Partial Connection Based on Channel Attention for Differentiable Neural Architecture Search

Yu Xue, Member, IEEE, and Jiafeng Qin

Abstract—Differentiable neural architecture search (DARTS), as a gradient-guided search method, greatly reduces the cost of computation and speeds up the search. In DARTS, the architecture parameters are introduced to the candidate operations, but the parameters of some weight-equipped operations may not be trained well in the initial stage, which causes unfair competition between candidate operations. The weight-free operations appear in large numbers, which results in the phenomenon of performance crash. Besides, a lot of memory will be occupied during training supernet, which causes the memory utilization to be low. In this article, a partial channel connection based on channel attention for differentiable neural architecture search (ADARTS) is proposed. Some channels with higher weights are selected through the attention mechanism and sent into the operation space while the other channels are directly contacted with the processed channels. Selecting a few channels with higher weights can better transmit important feature information into the search space and greatly improve search efficiency and memory utilization. The instability of network structure caused by random selection can also be avoided. The experimental results show that ADARTS achieved 2.46% and 17.06% classification error rates on CIFAR-10 and CIFAR-100, respectively. ADARTS can effectively solve the problem that too many skip connections appear in the search process and obtain network structures with better performance.

Index Terms—Channel attention, image classification, neural architecture search, partial connection.

I. INTRODUCTION

Deep neural network [1], [2] is an important research topic in deep learning [3]. Compared with shallow neural network, its multilayer network structure can extract richer and more complex feature information to obtain higher performance. Therefore, it has made remarkable progress in semantic recognition [4], image recognition [5], and data forecasting [6], [7]. The performance of deep neural network mainly depends on network parameters and structure. The vital task is to design appropriate strategies to optimize the structure and parameters of the neural network to improve its final performance. At present, gradient methods are mainly used to optimize network parameters, such as SGD optimizer [8] and ADAM optimizer [9]. However, unlike parameter optimization, how to build a better performance network structures requires a lot of expert experience. For example, some of the better neural networks, such as VGG [10] and ResNet [11], were designed by experts. But all of them are designed manually by using the trial and error manner. A lot of time and resources are taken to design these networks, and different network structures need to be built for different problems and datasets. Neural architecture search (NAS) [12] as an effective method can automatically search network structures with higher performance.

As a part of automated machine learning (AutoML), the NAS builds different neural network structures by constructing a large search space and search strategies. In NAS, the search space, the appropriate search method, and the evaluation method are three main tasks.

In general, there are many methods that can be used for architectural search, such as reinforcement learning (RL) [13], evolutionary computing (EC) [14], and gradient-based. Reinforcement learning regards the search of neural network structure as an agent’s action. The network is constructed through different behaviors and rewards, which are based on the evaluation of the network performance on the test set. Representing and optimization of the agent policy are the two keys to using reinforcement learning to search network architecture. Zoph et al. [15] used recurrent neural network (RNN) strategies to sequentially sample strings and then encode the neural architecture. Baker et al. [16] used a q-learning training strategy, which in turn selects the type of layer and corresponding hyperparameters. However, the method based on reinforcement learning consumes extremely computing resources. For example, Zoph [15] used 800 GPUs to complete the search process for three to four weeks. Therefore, as another method to replace reinforcement learning, evolutionary computing can reduce computational consumption and find a better solution compared with reinforcement learning. Xie et al. [14] represented network structure with a fixed length binary code, and used genetic algorithm to initialize individuals and explore network space through selection, crossover, and mutation. It only used 17 GPUs to train for one day on the same dataset, which is much faster than the reinforcement learning method. The difference of evolutionary computation methods
Fig. 1. Searched normal cell on Cifar-10 which has too many skip connections.

mainly exists in how to choose the initial population, update the population, and generate offspring. Real et al. [17] used the tournament method to select the parent and deleted the worst individual from the population, and the new offspring inherited all the parameters of the parent. Although such inheritance is not strict with inheritance performance, it can also speed up the learning process compared with random initialization. Elsken et al. [18] adopted the sample parent from the multiobjective Pareto frontier to generate better offspring. In terms of initialization methods, these algorithms often encoded convolution, pooling, and other operations directly [19]. In addition, the basic modules used in ResNet and DenseNet networks can well deal with gradient disappearance. Sun et al. [20] used the genetic algorithm to search ResNet and DenseNet blocks to get constructed network with high performance. Yang et al. [21] used the cell which is a continuous search space as a basic module to speed up the search. However, the method based on evolutionary computation still has considerable computational overhead, and it takes a lot of time to evaluate multiple network structures. To further reduce search cost, differentiable architectural search (DARTS) [22] used weight sharing and optimized the process of supernet training and subnet search so that it can search for the neural network architecture quickly. However, DARTS has the phenomenon that the weight-free operations increase in the later stage of the search, such as skip connection and max pool. A large number of skip connections appear in a cell in Fig. 1. This type of cell cannot extract image features well, resulting in network performance collapse.

To search for the network architecture with high performance and further improve search efficiency, this article proposes a partial channel connection based on attention mechanism for differentiable neural architecture search (ADARTS). The channel attention mechanism is used to select important channels and reinforce feature information. In addition, partial channel connections reduce unfair competition between operations and reduce memory occupation, making the search process faster and more stable. We summarize our contributions as follows:

1) The attention mechanism is employed to extract the importance of the channels of the input data in the searching process. Then, the obtained attention weights are multiplied by the original input data to generate new input so that the key features in the input data can be identified and the neural network can use more important information.

2) The channel selection is used to send the channels with higher attention weights into operation space, and the other channels are directly contacted with the output of operation space to improve the search efficiency and stability. Furthermore, it can weaken the unfair competition between candidate operations caused by the parameters of weight-equipped operations which are not trained well in the initial stage.

3) The proposed method searches 0.2 GPU days on CIFAR-10 dataset, and the searched structure achieves 2.46% classification error and 2.9 M parameters. It also achieves 17.06% classification error when transferred to CIFAR-100 dataset for evaluation, which is better than other comparative NAS methods.

The rest of this article is organized as follows. In Section II, we introduce the basic ideas and methods of DARTS. In Section III, we describe the proposed algorithm that uses the channel attention strategy to select some channels for connection. Section IV introduces the experimental plan, datasets, parameters, and accuracy in image classification. In addition, we prove its effectiveness through comparison with other NAS algorithms and ablation studies. Finally, Section V conclude this article.

II. RELATED WORKS

Differentiable neural architecture search makes the search space continuous and uses gradients to alternately optimize network parameters and network structure weights, which greatly reduces the cost of calculation and improves the search speed. Fig. 2 describes the basic process of DARTS. First, the search space is composed of nodes, and there is a candidate operation space between the nodes. DARTS needs to select the connection mode between two nodes. The operation with the highest weight in the candidate space is selected as the connection operation. After all connected directed edges and corresponding operations are selected, the final network structure is determined. However, unfair competition in the operation space is caused because the weight-equipped operations in the initial training stage are not trained well, which leads to the network collapse. There are a large number of skip connections in the network and the performance of the network drops sharply. FairDARTS [23] used sigmoid functions instead of softmax functions to calculate weights to avoid unfair competition between operations. PDARTS [24] used dropout to randomly cut skip connections to mask some skip connections during network searching and reduced the proportion of dropout with the gradual training of other operating parameters. In addition, a progressive search method was introduced, and the number of cells was gradually increased to make it more similar to the final trained network. However, a larger memory occupation was generated when the cells were increased. PDARTS continuously reduced the candidate operations in the training process, but the final structure could be restricted. To increase the memory utilization of DARTS in searching, GDAS [25] only sampled a part of the operation each time for optimization. PCDARTS [26] adopted the method of partial channel
connection to reduce the operation in the operation space, but the random selection of the channel lead to the instability of the network structure. The proposed method in this article selects more important channels by channel attention weight to update the operation weight, making the search more stable and faster.

A. Search Space

In DARTS, some cells are stacked to form the network structure. Cells are divided into reduction cells and normal cells. The same type of cell has the same internal connection mode and share weight. A cell is a directed graph containing $k$ nodes $x_i = \{x_1, x_2, ..., x_k\}$, two input nodes, one output node, and $k-3$ intermediate nodes. In addition, the $j$th intermediate node is connected to the all predecessor nodes, respectively, and the input of the middle node is obtained through its predecessor nodes as follows:

$$x_j = \sum_{i<j} o^{(i,j)}(x_i)$$  \hspace{1cm} (1)

where $x_j$ is the $j$th node, $x_i$ is the $i$th node, and $o(i,j)$ represents the candidate operation between node $j$ and node $i$.

Each directed edge $(i, j)$ has corresponding operation $o(i,j)$ and weight $\alpha_{o(i,j)}^k$, which is calculated by softmax, as shown in (2). The output of the candidate operation is weighed with the corresponding weight $\alpha_{o(i,j)}^k$. The total input $f(x_j)$ is expressed as:

$$\alpha_{o(i,j)}^k = \frac{\exp(\alpha_{o(i,j)}^k)}{\sum_{k' < o} \exp(\alpha_{o(i,j)}^{k'})}$$ \hspace{1cm} (2)

$$f(x_j) = \sum_{k \in o} \alpha_{o(i,j)}^k o^{(i,j)}(x_i)$$ \hspace{1cm} (3)

where $o$ is the candidate operation space, including ‘none’, ‘maxpool\_3\times3’, ‘avgpool\_3\times3’, ‘skip\_connect’, ‘sepconv\_3\times3’, ‘sepconv\_3\times3’, ‘sepconv\_5\times5’, ‘dilconv\_5\times5’, ‘dilconv\_5\times5’, $o^{(i,j)}$ is the candidate operation, and $\alpha_{o(i,j)}^k$ is the weight of candidate operation on each directed edge $(i, j)$.

B. Search Method

There are two groups of parameters in DARTS. In addition to network weight $\omega$ was used to train the network. Besides, architecture parameter $\alpha$ is added to transform the discrete space into a continuous search space and the parameters are trained by gradient descent. The training loss $L_{train}$ is used to optimize the network weight parameter $\omega$, and the valid loss $L_{val}$ is used to optimize the architecture parameter $\alpha$. The two optimization methods are carried out as (4)–(5) and finally the subnet with better performance is searched:

$$\min \alpha \  L_{val}(\omega^*(\alpha), \alpha)$$ \hspace{1cm} (4)

s.t. $\omega^*(\alpha) = \arg\min\omega \  L_{train}(\omega, \alpha)$ \hspace{1cm} (5)

where $L_{val}$ is valid loss, $\omega$ is network weight, $\alpha$ is network architecture weight, and $L_{train}$ is training loss.

After $\omega$ and $\alpha$ parameters are updated in each generation, for each intermediate node $x_j$, two connecting edges are selected according to the largest $\alpha_{o(i,j)}^k$ on the directed edge $(i, j)$ of the node, and the corresponding operation $o_{select}^{(s,j)}$ will be selected. After the connection mode and operation of all intermediate nodes are determined, other directed edges and candidate operations will be removed. The method of selecting operation is as follows:

$$o_{select}^{(s,j)} = \arg\max_{k \in o} \alpha_{o(i,j)}^k$$ \hspace{1cm} (6)

where $o_{select}^{(s,j)}$ is the selected operation between node $i$ and node $j$ and $\alpha_{o(i,j)}^k$ is the corresponding weight of operation.

III. METHODOLOGY

In the search process of DARTS, each operation and its corresponding output need to be stored in the node, which occupies a large amount of memory. Therefore, the researchers have to set a small batchsize, which limits the memory utilization. The proposed partial channel connection method can greatly improve the operation efficiency and reduce the unfair competition between candidate operations. However, the random selection of channels also leads to instability in the search process. Therefore,
this article proposes to use the attention mechanism to select more meaningful channels and strengthen the features of some channels to make the searched network structure more stable. In the following section, we first introduce the framework of our algorithm, and then elaborate the important steps of attention mechanism, channel selection, and partial channel connection in turn.

### A. Overall Framework

The framework of ADARTS is simply described as follows: first, the attention mechanism is added before each operation space to produce inputs, i.e., the features of all channels are extracted through global pooling and used as the inputs of a multilayer perceptron (MLP). Then, channel attention weights \( F_c \) are obtained through the MLP. After that, the obtained weight \( F_c \) is multiplied by the original input to get the new input. Finally, the top \( 1/K \) new channels with the larger weights are marked as 1 and the rest marked as 0, and the channels marked with 1 are sent into the operation space of ADARTS for the following calculation, while the other channels are directly contacted with the output.

### B. Attention Mechanism

Usually, many channels are generated by convolution operations in deep neural networks, but some feature information from some channels has little effect. Channel attention mechanism focuses on distinguishing the importance of channels [27]. In the channel attention mechanism, it is necessary to compress channels to extract feature information. Therefore, max pooling and average pooling are used to obtain spatial features. The pseudocodes of attention mechanism are given in Algorithm 1. For an input image data \( F \) with width of \( H \), width of \( W \), and channels of \( C \) in the network, max pooling and average pooling are used for each channel, respectively, to obtain the feature data \( f_{ap} \) and \( f_{mp} \) with the size of \( 1 \times 1 \times C \). Then, the two feature data are input into a shared MLP. The neurons of the hidden layer are set to \( CK \) to reduce the computational complexity. Through MLP, \( f_{ap} \) and \( f_{mp} \) obtained are added to generate channel attention weight \( F_c \). The calculation process of the channel attention weights is shown in Fig. 3. The map function of channel attention mechanism is given as follows:

\[
F_c = \sigma(MLP(avgpool(F)) + MLP(maxpool(F)))
\]

\[
= \sigma(MLP(f_{ap}) + MLP(f_{mp}))
\]

where \( F \) is input image data, \( f_{ap} \) is average pooled data \( F \), \( f_{mp} \) is max pooled data \( F \), MLP is the multilayer perceptron, \( \omega_1 \) are weights between input layer and hidden layer, \( \omega_2 \) are weights between hidden layer and output layer, \( f'_{ap} \) is the result of \( f_{ap} \) processed by MLP, \( f'_{mp} \) is the result of \( f_{mp} \) processed by MLP, \( F_c \) is the channel attention weight, and \( \sigma \) is a sigmoid function which is \( 1/(1+e^{-x}) \).

Then, new feature maps are generated by multiplying the channel attention weight and the original feature maps, so the importance of input features can be identified. In detail, Fig. 4 describes the calculation process that the data \( F \) with the original input size of \( H \times W \times C \) is multiplied with \( F_c \) of \( 1 \times 1 \times C \) in channel order to obtain \( F' \). The calculation method is expressed as:

\[
F' = F_c * F
\]

where \( F_c \) is channel attention weight, \( F \) is input feature data, \( F' \) is feature data multiplied by channel attention weight.

### C. Channel Selection

The weights of channel attention can reflect the importance of feature channels. The channels with higher attention weight possess more important information. To better carry out partial channel connections, the more important channels need to be selected to make the search more stable and accurate. In this article, channel mask \( M^{(i,j)} \) which is computed according to
Algorithm 1: Attention Mechanism.

**Input:** Feature data \( F \)

**Output:** Result data \( F' \) of channel attention

1: function Attention\( F \)
2: \( f_{ap} \leftarrow \text{Global average pooling of } H \times W \times C \text{ feature data } F \)
3: \( f_{mp} \leftarrow \text{Global max pooling of } H \times W \times C \text{ feature data } F \)
4: \( f'_{ap} \leftarrow 1 \times 1 \times C \text{ feature data } f_{ap} \text{ is sent into MLP for calculation} \)
5: \( f'_{mp} \leftarrow 1 \times 1 \times C \text{ feature data } f_{mp} \text{ is sent into MLP for calculation} \)
6: \( F_{c} \leftarrow \text{Add the } f'_{ap} \text{ and } f'_{mp} \text{ to get channel attention weight} \)
7: \( F' \leftarrow \text{Multiply } F_{c} \text{ with the original feature data } F \) according to the channel order
8: return \( F' \)
9: end function

**Fig. 5.** Process of selecting channels according to the channel attention weight. The top \( 1/K \) channels are selected and marked as 1 and the rest as 0.

(9) is used to represent the selected feature channels and masked feature channels in the operation space \( o^{(i,j)} \). The pseudocodes of feature channel selection are shown in Algorithm 2. In the process, the top \( 1/K \) channels will be selected according to the weight of channel attention and the rest are masked channels. The selected channels will be assigned a value of 1 and transferred to the operation space for calculation. The masked channels will be assigned a value of 0 and will directly skip the operation space and be concatenated with the output. This feature channel process is shown in Fig. 5

\[
M_{(i,j)} = \begin{cases} 
1 & F_k \in 1/K \text{ channels} \\
0 & F_k \notin 1/K \text{ channels} 
\end{cases}
\]  (9)

where \( M_{(i,j)} \) is the channel mask between the \( i \)th node and \( j \)th node and \( F_k \) is the \( k \)th channel attention weight.

D. Partial Channel Connection

To solve the problem that too small batchsize will cause the instability of parameters and structures during the search process, this article proposes partial channel connections to reduce the memory usage in operation space at runtime. On the same batchsize, if the number of partially connected channels is \( 1/K \), the running memory usage will be reduced to the original \( 1/K \). In this way, the batchsize of input data can be greatly increased, which can speed up the operation and make the network search more stable. Besides, DARTS tends to choose weight-free operations. This is because the weight-free operations provide more precise information than the weight-equipped operations when the neural network is not well trained. For example, the skip connection is to transfer data directly to the next node. Because the weight training needs a lot of iterations, weight-free operations will accumulate a lot of advantages before training completely. In addition, there is a competitive relationship between operations, so even in the late training period, the weight-free operations have more advantages, which results in the phenomenon of network structure collapse. The pseudocodes of the partial channel connection are provided in Algorithm 3. In this algorithm, by using partial channel connections, only the selected channels enter the operation space for training, while the rest of the channels are not processed and then contacted with the outputs of the selected channels. The process is described in Fig. 6. The calculation of output of operation space \( o^{(i,j)} \) between node \( j \) and node \( i \) is expressed by (10). \( M_{(i,j)}^{(i,j)} \) represents the selected and masked channels. Even weight-equipped operations are not trained well, the loss of using partial channels is smaller than the method that all channels enter the operation space. It makes the advantage of weight-free operations less obvious. However, because only the partial channels are sent into the operation space, random selection of connection channels will also cause instability of network architecture and parameters. Therefore, channel attention weight \( F_{c} = \{ F_1, F_2, \ldots, F_n \} \) in Section II-B is used when selecting channels. The selection of \( 1/K \) channels with larger weights as the input channel can not only strengthen the data feature and speed up the search efficiency, but also make the weight parameters more stable during training to strengthen the searched network structure and avoid the phenomenon of structural collapse in the later
Fig. 6. Selected channels are marked as 1 and the masked channels are marked as 0 according to the channel attention weight. The selected channels will go through operation space, and a calculation result of each channel is multiplied by the weight $\alpha$. Finally, the output channels are contacted with the remaining channels.

Algorithm 3: Partial Channel Connection.

Input: Feature data through channel attention $F'$, channel mask $M^{(i,j)}$

Output: Hybrid computation $f(x_j)$ of node $j$

1. function Partialconnection$F', M^{(i,j)}$
2. $k \leftarrow 0$
3. while $k < c$
4. if $M_k^{(i,j)} == 1$ then
5. $F'_k$ is sent to operation space to perform hybrid operations.
6. else
7. $F'_k$ skips the operation space
8. end if
9. $k++$
10. end while
11. The feature output through the operation space and the unprocessed feature are contacted to produce $f(x_j)$ according to the original channel order.
12. return $f(x_j)$
13. end function

search

\[ f(x_j) = \sum_{k \in O} \alpha_{o(i,j)}^{k} o^{(i,j)}(x_i \ast M^{(i,j)}) + (1 - M^{(i,j)}) \ast x_i \]

(10)

where $x_i \ast M^{(i,j)}$ are selected channels, $(1 - M^{(i,j)}) \ast x_i$ are masked channels, $f(x_j)$ is the input of node $j$, $\alpha_{o(i,j)}^{k}$ is the weight of candidate operation on each directed edge $(i, j)$, and $o^{(i,j)}$ is the candidate operation between node $j$ and node $i$.

IV. EXPERIMENTS ON CLASSIFICATION TASKS

We applied ADARTS to image classification to test its performance by using common image datasets (CIFAR-10/ CIFAR-100). The network architecture found on CIFAR-10 is transferred to CIFAR-100 for evaluation. In addition, we make ablation experiments to verify the classification accuracy and stability of ADARTS.

A. Dataset

CIFAR-10 and CIFAR-100 have 60 000 color images with $32 \times 32$ resolution. There are ten classes with 6000 images each in CIFAR-10 and 100 classes with 600 images each in CIFAR-100. Each dataset is divided into two parts, i.e., training set and test set. The training set is used to search for network structure, and the test set is used to verify the performance of the network.

B. Implementation Details

1) Network Structure: The network consists of the normal cells and the reduction cells. There are two input nodes, four intermediate nodes, and one output node in each cell. One-third and two-thirds of the network are reduction cells and the rest are normal cells. The operation space between nodes has eight candidate operations, which are 'none', 'maxpool_3 x 3', 'avgpool_3 x 3', 'skip_connect', 'sepconv_3 x 3', 'dilconv_3 x 3', 'sepconv_5 x 5', 'dilconv_5 x 5'.

2) Experimental Methods: The search structure and evaluation structure are shown in Fig. 7(a) and (b). In the searching stage, eight cells are stacked into a network, and some epochs are run on CIFAR-10 to search the cell structures. The searched normal cells and reduction cells are used to generate an evaluation structure by stacking again. Then, the network parameters were trained to test its performance.

3) Parameter Setting: The epoch of network search was set to 80, batchsize was 96, and cross entropy loss was used to calculate $L_{train}$ and $L_{val}$. The SGD optimizer was used to update network parameter $\omega$, the initial learning rate was 0.025, momentum was 0.9, and weight decay was 0.0003. The learning rate was gradually decreased to 0 by using cosine annealing strategy. The Adam optimizer was used to update the network architecture parameter $\alpha$ with an initial learning rate of 0.0006, a momentum of (0.5, 0.999), and a weight decay of 0.001.
C. Experimental Results and Analysis

1) Analysis of Searched Cells: The searched cell structures by ADARTS are shown in Fig. 8. Compared with final architecture in DARTS, the number of skip connections in the network architecture obtained by ADARTS is less and more stable, which can effectively alleviate the problems of gradient disappearance and network degradation. At the same time, more useful feature information can be extracted to enhance the performance of the network.

2) Evaluation on CIFAR-10 and CIFAR-100: Twenty cells are stacked into a new network for performance evaluation and the structure of the evaluation network is shown in Fig. 7(b). The network was trained for 600 epochs with a batchsize of 64. SGD optimizer was used to update network parameters with a momentum of 0.9 and a weight decay of 0.0003. The initial learning rate was 0.017. In addition, the cosine annealing strategy was used to decrease the learning rate to 0. The cutout length of 16, auxiliary weight of 0.4, and drop path probability of 0.3 were applied to prevent overfitting.

We compare the accuracy of ADARTS on CIFAR-10 and CIFAR-100 with some hand-designed networks, RL, evolution, and other gradient-based search methods. The results are given in Table I. We can see from this table that ADARTS achieved 2.46% classification error on CIFAR-10 and 17.06% classification error on CIFAR-100. More important, ADARTS only uses 0.2 GPU days, which is much faster than NASNet-A and AmoebaNet-B, and also faster than other gradient-based search methods. Compared with the baseline DARTS, ADARTS speeds up the search speed while also improving the classification accuracy by 0.3% on CIFAR-10 and 0.48% on CIFAR-100. Compared with GDAS, ADARTS achieves better classification accuracy within the same search time. Compared with PCDARTS, ADARTS can reduce the instability of selecting channels randomly so that it achieves better performance. Also, the parameters of ADARTS are 2.9 million which is less than most other algorithms. This indicates that ADARTS achieves better classification accuracy and faster searching speed.

Moreover, the network structure searched on CIFAR-10 is transferred to CIFAR-100 for testing and performs well, indicating that the searched network architecture have strong robustness.

D. Ablation Studies

1) Analysis of Skip Connection: In DARTS, too many skip connections lead to low network performance, which is caused by unfair competition between operations. The parameters of the weight-equipped operations are not trained well in the early stage of the search and the weight-free operations accumulate too many advantages. There will be more and more skip connections in the network with the increasing of training epochs. The number of skip connections is shown in Fig. 9. In this figure, we can see that four skip connections appeared when DARTS searched to 50 epochs, which greatly reduces the feature information extracted by the network, and results in low performance. When partial connection method is added to DARTS, the number of feature channels in the operation space is reduced, which can weaken the unfair competition in the early stage and reduce the accumulation of advantages of weight-free operations. It can also find that the number of skip connections of DARTS with partial connection is less than that of DARTS. However, the random channel selection also leads to the instability of search, so the channel attention mechanism can select more important channels and strengthen feature information to make the search process more stable. Only in the middle and late stages of the search, the network allows a few skip connections to alleviate the problem of gradient disappearance. All in all, ADARTS can effectively solve the problem of too many skip connections and obtain more stable and better network structures.

2) Analysis of Channel Proportion 1/K: To verify the influence of the K value on memory usage and classification accuracy, and find the best K value, we set different values of K={1,2,4,8,16} for experimental analysis since the initial number of channels for input data is 16. Fig. 10 shows the experimental results with different K values. With the increase of K, the memory usage decreases continuously, while the classification accuracy of the network improves first and then decreases. When K is set to 4, the classification accuracy of the searched network structure is the highest, and the memory usage decreases greatly, which speeds up the search speed. This shows that partial channel connection based on attention mechanism can more effectively search network structure with better performance. However, the number of channels entering the search space should not be too small, otherwise the network cannot get enough feature information during training, and thus cannot get a good network structure.

3) Analysis of Methods of ADARTS: To verify the effectiveness of attention mechanism and partial connection on architecture searching, we use DARTS with random partial connection, DARTS with attention mechanism and ADARTS to search on CIFAR-10, and test the searched network structures...
Fig. 8. Structures of searched cells on CIFAR-10 by ADARTS. (a) Normal cell. (b) Reduction cell.

TABLE I

| Architecture                  | Params(M) | Search Cost(GPU days) | Test Error (CIFAR-10) | Test Error (CIFAR-100) | Search Method |
|-------------------------------|-----------|-----------------------|-----------------------|-------------------------|---------------|
| ResNet(depth=110) [11]        | 1.7       | -                     | 6.43                  | 25.16                   | Manual        |
| ResNet(depth=120) [11]        | 10.2      | -                     | 7.93                  | 27.82                   | Manual        |
| DenseNet-BC [28]              | 25.6      | -                     | 3.46                  | 17.18                   | Manual        |
| VGG [10]                     | 20.1      | -                     | 6.66                  | 28.05                   | Manual        |
| MobileNetV2 [29]              | 2.2       | -                     | 4.26                  | 19.20                   | Manual        |
| Genetic CNN [14]              | -         | 17                    | 7.10                  | 29.05                   | evolution     |
| AmoebaNet-B [30]              | 2.8       | 3150                  | 2.55                  | -                       | evolution     |
| Hierarchical Evolution [31]   | 15.7      | 300                   | 3.75                  | -                       | evolution     |
| CARS [21]                    | 2.4       | 0.4                   | 3.00                  | -                       | evolution     |
| ENAS [32]                    | 4.6       | 0.5                   | 2.89                  | 19.43                   | RL            |
| NASNet-A [13]                | 3.3       | 1800                  | 3.41                  | -                       | RL            |
| NASNet-A + cutout [13]        | 3.3       | 1800                  | 2.65                  | -                       | RL            |
| SMASH [33]                   | 16        | 1.5                   | 4.03                  | -                       | RL            |
| DARTS (first order) + cutout [22] | 3.3   | 1.5                   | 3.00                  | 17.76                   | gradient-based |
| DARTS (second order) + cutout [22] | 3.3   | 4.0                   | 2.76                  | 17.54                   | gradient-based |
| GDAS [25]                    | 3.4       | 0.2                   | 3.87                  | 19.68                   | gradient-based |
| GDAS-cutout [25]             | 3.4       | 0.2                   | 2.93                  | 19.38                   | gradient-based |
| SNAS + cutout [34]           | 2.8       | 1.5                   | 2.85                  | 17.55                   | gradient-based |
| P-DARTS + cutout [24]        | 3.4       | 0.3                   | 2.50                  | 17.20                   | gradient-based |
| PCDARTS + cutout [26]        | 3.6       | 0.1                   | 2.57                  | -                       | gradient-based |
| FairDARTS + cutout [23]      | 2.8       | 0.4                   | 2.54                  | 17.61                   | gradient-based |
| SDARTS-ADV + cutout [35]     | 3.3       | 1.3                   | 2.61                  | 16.73                   | gradient-based |
| EoNAS + cutout [36]          | 3.4       | 0.6                   | 2.50                  | 17.30                   | gradient-based |
| ADARTS                       | 2.9       | 0.2                   | 3.70                  | 18.21                   | gradient-based |
| ADARTS + cutout              | 2.9       | 0.2                   | 2.46                  | 17.06                   | gradient-based |

Fig. 9. Number of skip connections of cells searched on CIFAR-10 by ADARTS. (a) Skip connections in normal cell. (b) Skip connections in reduction cell.
on CIFAR-10 and CIFAR-100. The experimental results are shown in Table II. It is obvious that the search cost is greatly reduced through partial channel connection. In addition, the test classification errors also decrease on CIFAR-10/CIFAR-100 when using attention mechanism or partial channel connection. Moreover, ADARTS which combines attention mechanism and partial channel connection achieves the smallest parameters, fast search speed, and the best classification accuracy among all the methods. It shows that ADARTS can speed up the search process and improve the performance of the searched networks.

4) Analysis of Batchsize: Because the increase of batchsize will lead to the decrease of the step of parameter adjustment in the mini-batch gradient descent, the learning rate should be appropriately increased [37], [38]. Thus, we set several groups of learning rate and batchsize to verify the effect of increasing batchsize appropriately. The experimental results are shown in Table III. When the learning rate is set to 0.01, the accuracy of ADARTS increases as batchsize increases from 8 to 16. Besides, when learning rate is set to 0.025, the accuracy of ADARTS increases as batchsize increases from 32 to 96. Moreover, too small batchsize leads to a significant decrease in stability and accuracy of ADARTS. Therefore, increasing batchsize appropriately can improve the performance of ADARTS.

V. CONCLUSION

In this article, ADARTS was proposed to solve the performance crash due to too many weight-free operations in late search and instability of the search structure caused by low memory utilization. The channel attention and partial channel connection were used to solve problems. The channels with higher weights in the channel attention weight were sent to the operation space. These methods can reduce the consumption of calculation in the operation space to speed up the search speed and improve the stability of searching network structure. Besides, the number of weight-free operations in structure can be controlled so that the searched network can perform well. The algorithm was applied to CIFAR-10 and CIFAR-100 for image classification and the experiment shows that ADARTS can search out the network architecture quickly and stably. In the future work, we will further enhance the stability of channel selection and reduce redundant feature information. We are introducing some typical feature selection techniques to feature map selection problems in deep neural networks. In addition, we will conduct the proposed method on some industrial datasets for network search, and use the obtained network structure in facial recognition, target detection, semantic recognition, etc.

REFERENCES

[1] Y. Sun, J. Xu, G. Lin, W. Ji, and L. Wang, “RFB neural network-based supervisor control for maglev vehicles on an elastic track with network time delay,” IEEE Trans. Ind. Informat., vol. 18, no. 1, pp. 509–519, Jan. 2022.
[2] A. Slowik, “Application of an adaptive differential evolution algorithm with multiple trial vectors to artificial neural network training,” IEEE Trans. Ind. Electron., vol. 58, no. 8, pp. 3160–3167, Aug. 2011.
[3] Y. Deng, T. Zhang, G. Lou, X. Zheng, J. Jin, and Q.-L. Han, “Deep learning-based autonomous driving systems: A survey of attacks and defenses,” IEEE Trans. Ind. Informat., vol. 17, no. 12, pp. 7897–7912, Dec. 2021.
[4] R. C. Luo and C.-J. Chen, “Recursive neural network based semantic navigation of an autonomous mobile robot through understanding human verbal instructions,” in Proc. IEEE/RSJ Int. Conf. Intel. Robot. Syst., 2017, pp. 1519–1524.
[5] R. C. Luo, H.-C. Lin, and Y.-T. Hsu, “CNN based reliable classification of household chores objects for service robotics applications,” in Proc. IEEE 17th Int. Conf. Intell. Informat., 2019, pp. 547–552.
[6] S. M. J. Jalali, S. Ahmadian, A. Khouzam, M. Shahs-hak, S. Nahavandi, and E. P. S. Catalao, “A novel evolutionary-based deep convolutional neural network model for intelligent load forecasting,” IEEE Trans. Ind. Informat., vol. 17, no. 12, pp. 8243–8253, Dec. 2021.
[7] H. Jahangir, H. Taryanani, S. S. Gougheri, M. A. Golkar, A. Ahmadian, and A. Elkamel, “Deep learning-based forecasting approach in smart grids with microclustering and bidirectional LSTM network,” IEEE Trans. Ind. Electron., vol. 68, no. 9, pp. 8298–8309, Sep. 2021.
[8] H. Yu, R. Jin, and S. Yang, “On the linear speedup analysis of communication efficient momentum SGD for distributed non-convex optimization,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 7184–7193.
[9] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2015, arXiv:1412.6980.
[10] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015, arXiv:1409.1556.
[11] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.
[12] X. Liu, J. Zhao, J. Li, B. Cao, and Z. Lv, “Federated neural architecture search for medical data security,” IEEE Trans. Ind. Informat., vol. 18, no. 8, pp. 5628–5636, Aug. 2022.
[13] 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., 2018, pp. 8697–8710.
[14] L. Xie and A. Yuille, “Genetic CNN,” in Proc. IEEE Int. Conf. Comput.Vis., 2017, pp. 1379–1388.
[15] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” 2017, arXiv:1611.01578.

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

[17] E. Real et al., “Large-scale evolution of image classifiers,” in Proc. Int. Conf. Mach. Learn., 2017, pp. 2902–2911.

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

[19] Y. Sun, B. Xue, M. Zhang, and G. Y. G. E. N., “Evolving deep convolutional neural networks for image classification,” IEEE Trans. Evol. Comput., vol. 24, no. 2, pp. 394–407, Apr. 2020.

[20] Y. Sun, B. Xue, M. Zhang, and G. Y. G. E. N., “Completely automated CNN architecture design based on blocks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 4, pp. 1242–1254, Apr. 2020.

[21] Z. Yang et al., “CARS: Continuous evolution for efficient neural architecture search,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 1829–1838.

[22] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable architecture search,” 2018, arXiv:1806.09055.

[23] X. Chu, T. Zhou, B. Zhang, and J. Li, “Fair DARTS: Eliminating unfair advantages in differentiable architecture search,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 465–480.

[24] X. Chen, L. Xie, J. Wu, and Q. Tian, “Progressive differentiable architecture search: Bridging the depth gap between search and evaluation,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2019, pp. 1294–1303.

[25] X. Dong and Y. Yang, “Searching for a robust neural architecture in four GPU hours,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 1761–1770.

[26] Y. Xu et al., “PC-DARTS: Partial channel connections for memory-efficient architecture search,” 2019, arXiv:1907.05737.

[27] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, “CBAM: Convolutional block attention module,” in Proc. Eur. Conf. Comput. Vis., 2018, pp. 3–19.

[28] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 4700–4708.

[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 4510–4520.

[30] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for image classifier architecture search,” in Proc. AAAI Conf. Artif. Intell., 2019, pp. 4780–4789.

[31] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, “Hierarchical representations for efficient architecture search,” 2018, arXiv:1711.00436.

[32] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, “Efficient neural architecture search via parameters sharing,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 4095–4104.

[33] A. Brock, T. Lim, J. Ritchie, and N. Weston, “SMASH: One-shot model architecture search through hypernetworks,” 2018, arXiv:1708.05344.

[34] S. Xie, H. Zheng, C. Liu, and L. Lin, “SNAS: Stochastic neural architecture search,” 2018, arXiv:1812.09926.

[35] X. Chen and C.-J. Hsieh, “Stabilizing differentiable architecture search via perturbation-based regularization,” in Proc. Int. Conf. Mach. Learn., 2020, pp. 1554–1565.

[36] Y. Zhou, X. Xie, and S.-Y. Kung, “Exploiting operation importance for differentiable neural architecture search,” IEEE Trans. Neural Netw. Learn. Syst., to be published, doi: 10.1109/TNNLS.2021.3072950.

[37] P. Goyal et al., “Accurate, large minibatch SGD: Training ImageNet in 1 hour,” 2018, arXiv:1706.02677.

[38] A. Krizhevsky, “One weird trick for parallelizing convolutional neural networks,” 2014, arXiv:1404.5997.