 3 Comparison with state-of-the-art methods on CIFAR-100. The best results are highlighted with boldface

Architecture	Test Error. (%)	Params (M)	FLOPs (M)	Search Cost (GPU-days)	Search Method
DenseNet-BC [28]	17.18	25.6	-	-	manual
ResNeXt-29 [42]	18.56	25.2	-	-	manual
SKNet [20]	17.33	27.7	-	-	manual
DARTS (first order) + cutout [8]	17.74	3.4	536	1.5	GD
DARTS (second order) + cutout [8]	17.58	3.3	518	4	GD
CNAS + cutout [19]	22.24	3.67	-	2.5	GD
DLW-NAS + cutout (ours)	**17.12**	**2.47**	**376**	**0.45**	GD

Table 4 Comparison with state-of-the-art methods on ImageNet (mobile setting). The best results are highlighted with boldface

Architecture	Test Err.(%) top-1	top-5	Params (M)	FLOPs (M)	Search Cost (GPU-days)	Search Method
MobileNet [12]	29.4	10.5	4.2	569	-	manual
ShuffleNet [13]	26.4	10.2	5.0	524	-	manual
OFA-random [14]	26.2	-	7.7	-	-	RS
NASNet-B [43]	27.2	8.7	5.3	488	1800	RL
NASNet-C [43]	27.5	9	4.9	558	1800	RL
GHN [39]	27	8.7	6.1	-	0.84	SMBO
DARTS [8]	26.7	8.7	4.7	574	4	GD
DSO-NAS [18]	26.2	8.6	4.7	571	1	GD
SNAS [40]	27.3	9.2	4.3	522	1.5	GD
GDAS [41]	27.5	9.1	4.4	497	0.21	GD
BayesNAS [17]	26.5	8.9	3.9	-	**0.2**	GD
SparseNAS + HAPG [22]	**25.5**	**8.1**	5.3	-	1	GD
DLW-NAS (ours)	26.1	8.3	**3.8**	**397**	0.45	GD

Table 5 Contribution of LWSS, SSCC and AOS to model performance and complexity. The best results are highlighted with boldface

LWSS	SSCC	AOS	Test Err. (%)	Params (M)	FLOPs (M)
√			2.76	2.9	385
	√		2.98	3.1	478
		√	2.83	2.7	411
√	√		2.81	2.6	359
√		√	2.76	2.4	336
	√	√	2.84	2.6	398
√	√	√	**2.73**	**2.3**	**334**

the learning accuracy and complexity of the evaluation model. For example, the results in the fourth row are based on the DARTS search space, while that in the sixth row are based on the proposed LWSS. As we can see that using the proposed LWSS can search for models with lower computational complexity and higher recognition accuracy than using the DARTS search space. Comparing the results shown in the second and sixth rows, it can be seen that the proposed AOS can greatly reduce the parameter number and computational complexity. Overall, applying all the three components, DLW-NAS can obtain SOTA learning accuracy with low computational overhead.

Conclusion

In this work, we propose DLW-NAS, which is a differentiable light-weight neural architecture search method. To realize the light-weight architecture search from the source, we establish a novel light-weight search space. Furthermore, we propose a new differentiable architecture search strategy with complexity constraints. In addition, we introduce an architecture optimization strategy to sparsify the connections in the searched architecture. This strategy reduces the parameter number and computational complexity, but basically preserves the model performance. To evaluate the proposed DLW-NAS method, we test it on the CIFAR-10, CIFAR-100 and ImageNet datasets. The results demonstrate its advantages over the SOTA approaches.

Funding This work was partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0100400, the Natural Science Foundation of Shandong Province under Grants No. ZR2020MF131 and No. ZR2021ZD19, and the Science and Technology Program of Qingdao under Grant No. 21-1-4-ny-19-nsh.

Data Availability Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

Declarations

Ethics Approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed Consent Informed consent was obtained from all individual participants included in the study.

Conflicts of Interest The authors declare no competing interests.

References

1. Han K, Wang Y, Tian Q, Guo J, Xu C, Xu C. Ghostnet: more features from cheap operations. In: CVPR. 2020. p. 1577–586.
2. Sun Y, Xue B, Zhang M, Yen GG, Lv J. Automatically designing CNN architectures using the genetic algorithm for image classification. IEEE Trans Cybern. 2020;50(9):3840–54.

3. Orsic M, Segvic S. Efficient semantic segmentation with pyramidal fusion. Pattern Recognit. 2021;110:107611.
4. Yang L, Wang H, Zeng Q, Liu Y, Bian G. A hybrid deep segmentation network for fundus vessels via deep-learning framework. Neurocomputing. 2021;448:168–78.
5. Ebadi N, Jozani M, Choo KKR, Rad P. A memory network information retrieval model for identification of news misinformation. IEEE Transactions on Big Data. 2021.
6. Mao Y, Zhong G, Wang H, Huang K. MCRN: a new content-based music classification and recommendation network. In: ICONIP, vol. 1332. 2020. p. 771–79.
7. Zoph B, Le QV. Neural architecture search with reinforcement learning. In: ICLR. 2017.
8. Liu H, Simonyan K, Yang Y. DARTS: differentiable architecture search. In: ICLR. 2019.
9. Zhao J, Zhang R, Zhou Z, Chen S, Liu Q. A neural architecture search method based on gradient descent for remaining useful life estimation. Neurocomputing. 2021;438(1).
10. Mao Y, Zhong G, Wang Y, Deng Z. Differentiable light-weight architecture search. In: IEEE International Conference on Multimedia and Expo (ICME). 2021. p. 1–6. https://doi.org/10.1109/ICME51207.2021.9428132.
11. Iandola FN, Moskewicz MW, Ashraf K, Han S, Dally WJ, Keutzer K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. CoRR abs/1602.07360. 2016.
12. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H. Mobilenets: efficient convolutional neural networks for mobile vision applications. CoRR abs/1704.04861. 2017.
13. Zhang X, Zhou X, Lin M, Sun J. Shufflenet: an extremely efficient convolutional neural network for mobile devices. In: CVPR. 2018. p. 6848–6856.
14. Cai H, Gan C, Wang T, Zhang Z, Han S. Once-for-All: train one network and specialize it for efficient deployment. In: ICLR. 2020.
15. Wu B, Dai X, Zhang P, Wang Y, Sun F, Wu Y, Tian Y, Vajda P, Jia Y, Keutzer K. FBNet: hardware-aware efficient ConvNet design via differentiable neural architecture search. In: CVPR. 2019. p. 10734–10742.
16. Cai H, Zhu L, Han S. ProxylessNAS: direct neural architecture search on target task and hardware. In: ICLR. 2019.
17. Zhou H, Yang M, Wang J, Pan W. BayesNAS: a Bayesian approach for neural architecture search. In: ICML, vol. 97. 2019. p. 7603–613.
18. Zhang X, Huang Z, Wang N. You only search once: single shot neural architecture search via direct sparse optimization. CoRR abs/1811.01567. 2019.
19. Weng Y, Zhou T, Liu L, Xia C. Automatic convolutional neural architecture search for image classification under different scenes. IEEE Access. 2019;7:38495–506.
20. Li X, Wang W, Hu X, Yang J. Selective kernel networks. In: CVPR. 2019. p. 510–19.
21. Li G, Zhang X, Wang Z, Li Z, Zhang T. STACNAS: towards stable and consistent optimization for differentiable neural architecture search. CoRR abs/1909.11926. 2019.
22. Wu Y, Liu A, Huang Z, Zhang S, Gool LV. Neural architecture search as sparse supernet. CoRR abs/2007.16112. 2020.
23. Krizhevsky A. Learning multiple layers of features from tiny images. University of Toronto. 2012.
24. Deng J, Dong W, Socher R, Li L, Li K, Li F. Imagenet: a large-scale hierarchical image database. In: CVPR. 2009. p. 248–55.
25. Chen X, Xie L, Wu J, Tian Q. Progressive differentiable architecture search: bridging the depth gap between search and evaluation. In: ICCV. 2019. p. 1294–303
26. Szegedy C, Liu W, Jia Y, Sermanet P, Reed SE, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. Going deeper with convolutions. In: CVPR. 2015. p. 1–9.
27. Devries T, Taylor GW. Improved regularization of convolutional neural networks with cutout. CoRR abs/1708.04552. 2017
28. Huang G, Liu Z, vander Maaten L, Weinberger KQ. Densely connected convolutional networks. In: CVPR. 2017. p. 2261–269.
29. Li L, Talwalkar A. Random search and reproducibility for neural architecture search. In: Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI, Tel Aviv, Israel, July 22–25. AUAI Press; 2019. p. 367–77.
30. Real E, Aggarwal A, Huang Y, Le QV. Regularized evolution for image classifier architecture search. In: AAAI. 2019. p. 4780–4789.
31. Liu H, Simonyan K, Vinyals O, Fernando C, Kavukcuoglu K. Hierarchical representations for efficient architecture search. In: 6th International Conference on Learning Representations, ICLR, Vancouver, BC, Canada, April 30 - May 3, Conference Track Proceedings. 2018. OpenReview.net.
32. Yang Z, Wang Y, Chen X, Shi B, Xu C, Xu C, Tian Q, Xu C. CARS: continuous evolution for efficient neural architecture search. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, Seattle, WA, USA, June 13-19. IEEE; 2020. p. 1826–835.
33. Baker B, Gupta O, Naik N, Raskar R. Designing neural network architectures using reinforcement learning. In: 5th International Conference on Learning Representations. ICLR; 2017. OpenReview.net.
34. Zhong Z, Yang Z, Deng B, Yan J, Wu W, Shao J, Liu C. Blockqnn: Efficient block-wise neural network architecture generation. CoRR. 2018.
35. Pham H, Guan MY, Zoph B, Le QV, Dean J. Efficient neural architecture search via parameter sharing. In: ICML, vol. 80. 2018. p. 4092–4101.
36. Cai H, Yang J, Zhang W, Han S, Yu Y. Path-level network transformation for efficient architecture search. In: Proceedings of the 35th International Conference on Machine Learning, ICML, Stockholmsmässan, Stockholm, Sweden, July 10–15. PMLR; 2018. p. 677–86.
37. Liu C, Zoph B, Neumann M, Shlens J, Hua W, Li L, Fei-Fei L, Yuille AL, Huang J, Murphy K. Progressive neural architecture search. In: Computer Vision - ECCV - 15th European Conference, Munich, Germany, September 8–14, Proceedings, Part I. Springer; 2018.
38. Perez-Rua J, Baccouche M, Pateux S. Efficient progressive neural architecture search. In: British Machine Vision Conference, BMVC, Newcastle, UK, September 3-6. BMVA Press; 2018. p. 150.
39. Zhang C, Ren M, Urtasun R. Graph hypernetworks for neural architecture search abs/1810.05749. 2018.
40. Xie S, Zheng H, Liu C, Lin L. SNAS: stochastic neural architecture search. In: ICLR. 2019.
41. Dong X, Yang Y. Searching for a robust neural architecture in four GPU hours. In: CVPR. 2019. p. 1761–1770.
42. Xie S, Girshick RB, Dollár P, Tu Z, He K. Aggregated residual transformations for deep neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Honolulu, HI, USA, July 21–26. IEEE Computer Society; 2017. p. 5987–995.
43. Zoph B, Vasudevan V, Shlens J, Le QV. Learning transferable architectures for scalable image recognition. In: CVPR. 2018. p. 8697–8710.