Self-Adaptive Weight Based on Dual-Attention for Differentiable Neural Architecture Search

Yu Xue, *Senior Member, IEEE*, Xiaolong Han, and Zehong Wang

***Abstract*—Differentiable architecture search is a popular gradient-based method for neural architecture search, and has achieved great success in automating design of neural network architectures. However, it still has some limitations such as performance collapse, which seriously affects network architecture performance. To solve this issue, we propose an algorithm called self-adaptive weight based on dual-attention for differentiable neural architecture search (SWD-NAS) in this article. SWD-NAS utilizes a dual-attention mechanism to measure architectural weights. Specifically, an upper-attention module is used to adaptively select channels based on their weights before inputting into the search space. A lower-attention (LA) module is utilized to calculate architectural weights. In addition, we propose an architectural weight normalization to alleviate the unfair competition among connection edges. Finally, we evaluate the architectures searched on CIFAR-10 and CIFAR-100, achieving test errors of 2.51% and 16.13%, respectively. Furthermore, we transfer the architecture searched on CIFAR-10 to ImageNet, achieving top-one and top-five errors of 24.5% and 7.6%, respectively. This demonstrates the superior performance of the proposed algorithm compared to many gradient-based algorithms.**

***Index Terms*—Neural architecture search (NAS), differentiable architecture search (DARTS), channel attention, self-adaptive weight.**

I. Introduction

THE rapid development in deep learning [1] has facilitated notable breakthroughs in domains like computer vision and natural language processing. Nevertheless, numerous successful neural networks are derived via a persistent manual trial and error manner. In addition, different application scenarios often require unique network designs [2], [3], [4]. In order to automate the architecture design process, neural architecture search (NAS) has been proposed and gained considerable attention in recent years [5].

NAS is designed to automate the design of neural network architecture and has surpassed hand-crafted neural networks in many tasks. In some studies, NAS employs reinforcement learning (RL) [6] or evolutionary algorithm (EA) [7], [8] to directly explore the search space and identify the optimal neural network architecture by training thousands of architectures from scratch. Specifically, RL trains an agent to traverse the search space and learn which architecture performs best. In addition, EA has been applied in NAS, which treats each network architecture as an individual and generates superior individuals via crossover and mutation operators. However, these methods require extensive GPU resources. To address this problem, researchers introduced some technologies such as weight-sharing [9], [10] and weight-reusing [11] to enhance NAS efficiency. Although these techniques have gained some effective advancements, the excessive consumption of time and computational resources remains a significant challenge.

There is an urgent need for a more efficient approach to accelerate the entire search process. Liu et al. [12] proposed a novel approach called differentiable architecture search (DARTS) that uses the gradient descent method. Specifically, DARTS introduced a set of architectural parameters to relax categorical architectures and enable the construction of a continuous mixture of candidate operations. Experimental results demonstrated that this approach can significantly reduce search time while achieving high accuracy. Since DARTS introduced architectural parameters, it employed a bilevel optimization method to alternately optimize the network weights and architectural parameters. In this optimization method, nonparametric operations tend to accumulate more advantages than parametric operations during the incomplete training period, leading to a problem known as performance collapse. In other words, DARTS tends to prefer nonparametric operations over parametric ones in the final network. To alleviate performance collapse, Chu et al. [13] replaced the Softmax function with the Sigmoid function to avoid unfair competition between candidate operations. Dong et al. [14] sampled operations to alleviate competition. Unfair competition is one reason for performance collapse, but many studies lack in-depth research on it. Xue et al. [15] used partial channels to avoid the aggregation of too many advantages for nonparametric operations. Liang et al. [16] used an early stopping strategy to prevent nonparametric operations from gaining an advantage in the later stage of the search. However, these

Manuscript received 18 July 2023; revised 23 November 2023; accepted 24 December 2023. Date of publication 9 January 2024; date of current version 4 April 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62376127, Grant 61876089, Grant 61876185, Grant 61902281, and Grant 61403206, and in part by the Natural Science Foundation of Jiangsu Province under Grant BK20141005. Paper no. TII-23-2678. *(Corresponding author: Yu Xue.)*

Yu Xue and Xiaolong Han are with the School of Software, Nanjing University of Information Science and Technology, Nanjing, Jiangsu 210044, China (e-mail: xueyu@nuist.edu.cn; xiaolonghan@nuist.edu.cn).

Zehong Wang is with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: zwang43@nd.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2023.3348843.

Digital Object Identifier 10.1109/TII.2023.3348843

studies have not thoroughly considered the shared architectural parameters and their relation to performance collapse. We have observed that the aggregation of nonparametric operations, particularly skipping connections, can be attributed to the shared architectural parameters. Specifically, cells in deeper positions prefer nonparametric operations to transfer the well-extracted feature maps, while cells in shallower positions prefer parametric operations to extract feature maps of the input data. However, the constraint of shared architectural parameters creates a dilemma, as cells in different positions have conflicting needs. To address this issue, we propose assigning independent architectural weights to different cells based on their respective input data by a channel attention-based approach. Furthermore, during the practical implementation of the attention-based approach, we encountered challenges such as weight confusion and edge confusion. Weight confusion arises from the direct utilization of architectural weights, which consist of channel weights. It means that the operation with the largest architectural weight does not consistently contribute the most across all channels, thus making it challenging to adequately represent the mixed operation. On the other hand, edge confusion arises when attention weights are used to select the connection edge, neglecting the contribution rate of the selected operation on that particular connection edge.

To address performance collapse, we propose an algorithm called self-adaptive weight based on dual-attention for differentiable neural architecture search (SWD-NAS) that can adaptively adjust architectural weights of candidate operations. SWD-NAS replaces the original architectural parameters α used in DARTS with self-adaptive weights based on a dual-attention mechanism. To alleviate weight confusion, our dual-attention mechanism is composed of an upper-attention (UA) module and a lower-attention (LA) module. In the UA, the input of each operation is adaptively selected to prepare for subsequent calculation of the architectural weights. In the LA, the channel weights are calculated and then aggregated to form the architectural weights. To mitigate edge confusion, we propose an architectural weight normalization technique to ensure that the subnetwork topology is the best one to represent the super-network. To summarize, our contributions can be outlined as follows.

1) We introduce a novel approach, referred to as self-adaptive weight based on dual-attention, which solves the problem of the aggregation of nonparametric operations by searching adaptive cells.
2) We utilize a UA module to dynamically select the relevant input channels to alleviate weight confusion. An LA is used to dynamically calculate architectural weights based on the selected channels. Besides, we introduce an architectural weight normalization technique to mitigate edge confusion.
3) We evaluate the effectiveness of our method on CIFAR-10 and CIFAR-100, achieving errors of 2.51% and 16.13%. To evaluate its transferability, we transfer the architecture searched on CIFAR-10 to ImageNet, achieving top-one and top-five errors of 24.5% and 7.6%, respectively. We compare the performance of the network searched by our method with those searched by other gradient-based methods. The experimental results demonstrate that our method is more effective and easily transferable across different datasets.

II. Related Works

Convolutional neural networks are widely used in various fields, including emotion recognition [5] and facial expression recognition [17]. However, designing efficient neural networks often requires extensive expert knowledge. To address this challenge, NAS has been proposed to automate the neural architecture design process. Among many NAS methods, DARTS stands out due to its efficiency in terms of time and resource usage. However, DARTS suffers from unfair competition among candidate operations and performance collapse during the search phases.

A. Methods for Unfair Competition

To address the issue of unfair competition among candidate operations in DARTS, Fair DARTS [13] used a Sigmoid function with zero-one loss to ensure that there is no competition among candidate operations. DARTS-[18] revealed that skip connection has a natural advantage over other candidate operations and should be put into a separate candidate set. GDAS [14] sampled part of operations to alleviate unfair competition among candidate operations. AGNAS [19] replaced the architectural parameters in DARTS, whose experimental results demonstrate that it can search for the operation with higher accuracy. Unfair competition is one reason for performance collapse, while these studies lack in-depth research on performance collapse.

B. Methods for Performance Collapse

To mitigate the problem of performance collapse, many works focused on avoiding overfitting in the search phase. PDARTS [20] identified that there is a large depth gap between the search and retrain phases in DARTS, and proposed an incremental increase in search depth to address this issue. In addition, it used a step-by-step reduction of candidate operations to address the large number of parameters caused by increasing depth. PC-DARTS [21] proposed a method of extracting features in channels and spatial dimensions to reduce the consumption of memory and computational resources. Inspired by PDARTS and PC-DARTS, DARTS+[16] analyzed the root cause of the aggregation of skip connections and attributed it to overfitting during the optimization process. ADARTS [15] replaced random selection with channel selection based on an attention mechanism to enhance the stability of partial channel connection. DARTS-AER [22] proposed an architecture entropy regularizer with a negative or positive coefficient to help optimize architectural parameters. However, these studies have not thoroughly considered the shared architectural parameters and their relation to performance collapse. As a result, we propose

SWD-NAS, a novel approach to address the performance collapse in DARTS from a new perspective and accelerate the search process.

C. Preliminary

The search space of DARTS is defined as a computation cell, which forms the final architecture by stacking itself some times. In this cell, the node represents the feature maps and the connection edge represents the candidate operations. To accelerate the search process, DARTS relaxes the candidate operations to get a mixed operation by

$$O^{(i,j)}(x) = \sum_{l \in n} \frac{\exp(\alpha_l^{(i,j)})}{\sum_{l' \in n} \exp(\alpha_{l'}^{(i,j)})} I_l^{(i,j)}(x) \tag{1}$$

where $O^{(i,j)}(x)$ represents the mixed output, $I^{(i,j)}(x)$ represents the feature map obtained by processing the input through candidate operations, $\alpha^{(i,j)}$ represents architectural parameters of the candidate operations, and l and l' represent the serial number of candidate operations. The primary objective of the search process is to optimize the architectural parameters α. Therefore, DARTS uses a bilevel optimization method to alternately optimize the network weights and architecture parameters as shown in

$$\min_{\alpha} \ \mathcal{L}_{\text{val}}(\omega^*(\alpha), \alpha) \tag{2}$$

$$s.t. \ \ \omega^*(\alpha) = \text{argmin}_{\omega} \ \mathcal{L}_{\text{train}}(\omega, \alpha) \tag{3}$$

where α represents the upper-level variable and ω is seen as the lower-level variable.

Convolutional neural networks focus on local receptive fields and extract informative features by fusing spatial and channel-wise information. Therefore it is necessary to focus on more critical information in an adaptive manner. Hu et al. [23] proposed a novel architectural unit named squeeze-and-excitation (SE) block, which adaptively acquires channel weights of the feature maps. This way helps the convolution process pay more attention to the key channels. Woo et al. [24] proposed an improved attention module, which takes spatial information into consideration and computes channel information more comprehensively. It is worth noting that this embedded attention module introduces an additional set of attention weights, but the attention weights are optimized together with the network weights according to the following equation:

$$\omega, \alpha \ = \arg\min \mathcal{L}_{\text{train}}(\omega, \alpha) \tag{4}$$

where ω represents the network weights and α represents the attention weights.

Inspired by previous attention mechanisms, we accumulate the attention weights of candidate operations to get their respective architectural weights, which enables adaptive adjustment of architectural weights.

III. Methodology

In a deep network, cells at deeper positions tend to favor nonparametric operations to output the well-extracted feature maps, while cells at shallower positions prefer parametric operations to extract information from the input. However, when cells in different positions are forced to share architectural parameters, it creates a conflict in satisfying their respective requirements. Therefore, we propose an algorithm called SWD-NAS to address this issue by replacing architectural parameters with self-adaptive weights based on a dual-attention mechanism. Besides, we propose an architectural weight normalization to avoid the unfair comparison between connection edges. Overall, our proposed approach demonstrates self-adaptivity in the following three aspects: 1) adaptively searching for cells at different depths; 2) adaptively selecting feature maps with a dynamic number of channels; and 3) adaptively calculating architectural weights based on the input feature map.

A. Overall Framework

The framework of SWD-NAS is illustrated in Fig. 1. The top and left subfigures show the cell-based search space, UA represents the upper-attention module, and LA represents the lower-attention module. First, we extract important channels of the input I by UA module and retain the top $\frac{1}{k}$ channels as the new input I' for calculating the architectural weights. Next, I' is processed by candidate operations to get the concatenated feature map $F_{\text{in}}^{\text{con}'}$, which is then used in the LA module to obtain the channel weights $F_{\text{att}}^{\text{con}'}$. Later, the reweighted feature map $F_{\text{out}}^{\text{con}'}$ is obtained by multiplying $F_{\text{in}}^{\text{con}'}$ and $F_{\text{att}}^{\text{con}'}$ in the channel dimension, and the final output F'_{out} is obtained by applying element-wise addition on $F_{\text{out}}^{\text{con}'}$ in channel dimension. It is worth noting that F'_{out} and I^{none} need to be concatenated to ensure subsequent utilization, where I^{none} means the feature maps not selected in UA. At the end, the channel weights of each operation are summed and normalized to get the architectural weight $A^l_{o^{(i,j)}}$, and it will be used as a basis for determining the final selected operation.

B. Self-Adaptive Weight Based on Dual-Attention

In this section, we introduce a dual-attention mechanism that enables the adaptive calculation of architectural weights. To calculate the UA weights, we first apply global average pooling and global max pooling to the input feature map $I \in \mathbb{R}^{b \times c \times h \times w}$, obtaining $I_{ap} \in \mathbb{R}^{b \times c \times 1 \times 1}$ and $I_{mp} \in \mathbb{R}^{b \times c \times 1 \times 1}$, respectively. Specifically, b, c, h, and w represent the number of samples processed in a single batch, the depth of feature maps, the vertical dimension of the feature map, and the horizontal dimension of the feature map. These two feature vectors are then fed into a shared MLP, shown in the following equation:

$$MLP = \mathbf{W}_2(ReLU(\mathbf{W}_1(\cdot) + \mathbf{b}_1)) + \mathbf{b}_2. \tag{5}$$

The MLP is composed of two fully connected layers and a $ReLU$ activation function. Specifically, $\mathbf{W}$ and $\mathbf{b}$ denote the weights and biases of the corresponding fully connected layers, with subscripts indicating different layers within the network. The output of the MLP is added together and then processed by a Sigmoid activation function to get the attention weights I_{mix}.

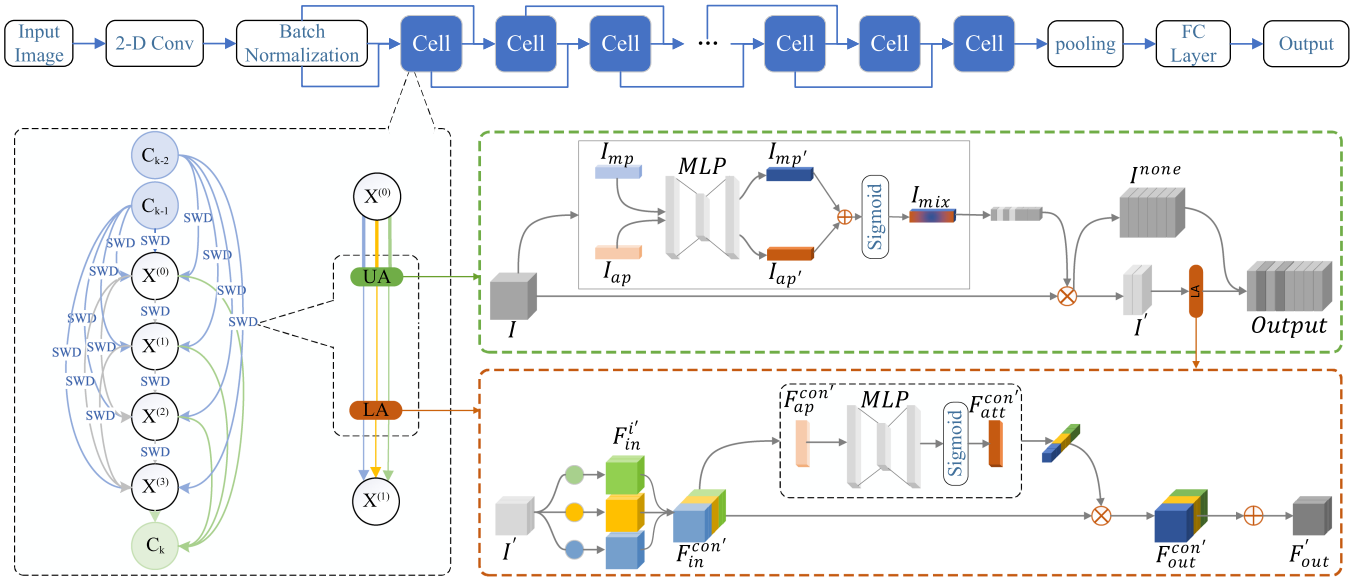

Fig. 1. Overall framework of our proposed SWD-NAS. The top subfigure shows the overall architecture of the network. The left subfigure displays a specific configuration of one cell and we point out the position where we apply SWD. The dual-attention is composed of UA and LA, which adaptively extracts input feature maps and calculates architectural weights.

Algorithm 1: Upper-Attention Module.

Input: Input data $I \in \mathbb{R}^{b\times c\times h\times w}$
Output: Channel attention $I_{mix} \in \mathbb{R}^{b\times c\times 1\times 1}$
$I_{ap} \leftarrow$ Global average pooling of I
$I_{mp} \leftarrow$ Global max pooling of I
$I_{ap'} \leftarrow$ Process I_{ap} through a MLP
$I_{mp'} \leftarrow$ Process I_{mp} through a MLP
$I_{temp} \leftarrow$ Integrate $I_{ap'}$ and $I_{mp'}$ by element-wise addition
$I_{mix} \leftarrow$ Normalize I_{temp} by Sigmoid activation function
return I_{mix}

The process is given as follows:

$$I_{\text{mix}} = \sigma(\text{MLP}(\text{avgpool}(I)) + \text{MLP}(\text{maxpool}(I))) \quad (6)$$

where σ represents the Sigmoid activation function, and the entire calculation process is shown in Algorithm 1.

Adaptively selecting channels will result in a decreasing number of channels on later connection edges, which will significantly affect the subsequent search process. To solve this problem, we use a masking mechanism based on the UA weights. First, we select the top $\frac{1}{k}$ weights of I_{mix} to determine which channels will be sent to the search space. Specifically speaking, we introduce two value arrays, M and N, both initialized with zeros and a length of c, to store masking information. For example, channels belonging to the top $\frac{1}{k}$ are assigned values of 1 in M, while channels beyond the top $\frac{1}{k}$ are assigned values of 1 in N. Finally, we get $I' \in \mathbb{R}^{b\times \frac{c}{k}\times h\times w}$ and $I^{\text{none}} \in \mathbb{R}^{b\times \frac{(k-1)c}{k}\times h\times w}$ according to the masking values in M and N. This masking mechanism guarantees the feasibility of adaptively selecting essential channels for candidate operations and ensures the stable execution of the subsequent architecture search. The entire masking process is shown in Algorithm 2.

Algorithm 2: Channel Extraction.

Input: Input data $I \in \mathbb{R}^{b\times c\times h\times w}$, Channel attention $I_{mix} \in \mathbb{R}^{b\times c\times 1\times 1}$
Output: Selected data $I' \in \mathbb{R}^{b\times \frac{c}{k}\times h\times w}$, Unselected data $I^{none} \in \mathbb{R}^{b\times \frac{(k-1)c}{k}\times h\times w}$
$i \leftarrow 0$
$I_{max} \leftarrow$ top $\frac{1}{k}$ weights in I_{mix} ▷find the top $\frac{1}{k}$ weights
while $i <$ c **do**
  if $I_i \in I_{max}$ **then**
    $M_i \leftarrow 1$
  else
    $N_i \leftarrow 1$
  end if
  $i \leftarrow i + 1$
end while
$I' \leftarrow M \bigotimes I$ ▷determine selected channels via M
$I^{none} \leftarrow N \bigotimes I$ ▷determine unselected channels via N
return I', I^{none}

We use an LA module to obtain the architectural weights based on channel attention weights. The new input I' is processed by candidate operations to get a concatenated feature map $F_{\text{in}}^{\text{con}'} \in \mathbb{R}^{b\times \frac{nc}{k}\times h\times w}$. The concatenated feature map is then processed by global average pooling and an MLP, as shown in (5), to obtain channel weights $F_{\text{att}}^{\text{con}'} \in \mathbb{R}^{b\times \frac{nc}{k}\times 1\times 1}$. The architectural weights are calculated by using

$$\text{Atten}^l = \sum_{x=l-1}^{l} F_{\text{att}}^{\text{con}'}\left(b, x\times\frac{c}{k}, 1, 1\right), l \in [1, n] \quad (7)$$

where Atten^l represents the architectural weight, $F_{\text{att}}^{\text{con}'}$ represents the concatenated channel weights, l represents the serial number of candidate operations, x is the variable of the summation function, $\frac{c}{k}$ represents the adaptively selected channels, and n represents the number of candidate operations. Then, the attention weights $F_{\text{att}}^{\text{con}'}$ need to be multiplied onto the concatenated feature map $F_{\text{in}}^{\text{con}'}$ as shown in

$$F_{\text{out}}^{\text{con}'} = F_{\text{in}}^{\text{con}'} \odot F_{\text{att}}^{\text{con}'} \tag{8}$$

where $F_{\text{out}}^{\text{con}'} \in \mathbb{R}^{b\times\frac{nc}{k}\times h\times w}$ represents the weighted feature map. Finally, the output is obtained by elementwise addition corresponding to each operation as shown in

$$F'_{\text{out}} = F_{\text{out}}^{\text{con}'[0:\frac{c}{k}]} \oplus F_{\text{out}}^{\text{con}'[\frac{c}{k}:\frac{2c}{k}]} \oplus \cdots \oplus F_{\text{out}}^{\text{con}'\left[\frac{(n-1)c}{k}:\frac{nc}{k}\right]} \tag{9}$$

where $F'_{\text{out}} \in \mathbb{R}^{b\times\frac{c}{k}\times h\times w}$ represents the output of mixed operations and is obtained by directly summing the feature maps corresponding to each operation. To ensure that the subsequent search process works properly, we concatenate the unused I^{none} with the obtained F'_{out} to form the input of the next stage. Finally, we select the operation according to the following equation:

$$\text{Atten}^* = \text{argmax}\left(\text{Atten}^l\right), l \in [1, n] \tag{10}$$

where Atten^* represents the serial number of the operation with the largest weight. However, there are many problems with this direct selection of operations based on attention weights, and we will analyze this problem and propose a solution in the next section.

C. Architectural Weight Normalization

In this article, a normalization method is used to address the issue of unfair competition between different edges in the search space of DARTS. The Softmax function used in the original DARTS method assigns weights to different candidate operations on a given edge based on their contribution to the final output. However, when attention weights are used instead, the importance of contribution rates of the operations on a connection edge is neglected, leading to unfair competition between different edges, and we refer to this problem as edge confusion. To address this issue, we propose to normalize the architectural weights before selecting the operation with the highest weight. The specific normalization is shown in

$$A^l_{o^{(i,j)}} = \frac{\text{Atten}^l_{o^{(i,j)}}}{\sum_{l'\in o^{(i,j)}} \text{Atten}^{l'}_{o^{(i,j)}}} \tag{11}$$

where A represents the architectural weight, which is composed of corresponding LA weights. $o^{(i,j)}$ represents connection edges between two nodes, and l and l' represent the serial number of candidate operations on this connection edge. This normalization method takes into account the importance of each connection edge and ensures that the operation with the highest weight is selected based on its true contribution to the final output. The whole process of calculating the architectural weights is shown in Algorithm 3.

Algorithm 3: Calculation of Architectural Weights.

Input: Input data $I' \in \mathbb{R}^{b\times\frac{c}{k}\times h\times w}$
Output: Architectural weights $A^l, l \in [1, n]$
1: $F_{in}^{i'} \leftarrow$ Process I' by candidate operations
2: $F_{in}^{con'} \leftarrow$ Concatenate $F_{in}^{i'}, i' \in [1, n]$
3: $F_{ap}^{con'} \leftarrow$ Global average pooling of $F_{in}^{con'}$
4: $F_{att}^{con'} \leftarrow$ Process $F_{ap}^{con'}$ through a MLP
5: $Atten^l \leftarrow$ Sum $F_{att}^{con'}$ corresponding to the operation, $l \in [1, n]$
6: $A^l \leftarrow$ Normalize the architectural weights $Atten^l$ according to the equation (11), $l \in [1, n]$
7: **return** $A^l, l \in [1, n]$

IV. Experiments

To validate the effectiveness of our proposed method, we conducted the experiments on three well-known datasets: 1) CIFAR-10; 2) CIFAR-100; and 3) ImageNet. Specifically, we performed architecture search on both CIFAR-10 and CIFAR-100, and subsequently transferred the architecture obtained from CIFAR-10 to the ImageNet dataset.

A. Datasets

Both CIFAR-10 and CIFAR-100 consist of 60 000 color images of 32×32 pixels, where 50 000 images are used for training and 10 000 images are used for testing. CIFAR-10 comprises ten classes with each class containing 6000 images. On the other hand, CIFAR-100 comprises 100 classes with each class containing 600 images. To prevent any potential data leakage during the test phase, we divide the training data in the searching phase into two equal parts, one for searching and the other for verification. In addition, to further evaluate the effectiveness of our proposed method, we transfer the architecture obtained from searching on CIFAR-10 to ImageNet. Specifically speaking, we use a subset of ImageNet, which contains 1000 object categories with 224×224 pixels, and we use 1.28 M images for training and 50 k images for validation.

B. Experimental Settings

1) Search Space: The primary objective of our search is to identify cells, including both normal cells and reduction cells, within a directed acyclic graph (DAG). Each cell comprises two input nodes, which are from the previous two cells, four intermediate nodes sorted in a specific order, and an output that aggregates the information from the four intermediate nodes. The edges connecting each pair of nodes x_i and x_j encompass pool O of eight potential operations, including $none, skip\_connect, avg\_pool\_3\times3, max\_pool\_3\times3, sep\_conv\_3\times3, sep\_conv\_5\times5, dil\_conv\_3\times3, dil\_conv\_5\times5$. To mitigate the issue of performance collapse from a novel perspective, we search for cells that can adapt to different positions in the network. In the retraining phase, the cells are extended to form a deeper neural network.

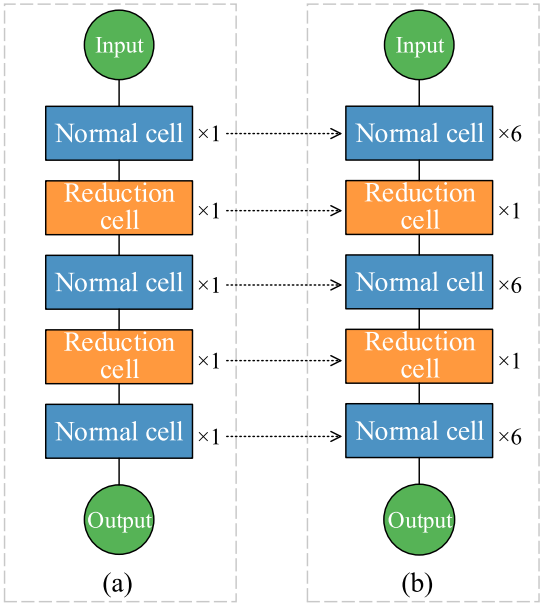

Fig. 2. Description of the relation between the cells in searching phase and the cells in retraining phase. (a) Search phase. (b) Retrain phase.

2) Parameter Settings: At the searching stage, we set the cell depth to 5 and stack three types of normal cells for six times during the retraining stage. This helps us to avoid manual screening when stacking deeper networks. We conducted the search for 30 epochs with a batch size of 64 and used cross-entropy loss to measure $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{val}}$. We used SGD as the optimizer to update network parameters and attention weights, with an initial learning rate of 0.025, a minimal learning rate of 0.001, a momentum of 0.9, and weight decay of 3×10^{-4}. In addition, we employed a cosine annealing strategy to decrease the learning rate to 0.001.

C. Experimental Results

The relationship between cells in the search phase and cells in the retrain phase is described in Fig. 2, and the cells searched by SWD-NAS are described in Fig. 3.

1) Results on CIFAR-10 and CIFAR-100: To validate the reliability of the searched cells, we constructed a network composed of 20 cells and we trained it using an optimizer with a batch size of 96. The network parameters were updated using the stochastic gradient descent (SGD) algorithm with a momentum of 0.9 and a weight decay of 3×10^{-4}. Moreover, we initialized the learning rate to 0.025 and utilized the cosine annealing strategy to decrease it to 0. In addition, we set the cutout length to 16, set the weight for auxiliary loss to 0.4, and set the drop path probability to 0.2. To ensure a fair comparison among algorithms and eliminate any interference caused by different machines, we separately retrained the architecture within the source code of the primary comparison algorithms and our approach using the same graphics card and parameter settings. The experimental results listed in Table I demonstrate that our approach achieved accuracy rates of 97.49% on CIFAR-10 and 83.87% on CIFAR-100, which exceed the results of many other algorithms.

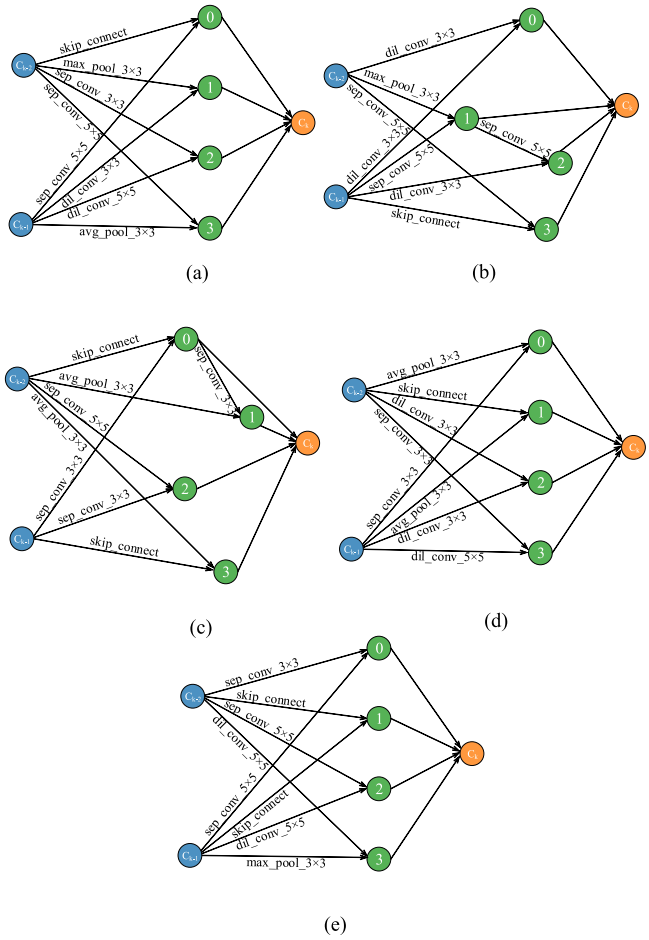

Fig. 3. Cells searched by our proposed method. (a), (c), and (e) represent normal cells, while (b) and (d) represent reduction cells. (a) Cell_0. (b) Cell_1. (c) Cell_2. (d) Cell_3. (e) Cell_4.

2) Results on ImageNet: To further test the effectiveness and transferability of our proposed method, we transferred the architecture searched on CIFAR-10 to ImageNet for retraining. The network was trained for 250 epochs with a batch size of 256 and a depth of 20 layers, with SGD used as the optimizer to update network parameters. The momentum was set to 0.9, and weight decay was set to 3×10^{-5}. The initial learning rate was set to 0.1, and a cosine annealing strategy was employed to decrease the learning rate to 0. In addition, the cutout length was set to 16, the drop path probability was set to 0.2, and the weight for auxiliary loss was set to 0.4. Our experimental results, as shown in Table II, demonstrate that the architecture searched on CIFAR-10 achieved top-one and top-five test accuracies of 75.5% and 92.4% on ImageNet, outperforming many other algorithms. These results indicate that our proposed method can greatly reduce search time while maintaining high transferability.

D. Ablation Study and Results Analysis

1) Analysis of Attention Mechanism: The problem of performance collapse arises from the aggregation of a large number

TABLE I
COMPARISON OF TEST ERROR RATE ON CIFAR-10 AND CIFAR-100

Algorithm	Test error		Params	Search cost	Search method
	CIFAR-10(%)	CIFAR-100(%)	(M)	(GPU days)	
MobileNet-V2 [25]	4.26	19.20	2.2	–	Manual
NASNet-A + cutout [6]	2.65	–	3.3	1800	RL
ENAS + cutout [26]	2.89	–	4.6	1	RL
AmoebaNet-A + cutout [7]	3.34	18.93	3.2	3150	Evolution
AmoebaNet-B + cutout [7]	2.55	–	2.8	3150	Evolution
CARS + cutout [27]	2.86	–	3.0	0.4	Evolution
EPCNAS-C + cutout [28]	3.07	–	1.16	1.1	Evolution
DARTS(first order) + cutout [12]	3.00	17.76	3.3	1.5	Gradient-based
DARTS(second order) + cutout [12]	2.76	17.54	3.3	4.0	Gradient-based
GDAS + cutout [14]	2.93	18.38	3.4	0.21	Gradient-based
DARTS- + cutout [18]	2.50	17.16	3.5	0.4	Gradient-based
PDARTS + cutout [20]	2.50	17.20	3.4	0.3	Gradient-based
PC-DARTS + cutout [21]	2.57	–	3.6	0.1	Gradient-based
Fair DARTS + cutout [13]	2.54	17.61	2.8	0.4	Gradient-based
SmoothDARTS + cutout [29]	2.61	–	3.3	1.3	Gradient-based
AGNAS + cutout [19]	2.60	–	3.6	0.4	Gradient-based
EDD-DARTS + cutout [30]	2.52	–	3.6	0.4	Gradient-based
DARTS-AER + cutout [22]	2.60	–	3.39	4	Gradient-based
MR-DARTS + cutout [31]	2.52	17.48	2.5	0.4	Gradient-based
β-DARTS + cutout [32]	2.53	16.24	3.75	0.4	Gradient-based
DARTS-PT + cutout [33]	2.61	–	3.0	0.8	Gradient-based
RF-DARTS + cutout [34]	2.6	16.5	4.6	–	Gradient-based
SWD-NAS(CIFAR-10) + cutout	2.51	17.08	3.17	0.13	Gradient-based
SWD-NAS(CIFAR-100) + cutout	2.53	16.13	3.56	0.13	Gradient-based

TABLE II
TEST ERRORS OF TOP-ONE AND TOP-FIVE ON IMAGENET

Algorithm	Test error		Params	Search cost
	Top-1(%)	Top-5(%)	(M)	(GPU days)
MobileNet-V2 [25]	28.0	9.0	3.4	-
NASNet-A [6]	26.0	8.4	5.3	1800
NASNet-B [6]	27.2	8.7	5.3	1800
NASNet-C [6]	27.5	9.0	4.9	1800
AmoebaNet-A [7]	25.5	8.0	5.1	3150
AmoebaNet-B [7]	26.0	8.5	5.3	3150
CARS [27]	26.3	8.4	4.4	0.4
DARTS [12]	26.9	9.0	4.9	4.0
GDAS [14]	26.0	8.5	5.3	0.21
GDAS(FRC) [14]	27.5	9.1	4.4	0.17
DARTS- [18]†	23.8	7.0	4.9	4.5
PDARTS [20]	24.7	7.5	5.1	0.3
SETN [35]	26.7	8.6	5.2	1.8
PC-DARTS [21]	25.1	7.8	5.3	0.1
BayesNAS [36]	26.5	8.9	3.9	0.2
EDD-DARTS [30]	24.6	7.4	5.0	0.4
EoiNAS [37]	25.6	8.3	5.0	0.6
MFENAS [38]	26.06	8.18	5.98	0.6
SWD-NAS	24.5	7.6	6.3	0.13

† searched on ImageNet
Unless otherwise specified, all the networks are searched on CIFAR-10 and directly transferred to ImageNet

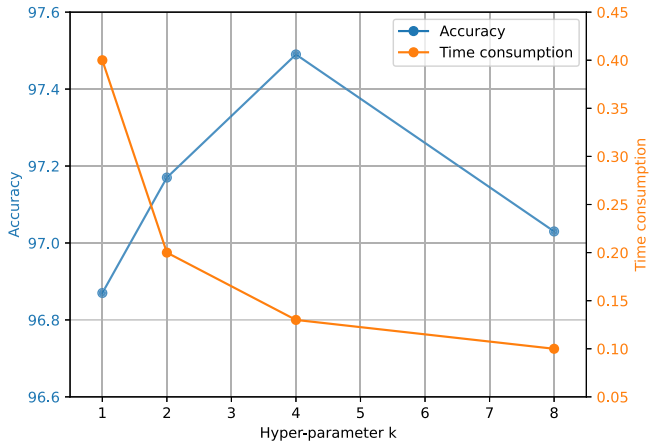

Fig. 4. Classification accuracy and time consumption with different k values.

of nonparametric operations. In this article, we analyzed the problem from a new perspective. As shown in Fig. 3, the cells searched by SWD-NAS have no problem of the aggregation of nonparametric parameters, and the results demonstrate that the searched cells have a better performance. To evaluate the impact of the hyperparameter k in our proposed method, we conducted experiments on CIFAR-10 using different values of k to determine the number of channels for calculating self-adaptive weights. To obtain more rigorous results, we conducted three time experiments for each value of k and selected the best result for each k value to generate Fig. 4. The time consumption represents the runtime with an NVIDIA 3090 graphics card (GPU days). We observed that the selection of k had a notable effect on the search process. For instance, smaller values of k resulted in more search time. However, we noticed that the accuracy of the searched architecture was not consistently correlated with the value of k. To find the most effective value of k, we tested four different values and analyzed the results. After several experiments, we found that setting the value of k to 4 resulted in the optimal balance between search time efficiency and classification accuracy, as depicted in Fig. 4. This choice of k ensures that weight confusion is minimized and the input data retain sufficient information. The results indicate that selecting an appropriate value for k is crucial for balancing search efficiency and accuracy in our method. By setting k to 4, we obtain a favorable balance between search time consumption and the quality of the searched architecture.

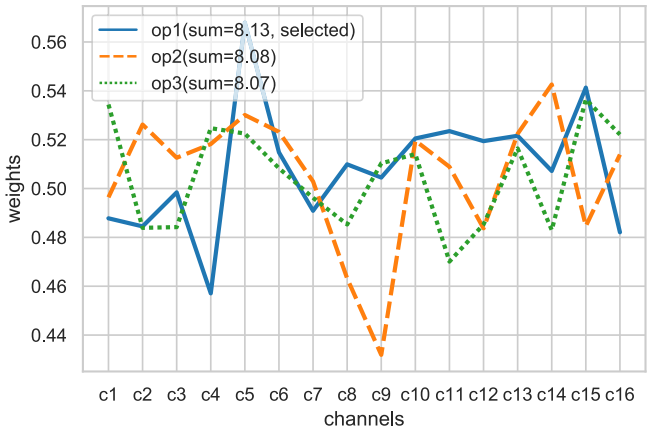

Fig. 5. Example of weight confusion. The *x*-axis represents channels of input and the *y*-axis represents the channel attention weights. The selected solid line does not contribute the most on all channels.

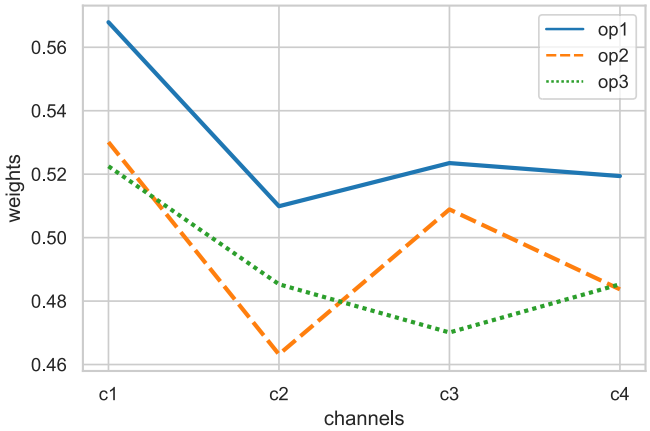

Fig. 6. Internal composition of the architectural weights in our proposed method.

In practice, the LA module can directly contribute to the calculation of architectural weights. However, this method of calculating architectural weights heavily relies on channel attention weights. It may lead to significant interference when the number of channels is high, especially in the search space design of DARTS, the number of channels increases continuously with depth. In such a case, it becomes challenging for the operation with the highest architectural weight to guarantee the most significant contribution across all the channels. As a result, the final selected operation may not accurately represent the entire mixed operations. A specific example is shown in Fig. 5. While our proposed SWD-NAS ensures that the selected operation with the highest architectural weight can contribute the most across all the channels. To further demonstrate the superiority of our proposed method over other similar methods, we visualize the attention weights of a set of operations of other methods and a set of operations of our method separately. Our UA module selects important input information, alleviating the inference of unimportant channel information, which ensures that the selected operation contributes the most across all channels in LA module as shown in Fig. 6. We argue that when the parameter k is set to 4, it not only ensures that the selected operation has the maximum contribution across all used channels but also does not miss much channel information. In addition, there are subtle differences in the utilization of channel weights in different scenarios, which is manifested in whether global max pooling (GMP) or global average pooling (GAP) is used in preprocessing. In general, this is categorized into the use of GAP only and the use of both GAP and GMP. In order to test their roles in the dual-attention mechanism, we performed additional experiments to demonstrate the effectiveness of each module. The specific experimental results are shown in Table III. Based on the results presented in Table III, we observed that the performance was suboptimal when the UA module solely utilized GAP. This can be attributed to the fact that GMP helps preserve locally important information of the feature map during channel selection. However, when we incorporated both GAP and GMP in the calculation of architectural weights, we noticed that the results were not satisfactory. This may be due to the instability of the information obtained by GMP compared with that obtained by GAP, making it challenging to accurately reflect the overall importance of the feature map and GMP will bring more noise.

TABLE III
CLASSIFICATION ACCURACY WITH DIFFERENT PREPROCESSINGS

UA	LA	Accuracy(%)
GAP+GMP	GAP+GMP	96.99
GAP	GAP+GMP	96.67
GAP	GAP	97.15
GAP+GMP	GAP	97.49

2) Analysis of Architectural Weight Normalization: In gradient-based methods, there exists competition between architectural weights due to the utilization of Softmax function, which ensures that the last selected operation will have the greatest weight in the mixed operations. However, existing methods often neglect the contribution rate at the connection edge, and thus suffer from edge confusion, i.e., the highest attention weight does not necessarily mean the highest contribution rate on the edge. Specifically speaking, the final subnetwork topology is composed of several connection edges in the super-network, where the selection of connection edge is determined by architectural weights. But the highest weight value may not reflect the contribution of this operation on the particular edge, as shown in Fig. 7. According to the principle of contribution rate in DARTS, the purity of this edge may not be optimal. In our method, we consider the competition among connection edges and normalize the architectural weights to select operations with higher representation. The ablation study presented in Table IV demonstrates that this approach can further search for networks with better performance.

3) Analysis of Network Depth: In the standard configuration of the DARTS algorithm, it is common to stack 8 cells for architecture search. However, in our experiments, we encountered challenges in achieving convergence during retraining in this setting. To address this issue, we reduced the number of searched cells to 5, which helps the network to be more easily trained to convergence. By reducing the number of cells, we

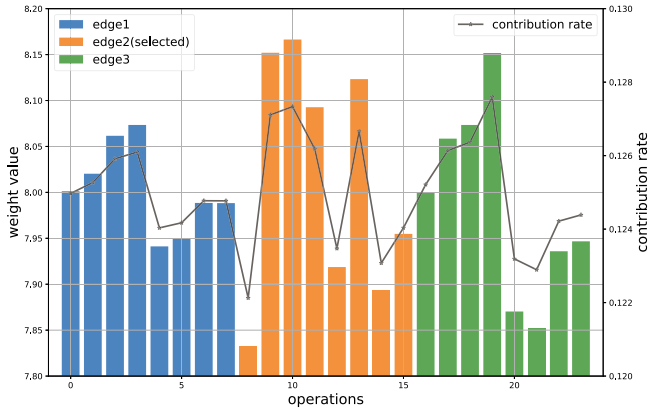

Fig. 7. Example of edge confusion. The *x*-axis represents all the operations on three edges with competitive relation, the left *y*-axis represents the operation weight values, and the right *y*-axis represents the contribution rate of each operation on its edge. The selected operation does not have the highest contribution rate on its edge.

TABLE IV
ABLATION STUDY ON CIFAR-10, WHERE UA REPRESENTS UPPER-ATTENTION MODULE, LA REPRESENTS LOWER-ATTENTION MODULE, AND AWN DENOTES ARCHITECTURAL WEIGHT NORMALIZATION

UA	LA	AWN	Test error (%)	Params (M)	Search cost (GPU days)
✗	✗	✗	2.76	3.3	4.0
✗	✓	✗	2.6	3.6	0.4
✓	✓	✗	2.56	3.3	0.27
✓	✓	✓	2.51	3.1	0.13

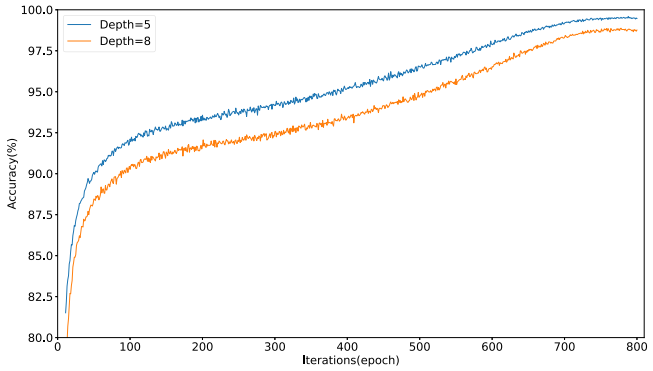

Fig. 8. Retraining accuracy curves of SWD-NAS at search depths of 5 and 8, where depth represents the diversity of cells.

effectively decreased the diversity and complexity within the architecture, as demonstrated in Fig. 8. Networks with higher diversity and complexity can be more challenging to train and may require additional computational resources or fine-tuning strategies to achieve convergence. It is worth noting that the optimal number of cells may vary depending on the specific dataset, problem domain, and experimental setup. In some cases, a lower number of cells can lead to better convergence and improve overall performance during retraining. However, it is essential to carefully evaluate the tradeoff between network complexity and convergence when selecting the appropriate number of cells.

TABLE V
COMPARISON OF INFERENCE TIME

Algorithm	DARTS	SWD-NAS	
		Depth=5	Depth=8
Inference time(ms)	107.823	114.481	167.655

4) Analysis of Inference Time: We have calculated the inference time for both the DARTS and SWD-NAS models during the search phase. The results are presented in Table V. We conducted 50 inferences for each model and reported the mean time, using an initial model channel of 16, 10 output classes, and an input test tensor size of (1, 3, 32, 32). Due to the incorporation of additional attention and channel slicing operations, our model exhibits an inference time approximately 1.55 times that of DARTS when both models use eight cells. However, when our model operates with five cells as explained in Section IV-D3, it achieves better results with an inference time equal to that of DARTS, which is about 1.06 times that of DARTS.

V. CONCLUSION

In this article, we proposed a novel gradient-based NAS method named SWD-NAS, which solved the problem of performance collapse in DARTS from a new perspective by adaptively calculating architectural weights. Our proposed method could search for more diverse cells, and these cells had no aggregation of nonparametric operations. In addition, we compared our method with other methods and achieved competitive results. In future work, we will further enhance the stability and efficiency of our method and introduce more spatial information to increase the importance of architectural weights. Besides, we intend to apply the algorithm to more practical tasks such as object detection and semantic segmentation.

REFERENCES

[1] M. Zhao, S. Zhong, X. Fu, B. Tang, and M. Pecht, "Deep residual shrinkage networks for fault diagnosis," *IEEE Trans. Ind. Informat.*, vol. 16, no. 7, pp. 4681–4690, Jul. 2020.
[2] J. Zhu, H. Shi, B. Song, Y. Tao, and S. Tan, "Convolutional neural network based feature learning for large-scale quality-related process monitoring," *IEEE Trans. Ind. Informat.*, vol. 18, no. 7, pp. 4555–4565, Jul. 2022.
[3] M. Wieczorek, J. Siłka, M. Woźniak, S. Garg, and M. M. Hassan, "Lightweight convolutional neural network model for human face detection in risk situations," *IEEE Trans. Ind. Informat.*, vol. 18, no. 7, pp. 4820–4829, Jul. 2022.
[4] 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.
[5] C. Li, Z. Zhang, X. Zhang, G. Huang, Y. Liu, and X. Chen, "EEG-based emotion recognition via transformer neural architecture search," *IEEE Trans. Ind. Informat.*, vol. 19, no. 4, pp. 6016–6025, Apr. 2023.
[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.*, 2018, pp. 8697–8710.
[7] 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.
[8] Y. Xue, C. Chen, and A. Słowik, "Neural architecture search based on a multi-objective evolutionary algorithm with probability stack," *IEEE Trans. Evol. Comput.*, vol. 27, no. 4, pp. 778–786, Aug. 2023.

[9] A. Brock, T. Lim, J. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through HyperNetworks," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–21.
[10] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. 35th Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 550–559.
[11] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 1–8.
[12] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Representations*, New Orleans, LA, USA, May 6–9, 2019, pp. 1–13.
[13] X. Chu, T. Zhou, B. Zhang, and J. Li, "Fair DARTS: Eliminating unfair advantages in differentiable architecture search," in *Computer Vision*. Berlin, Germany: Springer International Publishing, 2020, pp. 465–480.
[14] 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.
[15] Y. Xue and J. Qin, "Partial connection based on channel attention for differentiable neural architecture search," *IEEE Trans. Ind. Informat.*, vol. 19, no. 5, pp. 6804–6813, May 2023.
[16] H. Liang et al., "DARTS : Improved differentiable architecture search with early stopping," 2020, *arXiv:1909.06035*.
[17] C. Bisogni, A. Castiglione, S. Hossain, F. Narducci, and S. Umer, "Impact of deep learning approaches on facial expression recognition in healthcare industries," *IEEE Trans. Ind. Informat.*, vol. 18, no. 8, pp. 5619–5627, Aug. 2022.
[18] X. Chu, X. Wang, B. Zhang, S. Lu, X. Wei, and J. Yan, "DARTS-: Robustly stepping out of performance collapse without indicators," in *Proc. Int. Conf. Learn. Representations*, 2021, pp. 1–22.
[19] Z. Sun et al., "AGNAS: Attention-guided micro and macro-architecture search," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2022, pp. 20777–20789.
[20] 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.
[21] Y. Xu et al., "Partially-connected neural architecture search for reduced computational redundancy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 2953–2970, Sep. 2021.
[22] K. Jing, L. Chen, and J. Xu, "An architecture entropy regularizer for differentiable neural architecture search," *Neural Netw.*, vol. 158, pp. 111–120, 2023.
[23] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 8, pp. 2011–2023, Aug. 2020.
[24] 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.
[25] 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.
[26] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. 35th Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 4095–4104.
[27] Z. Yang et al., "CARS: Continuous evolution for efficient neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1826–1835.
[28] J. Huang, B. Xue, Y. Sun, M. Zhang, and G. G. Yen, "Particle swarm optimization for compact neural architecture search for image classification," *IEEE Trans. Evol. Comput.*, vol. 27, no. 5, pp. 1298–1312, Oct. 2023.
[29] X. Chen and C.-J. Hsieh, "Stabilizing differentiable architecture search via perturbation-based regularization," in *Proc. 37th Int. Conf. Mach. Learn.*, PMLR, 2020, pp. 1554–1565.
[30] J. Zhang and Z. Ding, "Small temperature is all you need for differentiable architecture search," in *Proc. 27th Pacific-Asia Conf. Knowl. Discov. Data Mining*, Osaka, Japan, Springer, 2023, pp. 303–315.
[31] F. Gao, B. Song, D. Wang, and H. Qin, "MR-DARTS: Restricted connectivity differentiable architecture search in multi-path search space," *Neurocomputing*, vol. 482, pp. 27–39, 2022.
[32] P. Ye, B. Li, Y. Li, T. Chen, J. Fan, and W. Ouyang, "b-DARTS: Beta-decay regularization for differentiable architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10874–10883.
[33] R. Wang, M. Cheng, X. Chen, X. Tang, and C.-J. Hsieh, "Rethinking architecture selection in differentiable NAS," in *Proc. 9th Int. Conf. Learn. Representations*, Austria, May 3–7, 2021, pp. 1–18.
[34] X. Zhang, Y. Li, X. Zhang, Y. Wang, and J. Sun, "Differentiable architecture search with random features," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 16060–16069.
[35] X. Dong and Y. Yang, "One-shot neural architecture search via self-evaluated template network," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 3680–3689.
[36] Z. Zhou et al., "Bayesian differentiable architecture search for efficient domain matching fault diagnosis," *IEEE Trans. Instrum. Meas.*, vol. 70, pp. 1–11, 2021.
[37] Y. Zhou, X. Xie, and S.-Y. Kung, "Exploiting operation importance for differentiable neural architecture search," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 11, pp. 6235–6248, Nov. 2022.
[38] S. Yang, Y. Tian, X. Xiang, S. Peng, and X. Zhang, "Accelerating evolutionary neural architecture search via multifidelity evaluation," *IEEE Trans. Cogn. Devel. Syst.*, vol. 14, no. 4, pp. 1778–1792, Dec. 2022.

Yu Xue (Senior Member, IEEE) received the Ph.D. degree in computer application technology from the School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2013.

He is a Professor with the School of Software, Nanjing University of Information Science and Technology, Nanjing, China. From 2016 to 2017, he was a Visiting Scholar with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. From 2017 to 2018, he was a Research Scholar with the Department of Computer Science and Engineering, Michigan State University, MI, USA. His research interests include deep Learning, evolutionary computation, machine learning, computer vision, and feature map selection.

Xiaolong Han received the B.S. degree in computer science and engineering from Shaoxing University, Shaoxing, China, in 2022. He is currently working toward the master's degree in electronic information with the Nanjing University of Information Science and Technology, Nanjing, China.

His research interests include deep learning, evolutionary computation, and neural architecture search.

Zehong Wang received the B.S. degree in computer science and engineering from Shaoxing University, Shaoxing, China, in 2022, and the master's degree in mathematics from the University of Leeds, Leeds, U.K., in 2023. He is currently working toward the Ph.D. degree in computer application technology with the Department of Computer Science and Engineering, University of Notre Dame, IN, USA.

His research interests include data mining, machine learning, and graph neural networks.