IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 8, 2019, accepted March 12, 2019, date of current version April 5, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2906369

Automatic Convolutional Neural Architecture Search for Image Classification Under Different Scenes

YU WENG, TIANBAO ZHOU, LEI LIU, AND CHUNLEI XIA
College of Information Engineering, Minzu University of China, Beijing 100081, China

Corresponding author: Tianbao Zhou (tianbaochou@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61772575, and in part by the National Key R&D Program of China under Grant 2017YFB1402101.

ABSTRACT The recent advances in convolutional neural networks (CNNs) have used for image classification to achieve remarkable results. Different fields of image datasets will need different CNN architectures to achieve exceptional performance. However, designing a good CNN architecture is a computationally expensive task and requires expert knowledge. In this paper, we propose an effective framework to solve different image classification tasks using a convolutional neural architecture search (CNAS). The framework is inspired by current research on NAS, which automatically learns the best architecture for a specific training dataset, such as MNIST and CIFAR-10. Many search algorithms have been proposed for implementing NAS; however, insufficient attention has been paid to the selection of primitive operations (POs) in the search space. We propose a more efficient search space for learning the CNN architecture. Our search algorithm is based on Darts (a differential architecture search method), but it considers different numbers of intermediate nodes and replaces some unused POs by channel shuffle operation and squeeze-and-excitation operation. We achieve a better performance than Darts on both the CIFAR10/CIFA100 and Tiny-ImageNet datasets. We retain the none operation in deriving the architecture. The performance of the model has slightly decreased, but the number of architecture parameters has been reduced by approximately 40%. To balance the performance and the number of architecture parameters, the framework can learn a dense architecture for high-performance machines, such as servers, but a sparse architecture for resource-constrained devices, such as embedded systems or mobile devices.

INDEX TERMS Image classification, convolutional neural architecture search, deep learning.

I. INTRODUCTION

With the fast development of the computational capability of hardware, embedded devices such as mobile phones and smart wearables can undertake more computationally intensive tasks such as image or voice processing. Among these tasks, image classification is one of the basic and most challenging problems, which needs to identify what a camera sees in the wild or what food is shown in a photograph, among many other tasks we encounter in everyday life. Image classification has a wide range of applications in different research fields, such as in smart monitoring [1], [2], the smart home [3] and the smart city [4]. It is very necessary to find an efficient and highly accurate classification algorithm. Image classification consists of two essential components: feature extraction from images and the use of the extracted features to classify the category contained in the images. Since Krizhevsky et al. used a deep convolution network to achieve the best performance in the ILSVRC2012 competition, the field of computer vision, especially image classification, has been dominated by variants of convolutional neural networks (CNNs).

A CNN is a feed-forward neural network that extracts image features using multiple convolutional layers and puts them into fully connected layers with a softmax function. When we use the CNN architecture trained by one dataset (e.g., CIFAR-10) on another similar dataset (e.g., CIFAR-100: This dataset is just like the CIFAR-10, except

The associate editor coordinating the review of this manuscript and approving it for publication was Ying Li.

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

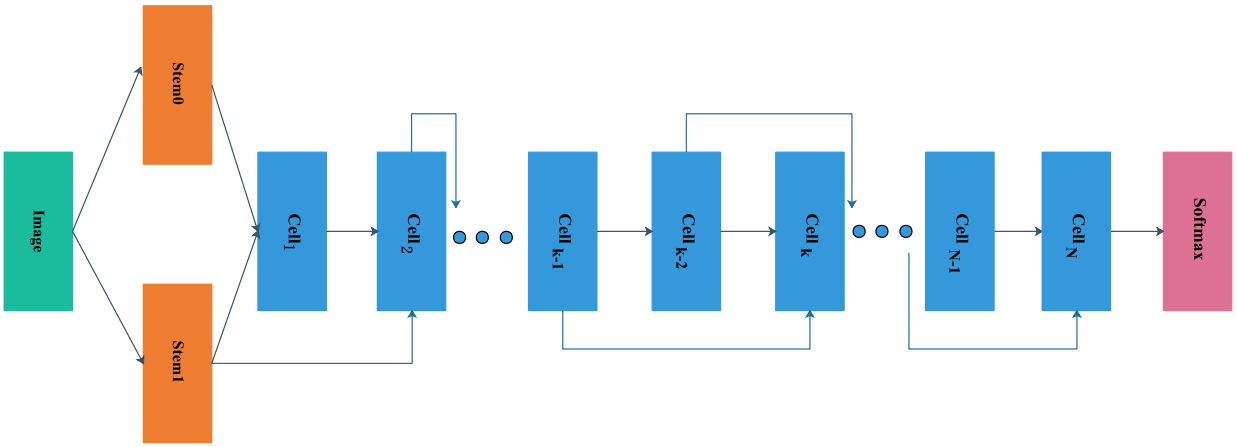


FIGURE 1. Methods of stacking cells proposed by Zoph. Stem0 and Stem1 are auxiliary stem layers to accept the input images.

it has 100 classes containing 600 images each. Both of two are a labeled subset of the 80 million tiny images dataset.), either some tuning, such as fine-tuning specific layers or modifying hyperparameters (which is called transfer learning), or retraining the model from scratch is necessary. However, if a dataset (e.g., Tiny-ImageNet: It runs similar to the ImageNet challenge (ILSVRC) and is very different from CIFAR-10/CIFAR-100 in the scene) is quite different from the dataset used for training, both retraining the model and transfer learning may result in a bad performance. In other words, a new architecture designed for that dataset would be better.

Manually designing new CNN architectures is complicated because many excellent CNN architectures are expertly designed over an extended period of time to achieve a satisfactory result, and an ordinary developer has almost no ability to design a good architecture. Therefore, in recent years, the problem of how to automatically learn an appropriate neural network architecture has attracted the attention of many scholars and experts. In certain tasks, such as image classification, the automatically searched network architecture already has a comparable or even a better performance than the current state-of-the-art manually designed architecture [5]–[8].

Automatic generation of a neural architecture is called neural architecture search (NAS). NAS (to our knowledge) was first proposed by Miller *et al.* [9] in 1989. For the past 10 years, automated network architecture searches have generally used neuro-evolution, which mimics the process of biological evolution in nature to derive a network architecture [10]–[13]. Unfortunately, limited by the lack of computing resources at the time, NAS has not exhibited promising results, similar to those of the neural network itself, in the past 10 years.

With the upsurge of research into deep learning, especially the development of CNNs, excellent CNN architectures, such as Alexnet [14], GoogleNet [15], Xception [16], VGGNet [17], ResNet [18] and DenseNet [19], have emerged. All of these required experienced experts to propose some improved architecture based on an existing network. The discovery of defects in existing network architecture and subsequent improvements can takes several months, even years. Therefore, some researchers have returned their focus to NAS.

The present research on NAS focuses on three aspects: the NAS space definition, search algorithms in NAS and model evaluation. A directed acyclic graph (DAG) is used to represent the network topology architecture, in which each node h_i represents a single latent neuron [10], [11] or the output of the previous layer (e.g., a feature map in the CNN) [6], [20]–[23] and each edge is associated with an operation applied to h_i. Since we are only discussing a CNN, h_i in this paper is an input image or a feature map, and edges represent a convolution operation, a pooling operation, a skip connection, etc. String representation is usually used for a DAG [5], [20], [23]. When the generation method of the DAG is unrestricted, its network architecture space will be very large, which will bring great challenges to the present search algorithm. Therefore, Zoph and Le [5] defined a minimum architecture called a cell (as shown in Figure 1), which has two input nodes: the input of the k^{th} cell, denoted $cell_k$, comes from the output of the cells $k-1$ and $k-2$. When determining the best cell architecture, we can stack the cells into a deeper network using some simple methods. To our knowledge, this idea actually comes from the successful experience of constructing deep networks in the Inception network and Resnet network by stacking basic Inception blocks or Resnet blocks. In this way, the network architecture space will be dramatically reduced, and the task of learning the whole network architecture reduces to learning the cell architecture. Xie and Yuille [20] limit the number of nodes in a cell to further reduce the size of the entire search space. The search algorithms include reinforcement learning (RL) [5]–[7], [24], [25], heuristic algorithms [10], [12], [20], [23], [24], [26]–[28] such as genetic algorithms, the Bayesian optimization

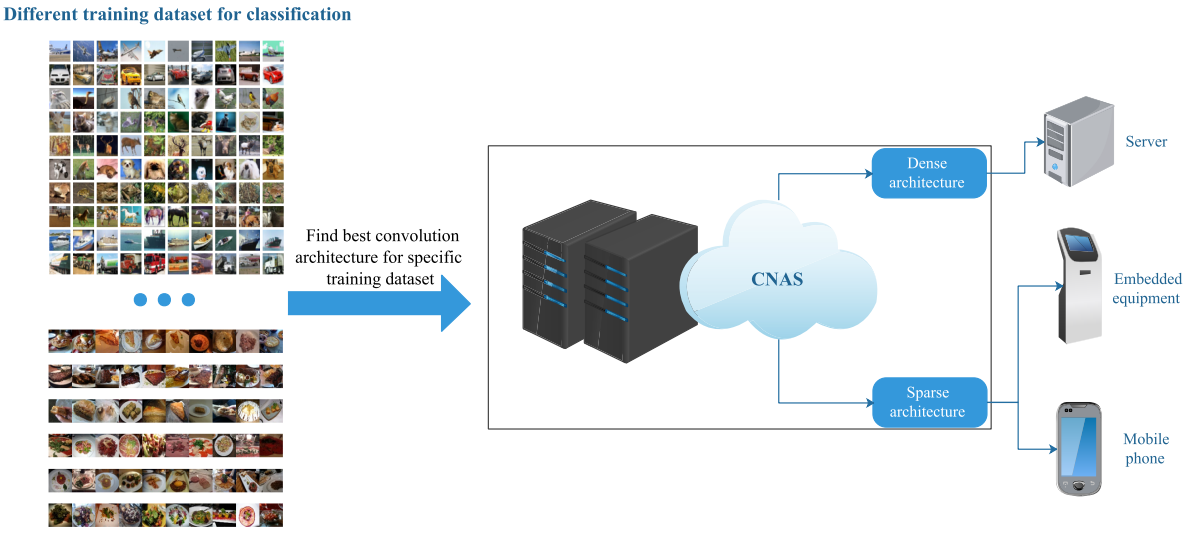


FIGURE 2. Searching for the best CNN architecture for specific datasets and different devices.

method [29], [30], the gradient-based method [22], [31], [32] and Monte Carlo tree search (MCTS) [33]. Model evaluation is divided into two aspects. On the one hand, the performance of the candidate model is evaluated (some works may predict the network performance by advanced ways [34]–[36]) to determine whether the model needs to be expanded or kept for the next update. On the other hand, after selecting the optimal cell, we need to stack the cell into a deeper network and put a specific dataset into it for training and to evaluate the performance.

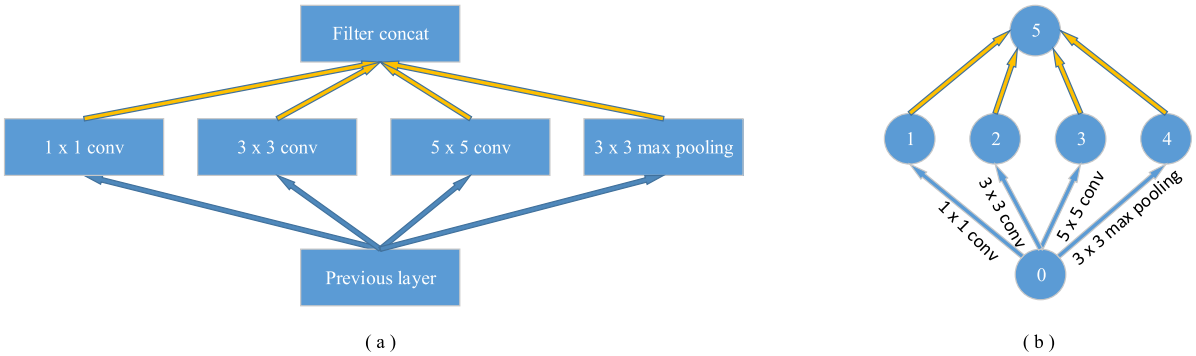
Recently, the work on NAS has mainly focused on reducing the network search space and finding a method that can quickly search for a network architecture as well on evaluating its performance. The network search space includes the topology of nodes and the operations between each connected node. Previous research has introduced many efficient algorithms to generate the topology of nodes; however, there is insufficient discussion about the operations between nodes. This paper will research the selection of operations and the impact of different numbers of intermediate nodes of the searched network architecture on the performance. The network architecture is stacked in the same way as in ENAS [8], which selects a search method to find the best cell architecture and simply stacks N cells along the depth dimension to build the final network architecture. Although we have reduced the original large search space as much as possible by searching for the best cell architecture among a limited number of operations and a limited number of nodes, the number of candidate cell architectures in this space is still exponential [20], [21]. We introduce the differential architecture search method called Darts proposed by Liu *et al.* [22], in which all discrete operations are converted into a continuous space by a softmax function. In this way, the architecture search problem is converted into learning a set of continuous variables. When we evaluate the performance of a network architecture with the validation set, we can optimize the model architecture by gradient descent, similar to updating the network parameters. The difference from Darts is that we focus on the selection of the primitive operations (POs) and the number of intermediate nodes in the search space.

What POs are selected in the search space and the number of intermediate nodes affect the performance of the network searched. As shown in the Table 1, the selection of POs differs in different NAS studies. Thirteen POs are used in NASNet [5], which were reduced to 5 when ENAS [8] was proposed. Convolution operations of size 2×2 were introduced in NAO [37] for enriching the NAS search space. Both Darts [22] and PNASNet [21] add a dilated convolution operation during the search progress. In addition, all of these NAS studies include Evolutionary search [24] share some operations, such as the identity operation, 3×3 separable convolution operation, 3×3 average pooling operation and 3×3 max pooling operation. In this paper, we introduce two extra operations never used in the current methods of NAS and empirically show that we achieve a better performance and fewer network parameters compared to Darts. A sparse architecture is not under consideration since Darts skips the none operation when deriving the cell architecture. We consider both the dense architecture and the sparse architecture during our search work. Our work reveals that a sparse architecture will reduce network parameters by almost 40% while allowing only a small portion of the performance to be sacrificed.

Finally, we develop a framework called CNAS to search for a best CNN architecture for different devices and datasets. We deploy the framework to the GPU-servers and use some service computing technologies [38]–[40] to improve the cooperation between servers. As shown in Figure 2, if we deploy our CNN architecture in a scenario in which computing resources are limited, e.g., mobile phone or embedded

TABLE 1. Selection of POs in current NAS methods.

POs	NAS-Net	ENAS	NAO	DARTS	PNASNet	Evolutionary search
Identity	✓	✓	✓	✓	✓	✓
1×1 **Conv**	✓	×	✓	×	×	✓
1×1 **SepConv**	✓	×	✓	×	×	×
2×2 **Conv**	×	×	✓	×	×	×
2×2 **SepConv**	×	×	✓	×	×	×
3×3 **Conv**	✓	×	×	×	×	×
3×3 **SepConv**	×	✓	✓	✓	✓	✓
3×3 **DilConv**	✓	×	×	✓	✓	×
1×3 *then* 3×1 **Conv**	✓	×	×	×	×	✓
5×5 **SepConv**	✓	✓	×	✓	✓	×
5×5 **DilConv**	×	×	×	✓	×	×
7×7 **SepConv**	✓	×	×	×	✓	×
1×7 *then* 7×1 **Conv**	✓	×	×	×	×	×
2×2 **MP**	×	×	✓	×	×	×
3×3 **MP**	✓	✓	✓	✓	✓	✓
5×5 **MP**	✓	×	×	×	×	×
7×7 **MP**	✓	×	×	×	×	×
2×2 **AP**	×	×	✓	×	×	×
3×3 **AP**	✓	✓	✓	✓	✓	✓
None	×	×	×	✓	×	×



(a) (b)

FIGURE 3. (a) is the Inception v1 model, and (b) is the corresponding computational graph.

devices, a sparse architecture is better than a dense one. In summary, our contributions are as follows:

1) We use more efficient and rich POs to extend the CNN search space by adding the channel shuffle operation and squeeze-and-excitation operation.
2) We show the impact of selecting a sparse architecture and a dense architecture on the performance of the final network based on the differential architecture search algorithms and develop a CNAS framework to automatically learn the best CNN architecture for different image classification scenes.
3) We achieve a better classification performance than Darts, which we attribute to our search space (both in CIFAR10/CIFAR100 and Tiny-ImageNet when the epoch number is 200 and without using auxiliary layers).

We have released our code at https://github.com/tianbaochou/CNAS.

II. THE CNAS FRAMEWORK

In this section, we will first describe the basic representation of the CNN architecture. We will briefly describe how to represent a network architecture as a DAG. Then, we will focus on the definition of the network architecture search space and the selection of POs and discuss the size of the space we choose. Following that is a description of the search algorithm we used. After that, we will explain how to build a deeper network architecture based on the architecture of searched cells. Finally, we will describe how to build an efficient automatic convolutional neural architecture search framework to fit different image classification tasks.

A. ARCHITECTURE REPRESENTATION

The neural network architecture itself has a strong topology. As shown in Figure 3(a), which depicts the Inception v1 module, we regard a group of feature maps as a node, and operations such as convolution, skip connect or pooling represent edges. Figure 3(b) is a diagram of the DAG transformed from the Inception v1 module, where the orange arrow indicates the aggregation operation.

In general, the CNN architecture α can be represented by a graph $G_\alpha(V_\alpha, E_\alpha)$. Each node $v \in V_\alpha$ is associated with an image input or a feature map, and each directed edge

$e_{(u,v)} \in E_\alpha$ is associated with an operation applied to the u node that was transformed to v.

B. SEARCH SPACE

Recently, the work on NAS has mainly focused on reducing the network search space and finding a method that can quickly search network architectures as well as on evaluating the network's performance. The network search space includes the topology of nodes and the operations between each node. Previous research has introduced many efficient algorithms to generate the topology of nodes; however, there is no reasonable discussion about the operations between nodes. This paper will research the selection of operations and the impact of different numbers of intermediate nodes of the searched network architecture on the performance.

1) The SELECTION OF POs

The choice of operations needs to meet three conditions: uniqueness, fewer parameters and fast calculation. Fewer parameters means that simple POs will be put into the search space in order to consume less GPU or RAM memory resources during the search process, fast calculation is vital to speed up both architecture search and model running, and uniqueness here means that each operation has some unique properties that cannot be replaced by the others. Large receptive fields, such as a 5×5 size convolution and a 7×7 size convolution, can be replaced by stacking 3×3 size convolutions along the depth. Therefore, all convolution operations and pooling operations are limited by a 3×3 size. We analyze the CNN architectures that have achieved significant performance in image classification, image segmentation and object detection in recent years. The following POs are selected.

- *none*
- *identity*
- *average-pooling*
- *max-pooling*
- *separable convolution*
- *dilation convolution*
- *squeeze-and-excitation*
- *channel shuffle convolution*

The none operation means that there is no connection between two nodes which is considered when we need generate a sparse architecture. The identity operation leads to the previous feature maps being added to the current layer's feature maps by skipping the intermediate layer. Therefore, information on different layers can appear in the same layer, which can smooth the loss landscape and help backpropagation escape a locally optimal solution [41]. The main role of average-pooling and max-pooling is to refine the feature from the previous feature map and reduce its dimensions. Separable convolution operation can dramatically reduce network parameters without sacrificing network performance due to a pair of (depthwise, pointwise) operations. A dilation convolution operation can let a 3×3 size convolution view a larger field of view in the current layer. When its dilation is 2, it can see the 7×7 size field of view, which enables the current layer's feature map to respond to different ranges of information at the same time. As we mentioned before, to speed up the search process without reducing performance, we use group convolution for all convolution operations, and each uses 1×1 convolution to concatenate feature maps in the end.

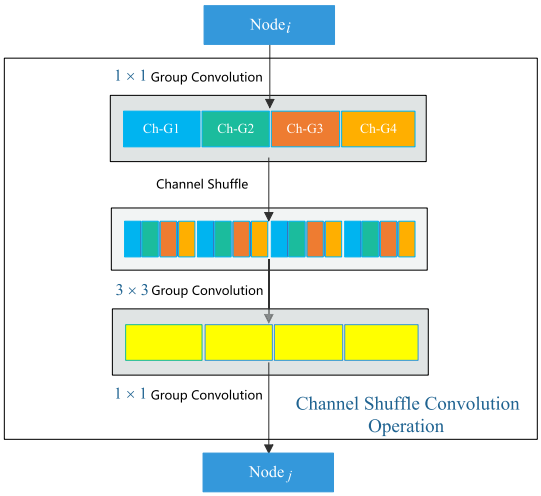


FIGURE 4. The channel shuffle convolution operation.

The above 6 POs have been used in current NAS methods [5], [8], [21], [37], but all of these methods do not consider the squeeze-and-excitation [42] operation and the channel shuffle convolution [43] operation.

Group convolution is one of the major contributions in AlexNet, in which channels of features are evenly distributed into different groups, and finally, the features are merged through two fully connected layers so that the features between the different groups can only be merged at the last moment. This approach is quite unfavorable for the generalization ability of the network. To solve this problem, ShuffleNet performs a channel shuffle every time before stacking the group convolutional layer (we call this the channel shuffle convolution operation, as shown in Figure 4), and the shuffled channels are allocated to different groups. After the channel shuffle, the output features after group convolution can take into account more channels of information and are more representative (meaning a better result of information fusion).

In Inception, DenseNet, or ShuffleNet, the features we generate for all channels are directly combined evenly. However, there is no reason why the features of all channels are equal in the model, so a better method is to automatically learn the weights of every channel. As shown in Figure 5, before starting, we perform a normal convolution and obtain a feature map that has two routes; the first route does nothing before the end of the second route, and the second route performs the squeeze operation (global average pooling) to compress the 2-dimensional features of each channel into one dimension. After that, a feature channel vector is obtained (each

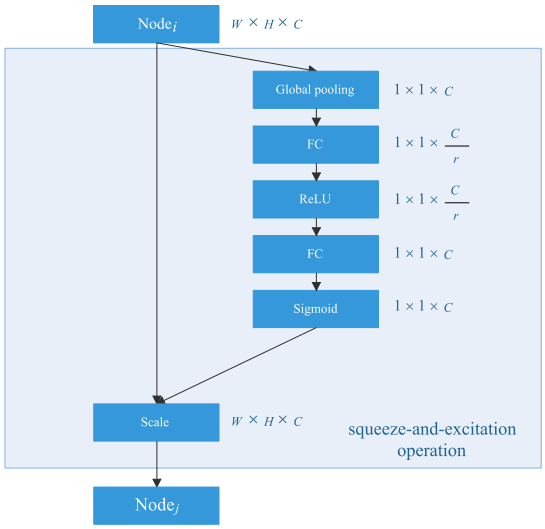


FIGURE 5. The squeeze-and-excitation operation.

number represents the features of the corresponding channels). Then, an excitation operation is performed, the column feature channel vector is put into two fully connected layers (the experience value of r is 16) and a sigmoid function, and the correlation between the feature channels is modeled. The obtained output is actually the weights corresponding to each channel, and the original features (the first route) are finally weighted by these weights. These procedures are exactly what the squeeze-and-excitation operation does. In short, the squeeze-and-excitation operation suppresses some redundant features and enhances useful features by assigning weights to feature channels.

2) NUMBER OF INTERMEDIATE NODES

The number of intermediate nodes in a cell determines the complexity of the searched network architecture. This paper will experimentally observe the impact of different numbers of intermediate nodes on the final performance of the model and select the appropriate number of intermediate nodes for training of the network to obtain the best performance on each set of training data.

3) SIZE OF THE SEARCH SPACE

Assume that the PO set is O, the size of intermediate node set is M, and the output of the cell is aggregated by all the intermediate nodes in the cell. If we do not limit the predecessor node of each intermediate node, then the entire search space will be very large. To reduce the size of search space, we number the intermediate nodes as $n_0, n_1, \cdots n_{(M-1)}$, similar to in Darts:

$$n_i = \sum_{j<i} o_{i,j}(n_j). \tag{1}$$

where n_i is the i^{th} node, and $o_{i,j}$ represents the n_j applied in an operation to obtain n_i. The size of the entire search space is:

$$|O|^{2M+\frac{M(M-1)}{2}}. \tag{2}$$

where $|O|$ is the dimension of the PO set, which is 8 in our paper. When $M = 4$, the size of the search space is 8^{14}; when $M = 5$, the size of the search space will dramatically increase to 8^{20}. We can see that varying the number of intermediate nodes has a huge impact on the size of the search space.

C. SEARCHING ALGORITHM

In this paper, we choose Darts, a differential architecture search method proposed by Liu et al., as our searching algorithm, in which all discrete operations are converted into a continuous space by a softmax function. In this way, the architecture search problem is converted into learning a set of continuous variables. When we evaluate the performance of a network architecture with the validation set, we can optimize the model architecture by gradient descent, similar to updating the network parameters.

Darts removes the meta-controller(or hypernetwork) by modeling NAS as a single training process of an over-parameterized network that comprises all candidate paths. Suppose $O = \{o_i\}$ is the set of N candidate primitive operations. To build the over-parameterized network that includes all architecture in the search space, Darts sets each edge to be a mixed operation that has N parallel path, denote as $MixO$. Given input x, the output of a mixed operation $MixO$ is defined based on the outputs of its N path and $MixO(x)$ is weighted sum of $o_i(x)$:

$$MixO(x) = \sum_{i=1}^{N} w_i o_i(x). \tag{3}$$

We can see from 3, if a primitive operation has a great contribution to the edge, its weight should be larger than others, which is a good metric that measures whether the operation is useful. That is why we use Darts instead of other search algorithms for verifying the validity of the two primitive operations introduced.

D. STACKING CELLS

The stacking methods of cells can also be determined by some search algorithm similar to the cell architectures we search, but the focus of this paper is not on this aspect. For the sake of fairness, we simply stack the cells along the depth, and the first and second nodes of cell k are set equal to the outputs of cell $k-2$ and cell $k-1$, respectively; auxiliary convolution layers with stride 2 are inserted as necessary (if the input image size is large, e.g., in ImageNet). At the same time, to reduce the size of the entire network, cells located at 1/3 and 2/3 of the total depth of the network are reduction cells (both in the process of searching and evaluation), for which all the operations adjacent to the input nodes are of stride two which will reduce the usage of memory and accelerate both searching and training.

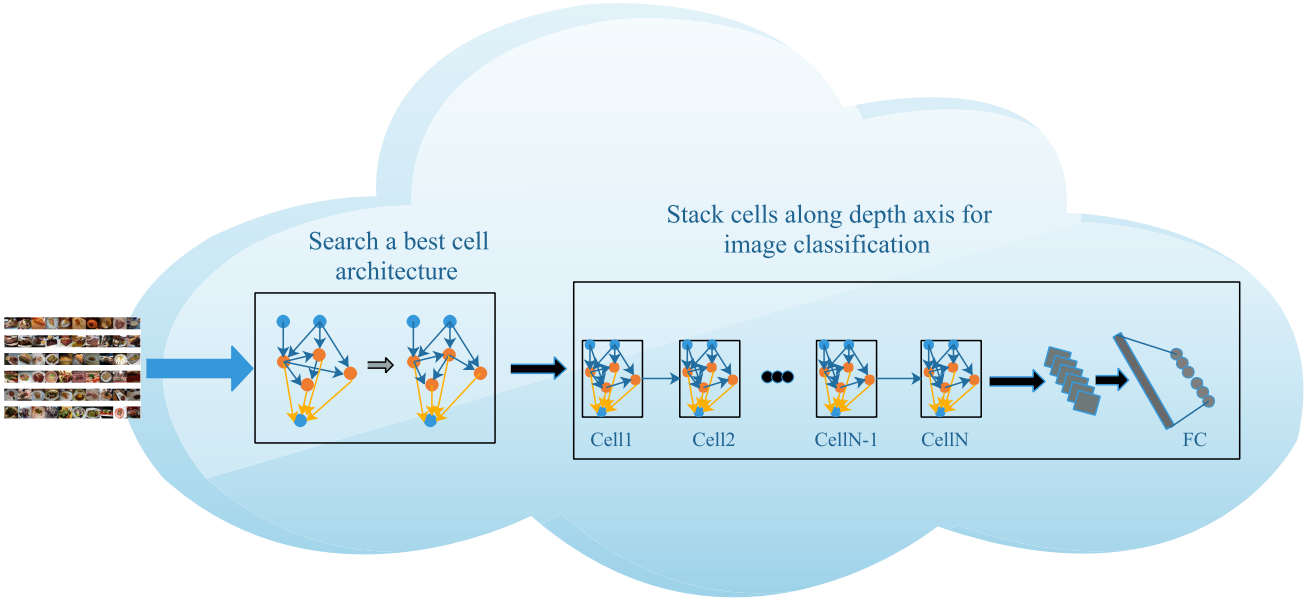


FIGURE 6. The framework of CNAS, which contains two steps; the first step is searching for a best cell for the dataset, and the second step is stacking the cells searched to obtain a deeper network. In the last cell, we need to link fully connected layers for image classification.

E. BUILDING THE CNAS FRAMEWORK

As show in Figure 6, based on the above search algorithm and the search space mentioned, we build a framework for automatically learning the convolutional neural architecture. When we need to find the best CNN architecture for a specific dataset, we can put the dataset into the search process to find the best normal and reduction cell architectures and retrain the architecture by stacking enough of the cells that we searched. All components in CNAS are written in Python and are recommended to run on servers with GPUs. The GPU cluster can be deployed to the cloud, and the network architecture configuration [44] is automatically recommended according to the target device information selected by the user [45], [46].

III. EXPERIMENTS

This article will perform search experiments on the CIFAR10 dataset. Based on Darts, we delete the 5×5 size separable convolution and 5×5 size dilation convolution in the original implementation (which not contribute to final architecture searched, so we call them unused POs) and add a 3×3 size squeeze-and-excitation operation and a 3×3 size channel shuffle convolution. At the same time, Darts leaves the none operation in the mixed operation but did not consider it when deriving the architecture from encoding (which we call dense architecture α). In our experiment, we found that the weight of the none operation will increase to a high value (e.g., over 0.8) during searching and that the normal cell searched will be very sparse (which we call sparse architecture $\tilde{\alpha}$), which means there are much fewer network parameters than in the dense architecture. At the same time, although the computational capability of embedded devices has been greatly improved, it is not sufficient to address network models with many parameters. Therefore, this paper will consider both none operation and non-none operation.

For comparison with Darts, we set convolution operations to have stride 1 and add padding to preserve the feature map dimension. The mixed operation is actually a weight sharing [8], [25], [29] method, which avoids retraining the currently generated network architecture by sharing the sub-network architecture's weights in the search process, thus saving much searching time. The authors of Darts proposed using second order approximation when updating network architecture parameters, but this would take more than twice as much time compared to first order approximation (first order approximation needs to perform 2 forward propagations and 2 backward propagations to update the architecture parameters α, while second order approximation requires 4 forward propagations and 4 backward propagations). Additionally, during experimentation, we found that the performance of the architecture optimized by second order approximation exhibits almost no improvement compared to that of the architecture optimized by first order approximation. Therefore, the search time is measured by using first order approximation for cell architecture searching. All of our experiments were performed using an NVIDIA GTX TITAN PASCAL GPU.

A. SEARCHING CELLS ON CIFAR-10

The CIFAR-10 dataset contains $50k$ and $10k$ images, which is categorized into 10 classes, for training and testing, and each image has a size of 32×32. It is much smaller than the ImageNet dataset, and the training images are sufficiently rich. Thus, we search our best cells on CIFAR-

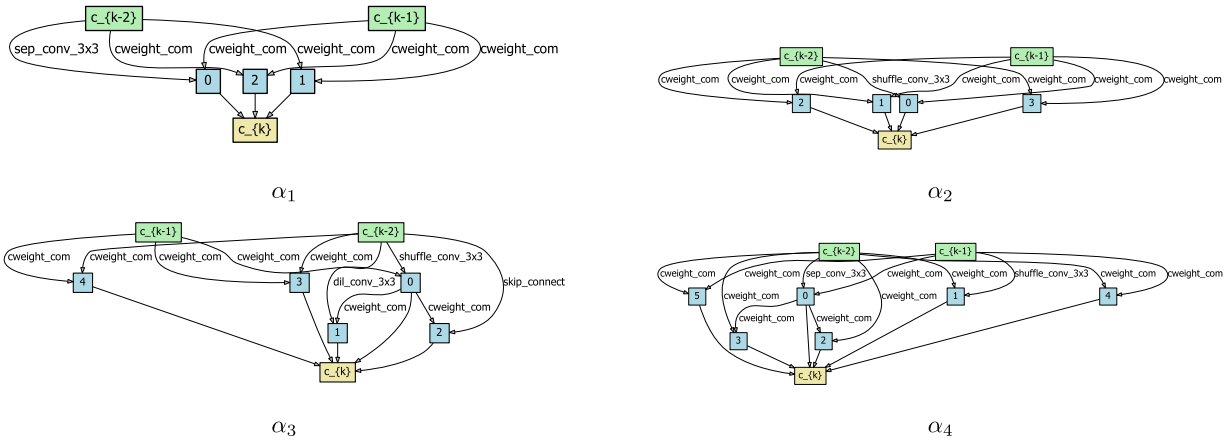


FIGURE 7. Normal cell architecture in the searched architecture $\alpha_i (i = 1, 2, 3, 4)$. cweight_com indicates the squeeze-and-excitation operation.

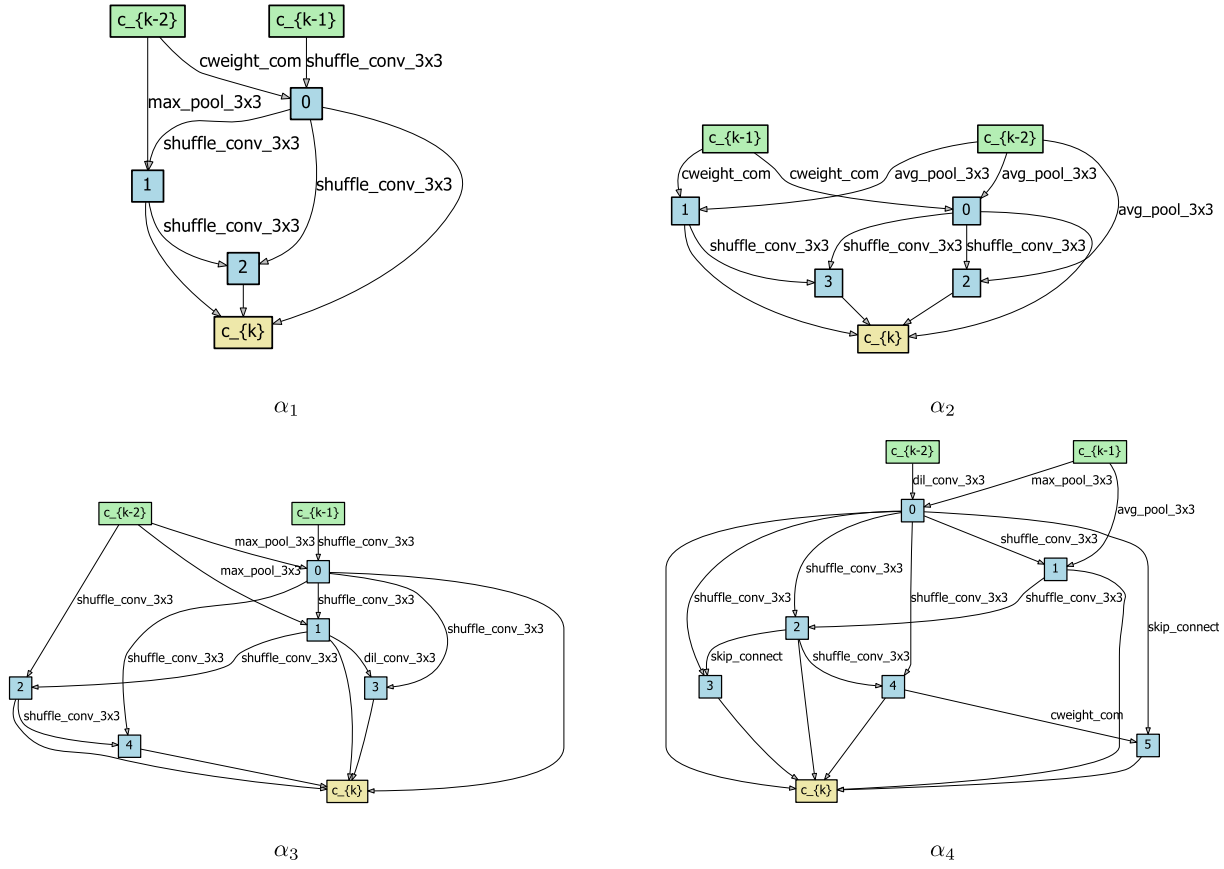


FIGURE 8. Reduction cell architecture in the searched architecture $\alpha_i (i = 1, 2, 3, 4)$.

10 and transfer them to CIFAR-100 (similar to CIFAR-10, CIFAR-100 also contains $50k$ and $10k$ images for training and testing, but categorized into 100 classes.) and Tiny-ImageNet.

The number of intermediate nodes is selected as $M = 3, 4, 5$, and 6, and the network architecture searched on CIFAR-10 is $\alpha_i (i = 1, 2, 3, 4)$, respectively. We keep half of the training data as the validation set. To obtain each architecture α_i, a small network obtained by stacking 6 cells is trained for 50 epochs with batch size 64, and we use the classification error rate on the validation set as the performance of the cells we searched.

The search time and model parameter size of α_i are shown in Table 2.

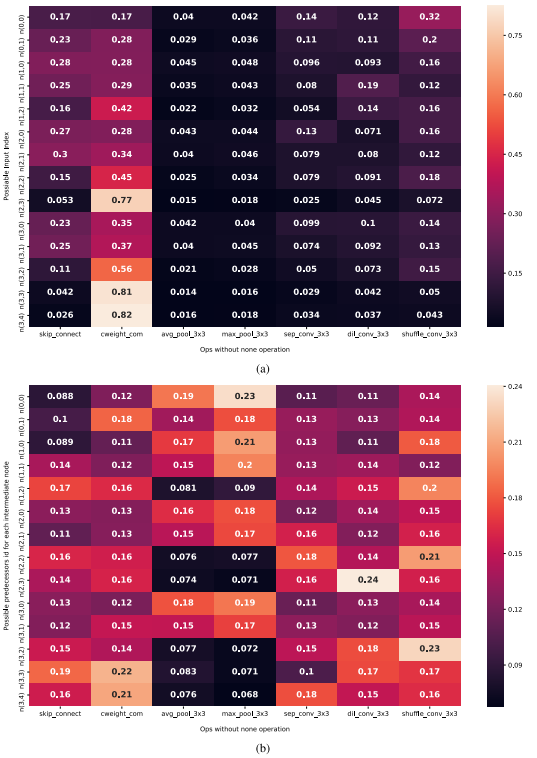


FIGURE 9. (a) is the weights of POs without the none operation in normal cells when the number of intermediate nodes is 4; (b) is similar but in reduction cells. $n(i, j)$ indicates that the predecessor's id is j for the intermediate node with id i. The values in the rectangles indicate the specific operation's weight in mixed operation.

TABLE 2. Time and model parameter costs for 4 architecture searched on CIFAR-10.

ARCHITECTURE	SEARCH TIME (DAYS/GPU)	MODEL PARAM (M)
α_1	0.25	1.73
α_2	0.5	2.04
α_3	1.1	2.80
α_4	2.5	2.95

The architectures of the corresponding normal cell and reduction cell are shown in Figure 7 and Figure 8, respectively. As shown in Figure 7, the proportion of the squeeze-and-excitation operation in the normal cell is very large, followed by the channel shuffle convolution operation. Similarly, the channel shuffle convolution operation accounts for the largest proportion in the reduction cell, followed by the max pooling operation, as Figure 8 shows. The weight distribution for POs without the none operation is shown in Figure 9. We can see that the squeeze-and-excitation operation almost has the largest weight applied for each candidate predecessor node in normal cells, which means that the optimization of the architecture of normal cells in this search space works well, as we expected (as show in Figure 9(a)). There is little difference between the colors of each column in a reduction cell (as show in Figure 9(b)) due to fewer reduction cells being used in the architecture search and insufficient optimization compared to normal cells.

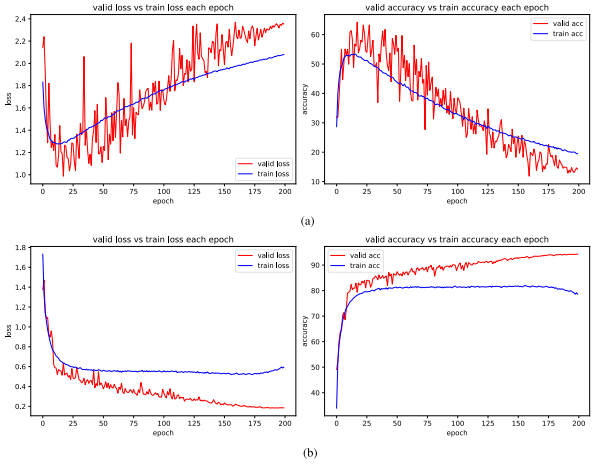


FIGURE 10. Comparison of loss and accuracy changes in the training process of $\tilde{\alpha_2}$ and $\tilde{\alpha_3}$ on CIFAR-10.

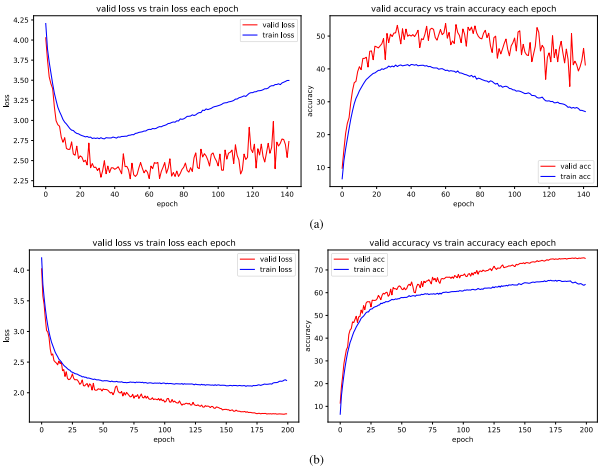


FIGURE 11. Comparison of loss and accuracy changes in the training process of $\tilde{\alpha_2}$ and $\tilde{\alpha_3}$ on CIFAR-100.

All of these results show that the search space proposed in this paper is better than the search space used in Darts in the image classification task.

B. EVALUATION OF THE ARCHITECTURE

The architectures $\alpha_i, (i = 1, 2, 3, 4)$ searched above are trained on CIFAR-10, CIFAR-100 and Tiny-ImageNet to evaluate their performance. For the sake of fairness, we train all our models using 200 epochs without auxiliary tower layers [16] in the path for every dataset and add an auxiliary drop path [47] with a fixed drop probability 0.25. The same data augmentation (e.g., normalization, random crop and random horizontal flip) is applied to each dataset. The error rate is selected as the evaluation metric since all the datasets are used for classification.

1) CIFAR-10 AND CIFAR-100

We use a large network of 20 cells (note that a much small network with 6 cells is stacked for searching, because the

TABLE 3. Classification error rate for α_i, $(i = 1, 2, 3, 4)$.

Dataset	Architecture	Top1	Top5	Evaluation Time	Model Size
CIFAR-10	α_1	6.38	–	12h	1.73M
	α_2	5.8	–	1d-1h	2.04M
	α_3	4.7	–	17h	2.80M
	α_4	**4.23**	–	22h	2.95M
CIFAR-100	α_1	25.53	6.18	13h	1.89M
	α_2	24.65	5.82	18h	2.27M
	α_3	23.26	5.44	1d-2h	3.08M
	α_4	**22.24**	**5.01**	1d-23h	3.67M
Tiny-ImageNet	α_1	45.81	21.17	2d-13h	1.58M
	α_2	41.91	19.81	2d-15h	1.89M
	α_3	38.26	16.34	1d-2h	2.59M
	α_4	**35.90**	**9.89**	3d-12h	2.71M

TABLE 4. Comparison with Darts on CIFAR-10.

Model	Accuracy	Evaluation time	Model size
Darts	4.74	23h	3.16M
Ours	**4.23**	22H	2.95M
Ours (with none)	5.4	**18H**	**1.92M**

TABLE 5. Comparison with Darts on CIFAR-100.

Model	Top1	Top5	Evaluation time	Model size
Darts	23.22	5.6	12h	3.03M
Ours	**22.24**	**4.54**	22h	3.67M
Ours (with none)	24.20	5.8	**9h**	**2.03M**

GPU memory will occupied too much if we stack 20 cells.) for training over 200 epochs with a batch size of 96 or 64 (when the number of intermediate nodes is 6). Other hyperparameters remain the same as the ones used for the architecture search, similar to Darts.

We have found that the performance of the network is almost unchanged after the number of intermediate nodes goes from 5 to 6, but the network parameter increases by approximately 6% and the evaluation time by approximately 30%. As we mentioned before, we consider both none operation and non-none operation. Suppose that the architecture searched by adding a none operation is $\tilde{\alpha_i}(i = 1, 2, 3, 4)$. When we evaluated the networks on CIFAR-10, we found that $\tilde{\alpha_1}$ and $\tilde{\alpha_2}$ performed badly. From Figure 10(a) and 10(b), we can see that the loss of $\tilde{\alpha_2}$ rebounded around epoch 15, and then, the loss rose until the end; however, the loss of $\tilde{\alpha_3}$ decreased over time. Far worse is that a similar situation exists for CIFAR-100 (as shown in Figure 11(c) and 11(d). We observe four network architectures and find that there is only one non-none operation in the normal cell architecture of $\tilde{\alpha_1}$ and $\tilde{\alpha_2}$, but there are two non-none operations in $\tilde{\alpha_3}$ and $\tilde{\alpha_4}$, which indicates that $\tilde{\alpha_1}$ and $\tilde{\alpha_2}$ have encountered overfitting on the CIFAR-10 dataset when searching. Overfitting leads to poor generalization of the model during real training and lack of transferability.

Although the network architectures $\tilde{\alpha_1}$ and $\tilde{\alpha_2}$ perform poorly when using the current hyperparameters, they may be trained well as long as the hyperparameters are properly adjusted. To unify the parameters used in each training data we trained on, this paper does not conduct in-depth research. To clarify the sparse architecture we used, when we mention sparse architecture, it means $\tilde{\alpha_4}$.

We compare the architecture that performs best on each dataset in Table 3 with the architecture obtained by Darts. As can be seen from Table 4 and Table 5, our model obtained the best performance on both CIFAR-10 and CIFAR-100. It is particularly worth mentioning that although the architecture achieved by allowing the use of the none operation when deriving it, $(\tilde{\alpha_4})$, is one percent less accurate than Darts, the parameter size of the entire model is reduced by approximately 40%.

2) TINY-IMAGENET

The Tiny-ImageNet dataset contains images with 200 different categories, and each image has a size of 64×64. The training set has $100k$ images, and the validation set has $10k$ images (50 images per category). We use a network with 12 cells for training the model, and the initial learning rate is 0.05. The images in Tiny-ImageNet differ greatly from the images in CIFAR-10; it is necessary to compare the performance of the architecture obtained by searching Tiny-ImageNet itself and the transferred architecture obtained by searching CIFAR-10. As shown in Table 6, the performance of the classification network transferred from CIFAR-10 is worse than that of the network trained using Tiny-ImageNet itself. In short, when

TABLE 6. Comparison with different architecture on Tiny ImageNet.

Model	Top1	Top5	Evaluation time	Model size
Darts	38.60	20.5	3d-18h	3.03M
Ours (search with CIFAR-10)	38.10	18.89	3d-22h	**2.71M**
Ours (search with Tiny-ImageNet)	**36.00**	**15.34**	**3d-12h**	2.26M

the new dataset is very different from the original dataset, re-searching the network architecture with the new dataset tends to result in a better performance.

IV. CONCLUSION

We proposed a more effective framework for solving different image classification problems using convolutional neural architecture search (CNAS), which can automatically learn the best CNN architecture for a specific dataset. We show that both the channel shuffle convolution operation and squeeze-and-excitation operation are almost the only two operations selected for normal cells in classifying tasks after searching. This result may be useful for reducing the search space further. Although the performance of a sparse architecture will decrease a bit compare to a dense architecture, the number of parameters of the model will reduced by approximately 40%, which is very useful in some IOT scenarios in which hardware resources are limited (e.g., mobile phones and embedded deviced). Our best architecture has achieved a higher performance than Darts on all Image dataset we used. In the future, we would like to use our search space for other differential architecture search methods such as NAO [37] or for many discrete architecture search methods, because deriving a sparse architecture may fail (we found that when using Tiny-ImageNet dataset to search a sparse architecture, the normal cell architecture will only contain none operations, which can be trained in the architecture search process because we mix up all candidate operations. however, holes will exist on the network architecture stacked by the cell during the evaluation phase, resulting the inability of the information flow to be transmitted). Additionally, performing deep research on the stacking of cell in more flexible manners is necessary.

REFERENCES

[1] K. Muhammad, J. Ahmad, I. Mehmood, S. Rho, and S. W. Baik, "Convolutional neural networks based fire detection in surveillance videos," *IEEE Access*, vol. 6, pp. 18174–18183, 2018.

[2] K. Muhammad, R. Hamza, J. Ahmad, J. Lloret, H. Wang, and S. W. Baik, "Secure surveillance framework for iot systems using probabilistic image encryption," *IEEE Trans. Ind. Informat.*, vol. 14, no. 8, pp. 3679–3689, Aug. 2018.

[3] S. T. M. Bourobou and Y. Yoo, "User activity recognition in smart homes using pattern clustering applied to temporal ANN algorithm," *Sensors*, vol. 15, no. 5, pp. 11953–11971, May 2015.

[4] K. Muhammad, M. Sajjad, and S. W. Baik, "Dual-level security based cyclic18 steganographic method and its application for secure transmission of keyframes during wireless capsule endoscopy," *J. Med. Syst.*, vol. 40, no. 5, p. 114, May 2016. doi: 10.1007/s10916-016-0473-x.

[5] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Feb. 2017, pp. 1–16.

[6] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2018, pp. 8697–8710.

[7] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Mar. 2017, pp. 1–18.

[8] H. Q. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018.

[9] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann, 1989, pp. 379–384. [Online]. Available: http://dl.acm.org/citation.cfm?id=93126.94034

[10] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[11] K. O. Stanley, "Efficient evolution of neural networks through complexification," Ph.D. dissertation, Dept. Comput. Sci., Univ. Texas at Austin, Austin, TX, USA, 2004. [Online]. Available: http://nn.cs.utexas.edu/?stanley:phd2004

[12] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artif. Life*, vol. 15, no. 2, pp. 185–212, Apr. 2009. doi: 10.1162/artl.2009.15.2.15202.

[13] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evol. Intell.*, vol. 1, no. 1, pp. 47–62, Mar. 2008. doi: 10.1007/s12065-007-0002-4.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[15] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[16] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Apr. 2015, pp. 1–14.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[19] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[20] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1388–1397.

[21] C. Liu *et al.*, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 19–34.

[22] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Jun. 2018, pp. 1–12. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX

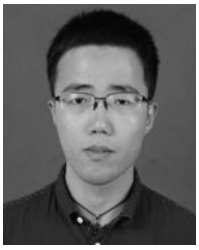
[23] T. Elsken, J. H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Feb. 2018, pp. 1–14. [Online]. Available: https://openreview.net/forum?id=SySaJ0xCZ

[24] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Feb. 2018, pp. 1–13. [Online]. Available: https://openreview.net/forum?id=BJQRKzbA-

[25] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. 32nd AAAI Conf. Artif. Intell.*, Apr. 2018, pp. 1–8.

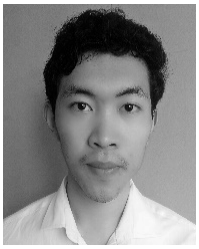
[26] O. E. David and I. Greental, "Genetic algorithms for evolving deep neural networks," in *Proc. Annu. Conf. Genetic Evol. Comput. (GECCO)*, Jul. 2014, pp. 1451–1452.
[27] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, Jul. 2017, pp. 497–504.
[28] R. Miikkulainen *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, R. Kozma, C. Alippi, Y. Choe, and F. C. Morabito, Eds. Amsterdam, The Netherlands: Elsevier, 2018. [Online]. Available: http://nn.cs.utexas.edu/?miikkulainen:chapter18
[29] H. Jin, Q. Song, and X. Hu. (2018). "Auto-Keras: Efficient neural architecture search with network morphism." [Online]. Available: https://arxiv.org/abs/1806.10282
[30] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *Proc. 32nd Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2018, pp. 2020–2029.
[31] R. Shin, C. Packer, and D. Song, "Differentiable neural network architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Jun. 2018, pp. 1–4.
[32] W. Grathwohl, E. Creager, S. Kamyar, S. Ghasemipour, and R. Zemel, "Gradient-based optimization of neural network architecture," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Jun. 2018, pp. 1–6.
[33] R. Negrinho and G. J. Gordon. (2018). "Deeparchitect: Automatically designing and training deep architectures." [Online]. Available: https://arxiv.org/abs/1704.08792
[34] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2018, pp. 2423–2432.
[35] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with Bayesian neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Mar. 2017, pp. 1–16.
[36] B. Deng, J. Yan, and D. Lin. (2017). "Peephole: Predicting network performance before training." [Online]. Available: https://arxiv.org/abs/1712.03351
[37] R. Luo, F. Tian, T. Qin, and T.-Y. Liu, "Neural architecture optimization," in *Proc. 32nd Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2018, pp. 7827–7838.
[38] H. Gao, Y. Duan, H. Miao, and Y. Yin, "An approach to data consistency checking for the dynamic replacement of service process," *IEEE Access*, vol. 5, pp. 11700–11711, 2017.
[39] H. Gao, H. Miao, L. Liu, J. Kai, and K. Zhao, "Automated quantitative verification for service-based system design: A visualization transform tool perspective," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 28, no. 10, pp. 1369–1397, 2018. doi: 10.1142/S0218194018500390.
[40] Y. Yin, Y. Xu, W. Xu, M. Gao, L. Yu, and Y. Pei, "Collaborative service selection via ensemble learning in mixed mobile network environments," *Entropy*, vol. 19, no. 7, p. 358, Jul. 2017.
[41] Y. Li and Y. Yuan, "Convergence analysis of two-layer neural networks with ReLU activation," in *Proc. NIPS*, 2017, pp. 597–607.
[42] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2018, pp. 7132–7141.
[43] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2018, pp. 6848–6856.
[44] H. Gao, W. Huang, X. Yang, Y. Duan, and Y. Yin, "Toward service selection for workflow reconfiguration:an interface-based computing solution," *Future Gener. Comput. Syst.*, vol. 87, pp. 298–311, Oct. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17320575
[45] Y. Yin, L. Chen, Y. Xu, and J. Wan, "Location-aware service recommendation with enhanced probabilistic matrix factorization," *IEEE Access*, vol. 6, pp. 62815–62825, 2018.
[46] Y. Yin, F. Yu, Y. Xu, L. Yu, and J. Mu, "Network location-aware service recommendation with random walk in cyber-physical systems," *Sensors*, vol. 17, no. 9, p. 2059, Sep. 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/PMC5621120/
[47] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Nov. 2017, pp. 1–11.

YU WENG received the Ph.D. degree in computer science from the University of Science and Technology Beijing, China, in 2010. He is currently an Associate Professor of computer science with the Information Engineering Department, Minzu University of China. He has published more than 20 conference and journal papers. His current research interests include machine learning, cloud computing, and distributed computing.



TIANBAO ZHOU is currently pursuing the master's degree with the Information Engineering Department, Minzu University of China. His major research interests include machine learning and computer vision.

LEI LIU is currently a master's degree in computer science and technology with the College of Information Engineering, Minzu University of China. His current research interests include message-oriented middleware and anomaly detection.



CHUNLEI XIA is currently pursuing the master's degree with the Information Engineering Department, Minzu University of China. His major research interests include machine learning and pattern recognition.