Information Sciences 581 (2021) 448–468

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A neural network architecture optimizer based on DARTS and generative adversarial learning

Ting Zhang [a], Muhammad Waqas [a,b], Hao Shen [a,c], Zhaoying Liu [a,*], Xiangyu Zhang [a], Yujian Li [a], Zahid Halim [b], Sheng Chen [d,e]

[a] Engineering Research Center of Intelligent Perception and Autonomous Control, Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China
[b] Faculty of Computer Science and Engineering, GIK Institute of Engineering Sciences and Technology, Topi 23460, Pakistan
[c] Research Institute of China Telecom Corporation LTD, Beijing 100032, China
[d] School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK
[e] King Abdulaziz University, Jeddah 21589, Saudi Arabia

ARTICLE INFO

Article history:
Received 3 April 2021
Received in revised form 9 September 2021
Accepted 15 September 2021
Available online 17 September 2021

Keywords:
Deep learning
Neural network architecture search
Differentiable architecture search (DARTS)
Pruning
Generative adversarial learning (GAL)

ABSTRACT

Neural network architecture search automatically configures a set of network architectures according to the targeted rules. Thus, it relieves the human-dependent effort and repetitive resources consumption for designing neural network architectures and makes the task of finding the optimum network architecture with better performance much more accessible. Network architecture search methods based on differentiable architecture search (DARTS), however, introduces parameter redundancy. To address this issue, this work presents a novel method for optimizing network architectures that combines DARTS with generative adversarial learning (GAL). We first find the module structures utilizing the DARTS algorithm. Afterwards, the retrieved modules are stacked to derive the initial neural network architecture. Next, the GAL is used to prune some branches of the initial neural network, thereby obtaining the final neural network architecture. The proposed DARTS-GAL method re-optimizes the network architecture searched by DARTS to simplify the network connection and reduce network parameters without compromising network performance. Experimental results on benchmark datasets, i.e., Mixed National Institute of Standards and Technology (MNIST), FashionMNIST, Canadian Institute for Advanced Research10 (CIFAR10), Canadian Institute for Advanced Research100 (CIAFR100), Cats vs Dogs, and voiceprint recognition datasets, indicate that the test accuracies of the DARTS-GAL are higher than those of the DARTS in the majority of the cases. In particular, the proposed solution exhibits an improvement in accuracy by 7.35% on CIFAR10 compared with DARTS, attaining the state-of-the-art result of 99.60%. Additionally, the number of network parameters derived by the DARTS-GAL is significantly lower than that by the DARTS method, with a pruning rate of 62.3% at the highest case.

* Corresponding author.
E-mail address: zhaoying.liu@bjut.edu.cn (Z. Liu).

https://doi.org/10.1016/j.ins.2021.09.041
0020-0255/

1. Introduction

Owing to the development of deep neural network architectures, such as VGG [1], GoogLeNet [2], ResNet [3] and DenseNet [4], the recognition capability of deep learning has been improving continuously in the recent years in various applications, including image processing, speech recognition and natural language processing [5].Professionals developed these novel neural network architectures, which have achieved breakthrough progress through persistent experimentation and attempts based on baseline theories and empirical analysis. This process requires domain expertise and consumes a huge amount of computational resources. The introduction of network architecture search (NAS) [6] has significantly alleviated this problem. An algorithm can now design neural network architecture, and no human expert needs to be involved in the designing process. The performance of networks derived by a NAS algorithm has exceeded the networks designed by experts on certain specific tasks' datasets [7]. Interestingly, NAS algorithms can also design some novel network architectures that have not been proposed by a human expert in the past [8]. For example, Google uses reinforcement learning approaches to devise the neural network architectures [8], and the devised neural networks outperformed the manually designed networks in image classification and natural language processing tasks, thus making the automatic network architecture design the mainstream practice.

The development of NAS has experienced three stages [9]. In the early days, the process of finding the neural network hyperparameters was often referred to as hyperparameter optimization. Common methods included the random search, Bayesian optimization, and evolutionary algorithm [10], to name a few.The conventional network hyperparameters typically had the learning rate, batch size, and regularization coefficient. However, in general, hyperparameters are high-dimensional, including the number of network layers in the neural network, the operator type of each layer, the size of convolutional kernels, the number of convolutional kernels and the convolutional stride. The use of random search, grid search and similar methods yielded low search efficiency. Hence, a more efficient search approach was needed for learning and searching for the network architectures.

Later, a new phase of research on NAS algorithms was launched. During this research period, most of the works focused on evolutionary algorithms [11–13]. In this period, the universal and representative network layers were relatively shallow, the neurons in each layer were small in number, and the overall network architectures were simple chain structures. Under such a limited combination of hyperparameters, optimization with evolutionary algorithms, such as genetic algorithms, was acceptable. However, deep learning has introduced deep neural networks with complex cross-connected and densely connected architectures. As a result, traditional evolutionary algorithms could no longer meet the requirements of searching for deep neural network architectures.

With the application of reinforcement learning methods in this field, NAS had ushered in a phase of blowout development, during which various novel NAS algorithms with excellent performance had been proposed in succession. The majority of these algorithms were inspired by the manual design of architectures and developed based on manual architecture design concepts. The NAS algorithms gained groundbreaking application in 2016, when the research community began to automatically find architectures and look into the in-depth details of network performance. For instance, a Google research team proposed NASNet [14]. Utilizing a recurrent neural network as the controller, the model initially samples and generates a string that describes the network architecture. Then, the architecture is trained, and its performance is evaluated. Finally, the controller parameters are learned by using reinforcement learning [15,16]. Although a neural network architecture with superior performance was obtained with this method, the search time was extremely high. In 2017, Google introduced the evolutionary algorithm to the search problem of neural network architectures [17–20], to shorten the search time and to obtain the more streamlined final network model. In 2018, a differentiable architecture search (DARTS) algorithm was put forward [21]. It innovatively converted the discrete NAS optimization problem into a differentiable process to be solved by the gradient descent method. However, this original DARTS only searches for the best cell through a shallow network, and therefore, the resulting architecture may not be optimal. To this end, Chen et al. proposed the progressive DARTS (P-DARTS) method [22], which deepens the depth of search architectures gradually during the training stage and reduces the computational overhead by approximation and regularization of search spaces. In 2019, in response to the large redundant search space and heavy computational burden with the DARTS, Xu et al. derived the partially-connected DARTS (PC-DARTS) method [23]. This method performs the differentiable architecture search in some channels, substantially reducing the memory chip's memory in the DARTS training.

The main limitation of DARTS is the redundancy in the resulting network architecture. Although splicing of cell architectures searched by DARTS according to the manually set number can yield good network performance, redundant connections and state nodes remain within the cells. Specifically speaking, a cell is composed of multiple modes in DARTS, connected with several kinds of operators like convolution and pooling. These operations are weighted by a few architecture parameters, which are learned in the search scenario. After finding the best cells, we manually stacked the cells according to the NASNet architecture to get a network. For example, DARTS finds the normal and reduction cells on the CIFAR10 dataset, then we stack them manually. Finally, the architecture achieves the test accuracy of 92.35% with 0.369 M parameters on the CIFAR-10 dataset. However, its parameters are less than those of NASNet, while there is remaining redundancy in the resulting network architecture. To address this problem, this work presents a novel network architecture optimization method based on DARTS and generative adversarial learning (GAL) [24], called DRATS-GAL. Our method performs structural pruning on the network architecture searched by DARTS, removing redundant connections and state nodes to re-optimize the orig-

inal DARTS network architecture. Compared to the DARTS method, the proposed DRATS-GAL further optimizes the architecture searched by DARTS, simplifies the network connection, and reduces the network parameters while ensuring the network performance.

The remainder of this paper is arranged as follows. Section 2 introduces the related works. Section 3 details our DARTS-GAL based network architecture optimizer. Section 4 presents the experimental results and performance analysis. Finally, Section 5 concludes this work.

2. Related Works

The related work is categorized into two areas: the network architecture search and the neural network pruning approach.

2.1. Neural network architecture search

There are currently four types of neural network architecture search methods: reinforcement learning-based methods, genetic algorithm-based methods, Bayesian optimization-based methods, and gradient optimization-based methods.

Zoph et al. were the first to propose a reinforcement learning-based neural network architecture search method in 2017 [25], where recurrent neural network parameters were estimated via reinforcement learning. The recurrent neural network is utilized to generate some string sequences that describe the architecture of convolutional neural networks (CNNs) [26]. These string sequences represent network architecture parameters such as the size and stride of the convolutional kernels of each layer and the input/output channel of each layer. Zhong et al. stacked blocks of the same structures into a larger network architecture using a block-based search space and performed optimization by Q-learning-based reinforcement learning algorithm, which achieved preferable results on the CIFAR10 dataset [27].

Genetic algorithm-based methods aim to describe the architecture as a specific genetic string prior to the genetic algorithm optimization and then perform crossover and stochastic mutation on this string during the search process. Each string represents a network architecture. Strings with good evaluation values are obtained by training various network architectures and evaluating these models on the validation set. Such methods require the network architectures under search to be simple. Therefore, they have a small number of parameters, which were used primarily in the early days of neural network architecture search [28]. Currently, the parameters of CNNs used in image classification, detection and segmentation generally exceed one million in number, making it almost impossible to search for network architecture parameters using genetic algorithms [29,30].

Bayesian optimization is often used to optimize the parameters of machine learning algorithm models. For example, Kirthevasan et al. designed NASBOT, a Gaussian model-based Bayesian optimization framework, for searching neural network architectures [31]. Klein et al. developed a Bayesian network to predict the network learning curves, based on which they terminated the evolution of poorly performing network architectures beforehand [32]. Zhou et al. proposed BayesNAS by combining the advantages of Bayesian learning (prevents over-fitting) and One-shot scheme (does not require training substructures from scratch) [33]. Their network can accomplish the search process in less than half a day.

All the above methods are targeted for discrete search spaces. In 2018, Grathwohl et al. used Concrete Relaxation technology to unify the network architecture and parameters under optimization into a continuous space for joint optimization [34]. Later, Liu et al. put forward DARTS, a gradient-based neural network architecture search algorithm. It transforms the originally discrete network architecture search into a differentiable search process, during which the optimization problem is solved by gradient descent. Despite the attainment of the best search results, there are two problems with the DARTS search process. The first is poor network architecture performance resulting from the excessive number of cross-layer connection operations in the cells at the later search stage. The other one is the high memory and computation overheads. Hence, several modified DARTS algorithms emerged, such as P-DARTS, PC-DARTS, DARTS+[35], FairDARTS and NoisyDARTS [36].

To address the phenomenon of jump connection aggregation, DARTS+, FairDARTS and NoisyDARTS have been proposed successively. DARTS + puts forward two early stopping mechanisms, via which the unit architecture search stops before the occurrence of jump connection aggregation to optimize the searched unit architecture. FairDARTS designs a verification loss function to enlarge the representative value difference during each operation, eliminating the unfair advantage of jump connections. Finally, NoisyDARTS offsets the aggregation and performance loss caused by unfair competition by injecting noise into the jump connections, thereby achieving the unit architecture optimization.

To address the memory and computation overheads, P-DARTS and PC-DARTS have been proposed successively. P-DARTS solves the overhead problem by gradually increasing the number of network layers during the search process. However, despite better results on some datasets, this leads to increased computational burden and decreased instability. On the other hand, PC-DARTS operates only on the part of the feature channels during the architecture search, and such operation is directly skipped for the remaining channels. Subsequently, features of these two operations are merged to obtain new features.

The above methods have improved DARTS either by eliminating the aggregation of cross-layer connections or reducing the memory and computation overheads of DARTS. Despite differing points of departure, both approaches aim to reduce

the connections of DARTS search architecture, which enhances the DARTS performance to some extent. However, the above techniques merely eliminate the cross-layer connections of cells, which fail to reduce the connections of searched architecture holistically. As a result, the problems of redundant connections and high memory overhead remain in the obtained architectures.

2.2. Neural network pruning

Depth and width are the two fundamental dimensions of deep neural networks. To improve network classification performance, researchers have continued to increase the network depth and width in their exploration of network design. Deeper networks have better nonlinear expression ability and can learn complex features more simply, while wider networks allow learning richer features at each layer. On the downside, the resulting neural networks have redundant parameters, and such large and complex neural networks extremely easily lead to over-fitting. An effective method for this kind of complex neural network is pruning and compression.

Depending on the meticulous degree, the neural network pruning algorithms can be classified into unstructured pruning and structured pruning. With unstructured pruning, the parameters at every location in the convolution kernels can be pruned, the pruning granularity is a single neuron, and the sparse effect is rather evident [37,38]. However, this pruning approach leads to the highly distinctive architecture of pruned neural network models. The weight matrix becomes sparse and irregular after pruning, relying on a particular algorithm library or hardware platform. A simple strategy is to delete those parameters that contribute the least to the final prediction result. For example, the optimal brain damage proposed by Lecun et al., abandons the concept that the weight size is equal to the contribution degree. The authors proposed a theoretically modified method for contribution degree measurement is designed to express the contribution degrees of parameters by taking their second derivatives using the objective function [39].

Structured pruning refers to the overall trimming of partial computation units in the network, which is highly efficient and does not require other hardware or software platforms. Standard structured pruning methods include layer-level pruning and the channel-level pruning [40–43]. For instance, Luo et al. proposed to trim the convolution kernels and corresponding feature maps hierarchically based on the statistical information calculated by the next layer[40]. Lin et al. identified the importance of convolution kernels by learning a global binary mask and trimmed the redundant kernels by setting the corresponding mask to 0 [41]. Additionally, most of the existing methods perform pruning relying on labels [42,43]. Despite the particular effectiveness of the above-structured pruning methods, problems like low efficiency, lack of slack and reliance on labels remain.

Based on the excellent performance of generative adversarial networks [44,45], Lin et al. developed a pruning method based on generative adversarial networks. Their method defines an objective function with sparse regularization and then uses a soft mask to adjust the network's output to align it with the baseline network. Afterwards, the problem optimization is performed by generative adversarial learning. This method has achieved superior results to the state-of-the-art models on multiple datasets.

In this paper, we prune the architectures searched by DARTS through a generative adversarial learning-based method, which achieves simultaneous pruning of redundant connections within and between cells to reduce the network architecture parameters.

3. DARTS-GAL Based Network Architecture Optimizer

The DARTS algorithm constructs the network by directly splicing the searched cell architectures. Although the splicing process brings beneficial manual influence on the optimal architecture, it also produces redundancy of network parameters. To this end, we propose a network architecture optimizer called DARTS-GAL. Initially, we search for the cell modules by the DARTS method. Then, we splice the searched cell modules to form a complete network. Next, we trim the redundant intermediate state nodes in the network using a GAL based neural network pruning algorithm. Finally, we retrain the newly pruned network. Fig. 1 depicts the flowchart of the proposed method.

3.1. DARTS-Based Search

As a gradient-based NAS algorithm, DARTS searches for two types of cell architectures: normal cell and reduction cell [21]. Normal cells refer to the cell computing units that do not alter the size of the input feature maps. In contrast, reduction cells refer to the cell computing units that reduce the length and width of the input feature maps to half of the original ones. Both types of cells are the directed acyclic graphs containing N ordered nodes, where N can be set manually. The edges in the cell architecture diagram represent the network computing operations defined in the search space, such as 3×3 convolution, 2×2 maximum pooling and cross-connection. Here the nodes refer to the feature maps obtained after operator operations in deep learning.

The goal of DARTS is to find a structural parameter α that minimizes the loss function on the validation set. For each learned α, training an optimal network weight is necessary. Let the loss functions on the training and validation sets be L_{train} and L_{val}, respectively. Then the optimization goal can be expressed as:

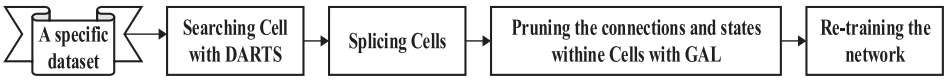


Fig. 1. Flowchart of DARTS-GAL.

$$\begin{aligned} &\min_{\alpha} L_{\text{val}}(w^*(\alpha), \alpha), \\ &\text{s.t. } w^*(\alpha) = \arg\min_{w} L_{\text{train}}(w, \alpha). \end{aligned} \tag{1}$$

The essence of the DARTS algorithm's learning process is to connect the nth node in the cell with all the $n-1$ nodes before it. There are multiple candidate operators between two connected nodes, and each operator is assigned a corresponding structural parameter for participating in the network learning and training together. The operator with the highest final structural parameter value is retrained. Let x^i denote the ith intermediate state node in the cell, and $o^{(i,j)}$ represent the candidate operator between the ith and the jth node, where $j < i$. Then the value of the ith node can be expressed as:

$$x^i = \sum_{j<i} o^{(i,j)}(x^j). \tag{2}$$

A structural weight is assigned for each candidate operator between every two nodes during training to determine the internal cell architectures. For instance, for the operator o between the ith and the jth nodes, its structural parameter is set to $\alpha_o^{(i,j)}$. During training, the feature value of each intermediate state node is a sum of the results of all the input features from the candidate operations. The feature value of the intermediate state node in the search process can be expressed as:

$$\overline{O}^{(i,j)}(x) = \sum_{o\in\mathcal{O}} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o\in\mathcal{O}} \exp\left(\alpha_o^{(i,j)}\right)} o(x). \tag{3}$$

where $\mathcal{O}$ denotes the set of all the operators, such as convolution, pooling, and padding.

In this way, the DARTS algorithm relaxes the discrete problem of selecting the optimal operator into a persistent problem, which can then be solved by the gradient descent method.

3.2. Cell Splicing

Regarding the Inception network construction method [2], DARTS stacks the normal and reduction cells alternately to obtain the entire network architecture. Expressly, at the locations of one-third and two-thirds of the network depth, the cells are set to reduction cells, and at the remaining locations, the cells are set to normal cells. Each cell has two inputs: the output of the adjacent previous cell and the cell's output just before the last cell. Each cell has an output value, the feature value of the feature maps for the intermediate state nodes spliced in the channel direction. It resembles the splicing method of cell output values in Inception. In Fig. 2, the specific connection modes between various cells in the DARTS network are illustrated.

3.3. GAL-Based Neural Network Pruning

The GAL-based algorithm for neural network pruning performs structural pruning of neural network by borrowing the idea of generative adversarial network [46]. It trims the convolutional kernels, branch structures and block structures in the neural network. Fig. 3 presents the flowchart of this GAL-based pruning algorithm.

As illustrated in Fig. 3, the algorithm initially sets the network searched by the DARTS as a baseline network and marks the Softmax value output by the baseline network as a true label. Then, it adds the sparse soft masks to the feature values of the intermediate state nodes for each cell module in the DARTS network, thereby forming the network to be pruned. This network is set as a generator G, and the Softmax prediction value output by G is marked as a false label. Next, like generative adversarial learning [24], it utilizes a 5-layer perceptron network as a discriminator D to perform dichotomous learning on the outputs of the baseline network and of the generator. In the network, the training of the generator and the discriminator is in a state of the adversarial game. During the adversarial learning, part of the soft masks in the generator are sparsified to 0 to reach the extent of being trimmed. Lastly, the structural pruning of the DARTS network is accomplished on the premise of low precision loss. The network optimization strategy consists mainly of two alternate phases:

1. Fixing the generator and soft mask parameter values, and updating the discriminator parameters;
2. Fixing the discriminator and updating the parameters of the generator and soft masks.

Finally, after the sparse training of soft masks, each cell's structured pruning of the intermediate state nodes can be completed according to the final soft mask values.

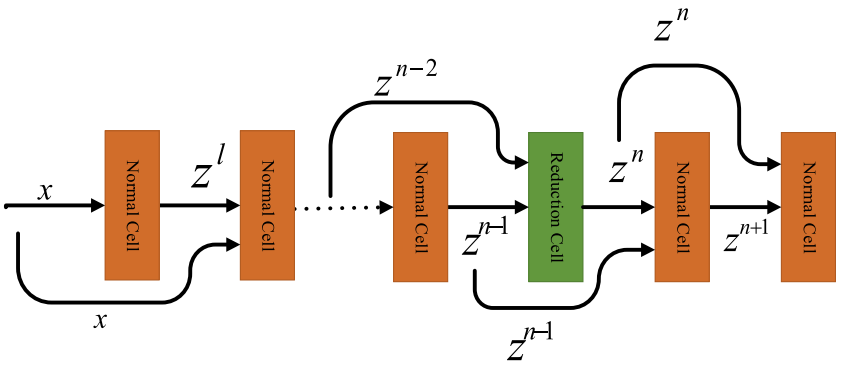


Fig. 2. A model spliced by cells.

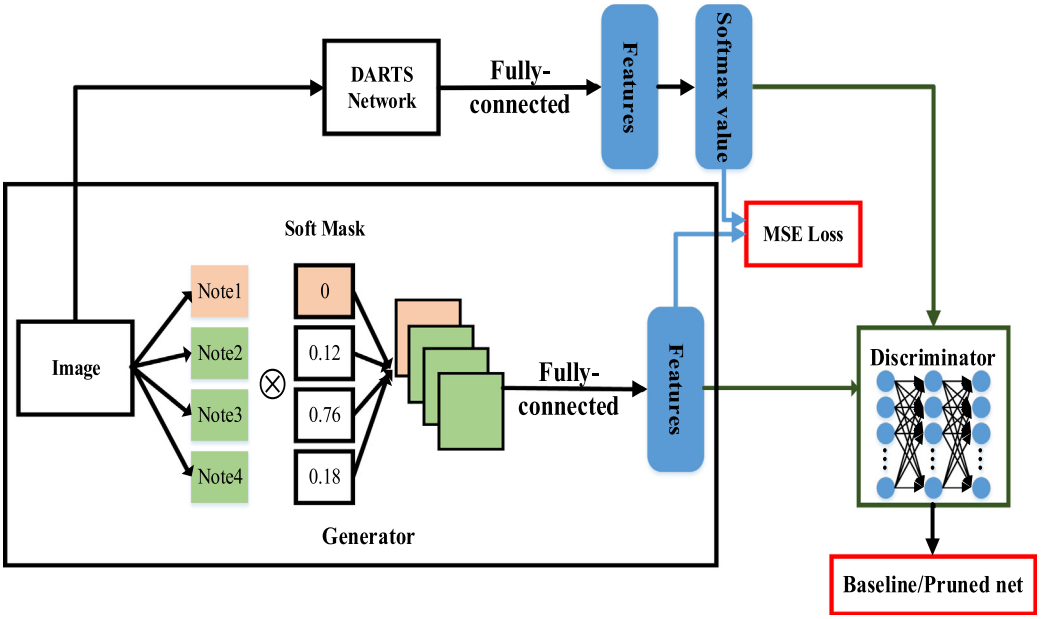


Fig. 3. Flowchart of GAL-based pruning algorithm.

During adversarial learning, the parameters of the baseline network are fixed. By contrast, the network parameters (generator) to be pruned W_G, the discriminator parameters W_D and the soft mask values s need to be adjusted and updated continuously during learning. Therefore, the overall optimization objective function can be expressed as:

$$L = \arg\min_{W_G,s}\max_{W_D}\left(L_{\text{Adv}}(W_G,s,W_D) + L_{\text{data}}(W_G,s) + L_{\text{reg}}(W_G,s,W_D)\right). \tag{4}$$

where $L_{\text{Adv}}(W_G,s,W_D)$ denotes the adversarial loss, $L_{\text{data}}(W_G,s)$ denotes the data loss between the features of the baseline network and the network to be pruned, and $L_{\text{reg}}(W_G,s,W_D)$ represents the regularization of W_G, s and W_D. These three losses can be expressed respectively as:

$$L_{\text{Adv}}(W_G,s,W_D) = E_{f_b(x)\sim p_b(x)}[\log(D(f_b(x),W_D))] + E_{f_g(x,z)\sim\left(p_g(x),p_z(z)\right)}\left[\log\left(1-D(f_g(x,z),W_D)\right)\right], \tag{5}$$

$$L_{\text{data}}(W_G,s) = \frac{1}{2n}\sum_x \|f_b(x) - G(x,W_G,s)\|_2^2, \tag{6}$$

$$L_{\text{reg}}(W_G,s,W_D) = R(W_G) + R_\lambda(s) + R(W_D). \tag{7}$$

Here $p_b(x)$ and $p_g(x)$ denote the feature distributions of the baseline network and the network to be pruned, respectively, $p_z(z)$ denotes the prior distribution of the noise input z, and n is the size of mini-blocks, while $f_b(x)$ is the feature used to train the pruned network, $G(x,W_G,s)$ denotes the pruned (Generator) network, and $f_g(x,z)$ denotes the feature learned from the

pruned network. The noise input z is used as the dropout and is active only when we are updating the pruned network. In addition, $D(f_b(x), W_D)$ denotes the output of the discriminator with the input $f_b(x)$, and $E_{f_g(x,z)\sim(p_g(x),p_z(z))}[\bullet]$ denotes the expectation with respect to $f_g(x,z)$ following the joint distribution of $(p_g(x), p_z(z))$. Furthermore, $R(W_G) = \frac{1}{2}\|W_G\|_2^2$ is the l_2 regularizer of the network to be pruned, $R_\lambda(s) = \lambda\|s\|_1$ is the sparse regularizer of s with regularization parameter λ, and $R(W_D) = E_{f_g(x)\sim p_g(x)}[\log(D(f_g(x), W_D))]$ represents the discriminator regularizer to prevent the discriminator from dominating the training.

3.4. Training the Pruned Network

After pruning optimization of the network architecture, we obtain a new network with fewer states and fewer connections. Therefore, compared with the network architecture before pruning, the model after pruning is different. Then, we load this new model to train with the original dataset while keeping the hyper-parameters, such as learning rate, the batch size, dropout, unchanged to compare with the network before pruning.

It should be noted that GAN is unstable during training and easy to fall into mode collapse. Here, we utilize three methods to solve this problem. Firstly, we use the Wasserstein distance to train the GAN. Secondly, we let the training epochs of the discriminator be more than that of the generator. That is, we train the discriminator five times, then train the generator once. Thirdly, we use the soft mask and add a regularizer on the generator and discriminator, respectively.

4. Experimental Results

To validate the performance of our proposed method, performance comparisons are made first between our DARTS-GAL and other techniques on two tasks of small-size image classification. Afterwards, a similar comparison is made on a task of large-size image classification. Finally, the performance of our DARTS-GAL is compared with other methods on the voiceprint recognition task. All the experiments are run on a deep learning server with Nvidia Tesla K40c GPUs and the 64-bit Windows7 operating system. All the algorithms are implemented in the deep learning framework PyTorch[1], and training occupies one GPU only.

As for training, the parameters of the fully-supervised networks like LeNet[47], VGG16 [1] and ResNet18 [3] were directly taken from their original papers.

For DARTS, while searching the architecture, the dropout is 0.3. As for pruning, we did not use dropout. During all the training procedures, The learning rate is 0.01, and the batch size is 128.

For P-DARTS, during all the training procedures, the learning rate is 0.025, the dropout is 0.3. The batch size is 64 except for the Cats vs Dogs dataset and is 4 for the Cats vs Dogs dataset.

For PC-DARTS, while searching the architecture, the learning rate is 0.1, while for pruning, the learning rate is 0.025. During all the training procedures, the dropout is 0.3. The batch size is 64 except for the Cats vs Dogs dataset and is 4 for the Cats vs Dogs dataset.

For DARTS-GAL, while searching the architecture, the batch size is 32. As for pruning, the batch size is 100. Therefore, the learning rate is 0.01 during all the training procedures.

4.1. Small-Size Grayscale Image Classification Experiment

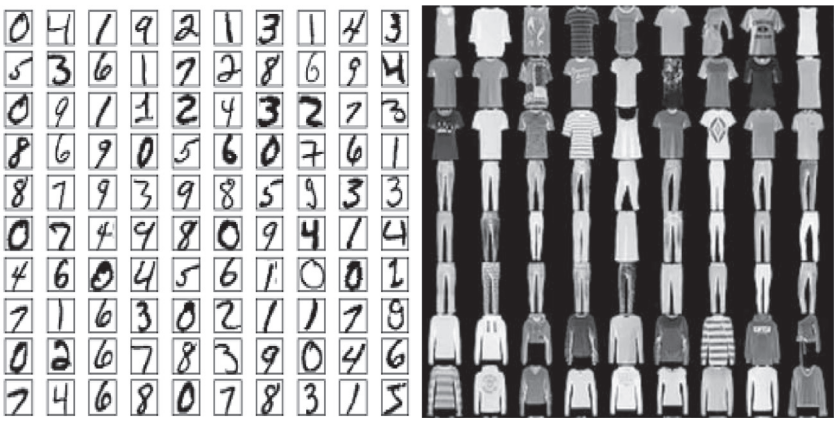
This subsection compares the classification performance of our DARTS-GAL with LeNet [47], DARTS [21], P-DARTS [22] and PC-DARTS [23] for the small-size grayscale images. The datasets adopted are MNIST [47] and FashionMNIST [48].

MNIST is a handwritten digit dataset [47], which comprises 10 categories of digits from 0–9, including 60,000 training images and 10,000 test images. All the images are 28×28 grayscale images, and Fig. 4(a) displays some example images. FashionMNIST is a product image dataset containing ten categories of products [48], and Fig. 4(b) depicts some example images of this dataset. The size, format of images and the partitioning of training and test sets in FashionMNIST are identical to the MNIST.

LeNet is a human-designed complete convolutional neural network for handwritten digit recognition [47]. We train this LeNet with the training sets of MNIST and FashionMNIST and evaluate the corresponding test performance. The training parameters of LeNet is the same as its original reference[47].

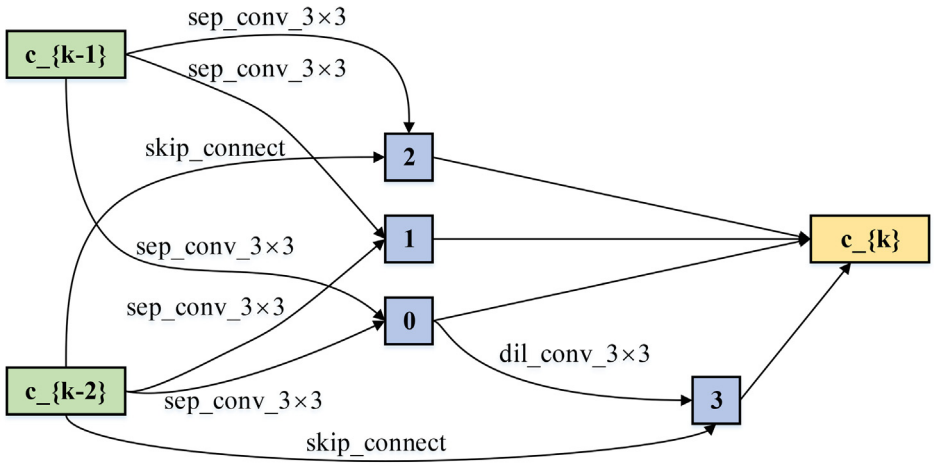
For the DARTS, P-DARTS and PC-DARTS, we do not use the training sets of MNIST and FashionMNIST to search for the network architecture. Rather, we use the network architecture searched by them on the CIFAR10 dataset [49], as described in their original papers. The reason is that these two datasets may be too 'simple', and a NAS may tend to result in shallow network architectures [6]. With the architecture found by them on the CIFAR10 dataset, we then use the training sets of MNIST and FashionMNIST to train the parameters of this network. The test performance is then evaluated on the test datasets of MNIST and FashionMNIST. In Fig. 5, the two cell architectures searched on CIFAR10 in the original DARTS paper are displayed, where the number of intermediate state nodes in each cell is 4. After determining the cell architectures, we first

[1] Facebook, Pytorch. [Online]. Available:https://pytorch.org/.

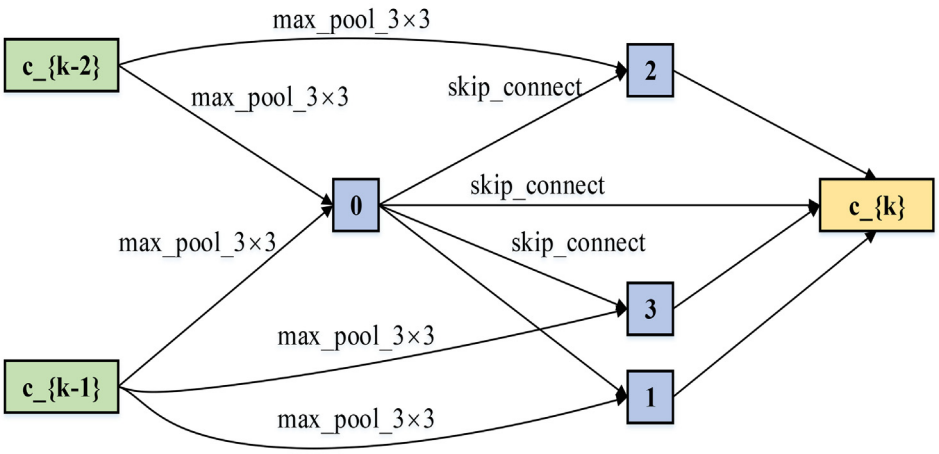


(a) MNIST (b) FashionMNIST

Fig. 4. Examples of MNIST series datasets.



(a) Normal Cell

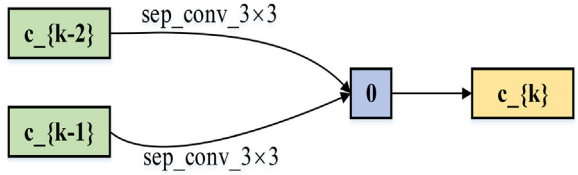


(b) Reduction Cell

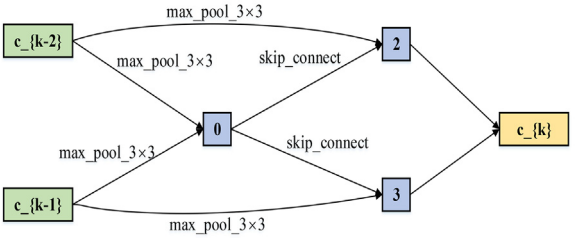
Fig. 5. Cell structure search on CIFAR10 by DARTS.

Table 1
Network pruning by DARTS-GAL on MNIST and FashionMNIST.

Datasets	Cell	Type	Reserved intermediate nodes
MNIST	1	Normal	0
	2	Reduction	2, 3
	3	Reduction	1
	4	Normal	2
	5	Normal	1, 2
FashionMNIST	1	Normal	1, 2
	2	Reduction	0, 1
	3	Reduction	3
	4	Normal	0, 1, 2
	5	Normal	1

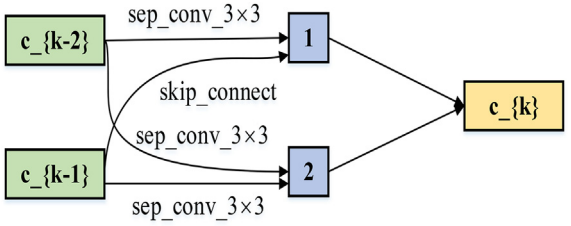


(a) First cell after pruning

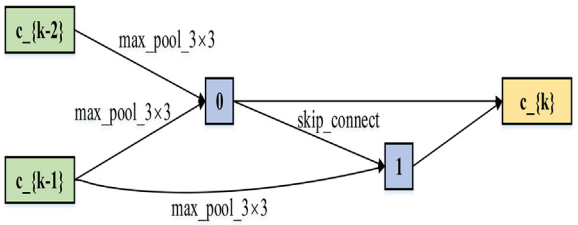


(b) Second cell after pruning

Fig. 6. Example cells for MNIST after pruning.



(a) The first cell after pruning



(b) The second cell after pruning

Fig. 7. Example cells of FashionMNIST after pruning.

Table 2
Performance comparison of different networks on MNIST and FashionMNIST.

Datasets	Models	Test Accuracies(%)	Param(M)
MNIST	LeNet	**99.60**	1.20
	DARTS	99.41	0.13
	P-DARTS	99.15	0.43
	PC-DARTS	99.29	0.74
	DARTS-GAL	99.32	**0.09**
FashionMNIST	LeNet	91.32	1.20
	DARTS	92.33	0.13
	P-DARTS	**93.47**	0.85
	PC-DARTS	92.42	0.31
	DARTS-GAL	92.50	**0.10**

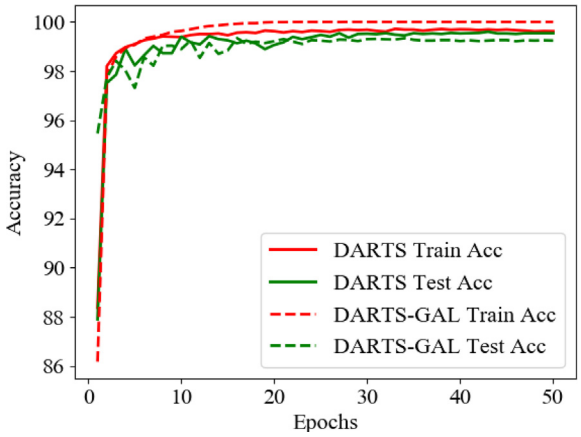


Fig. 8. Training process of DARTS and DARTS-GAL on MNIST.

splice 5 cell architectures as the baseline network of both datasets. The spliced network's first, fourth and fifth cells are normal, while the second and third cells are reduction cells.

For our DARTS-GAL, given the baseline network provided by the DARTS, pruning is performed using the GAL algorithm. Table 1 lists the serial numbers of the intermediate state nodes retained by each of the five cells shown in Fig. 5 on the MNIST and FashionMNIST datasets after pruning. Figs. 6 and 7 show the example cells after pruning the two datasets. The pruned network is then retrained, and the test performance is evaluated.

Table 2 summarizes the test accuracies of LeNet, DARTS, P-DARTS, PC-DARTS and DARTS-GAL on MNIST and FashionMNIST datasets, where Param(M) stands for the number of parameters (millions). Figs. 8 and 9 depict the accuracies

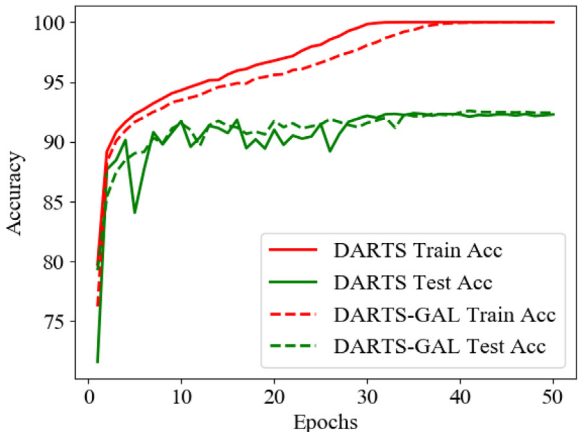


Fig. 9. Training process of DARTS and DARTS-GAL on FashionMNIST.

of DARTS and DARTS-GAL during training on MNIST and FashionMNIST datasets, respectively. The training rounds for these two datasets are both 50, and the first cell's initial channel numbers are set to 16. From the results shown in Tables 1,2 and Figs. 6–9, we can draw the following observations.

1. Not surprisingly, LeNet attains the best test accuracy of 99.60% on the MNIST dataset, as it is a human-designed network specifically catering for handwritten digit recognition. It is, however, a fully convolutional neural network with a huge number of parameters (1.2 million).
2. DARTS, P-DARTS and PC-DARTS are capable of arriving at much sparser network architectures with only 0.13, 0.43 and 0.74 million of parameters, which is less than 8.1%, 6.4% and 3.8% of LeNet on the MINST dataset, respectively. Our DARTS-GAL further removes the redundancy in the DARTS network architecture. For the MNIST dataset, DARTS-GAL achieves a network with only 0.09 million parameters, 0.04 million less than DARTS, 0.34 million less than P-DARTS, and 0.65 million less than PC-DARTS. In contrast, for the FashionMNIST dataset, DARTS-GAL arrives at a network with only 0.10 million parameters, which is 0.03 million less than DARTS, 0.75 million less than P-DARTS, and 0.21 million less than PC-DARTS.
3. For the MNIST dataset, our DARTS-GAL attains a test accuracy of 99.32%, which is 0.28% and 0.09% lower than LeNet and DARTS results; while is 0.17% and 0.03% higher than that of P-DARTS and PC-DARTS, respectively. For the FashionMNIST dataset, our DARTS-GAL attains a test accuracy of 92.50%, which is 1.18%, 0.17% and 0.08% higher than the results of LeNet, DARTS and PC-DARTS, respectively; while is 0.97% higher than that of P-DARTS.
4. DARTS-GAL converges slower than the DARTS during training, as can be seen from Figs. 8 and 9. This is because training the pruned network is equivalent to training a new network from scratch, and accordingly, more training epochs are required.

Combining 2. and 3, we can conclude that the proposed DARTS-GAL can simplify the network architecture searched by DARTS and reduce the network parameters while maintaining the network performance.

4.2. Small-Size Color Image Classification Experiment

This subsection compares the performances of our DARTS-GAL with DARTS, P-DARTS, as well as PC-DARTS on the datasets of CIFAR10 [49] and CIFAR100 [49].

CIFAR10 is a 10-class dataset including aircraft, horses and dogs, with 6,000 images per class. The dataset has a total of 50,000 training images and 10,000 test images. CIFAR100 contains images in 100 classes, with 600 images per class, including 500 training images and 100 test images.

We utilize the readily partitioned training and test sets for these two datasets to perform experiments. For both datasets, the initial channel number of the first cell is 16. For the CIFAR10 dataset, we splice 12 cells of Fig. 5 given by DARTS to form a complete network, of which the fifth and ninth cells are reduction cells, and the rests are normal cells. For the CIFAR100 dataset, we spliced 10 cells of Fig. 10 given by DARTS to form a complete network, of which the fourth and seventh cells are reduction cells, and the rests are normal cells. Thus, the number of intermediate state nodes in each cell is 4.

In Figs. 11 and 12, the cells structures are presented after pruning by DARTS-GAL on these two datasets. In Figs. 13 and 14, DARTS and DARTS-GAL training processes are displayed for the two datasets. Table 3 details the cell pruning results by GAL on the two datasets, whereas Table 4 details the test results by DARTS and DARTS-GAL on them. From the experimental results, we can draw the following observations.

1. Although DARTS converges slower than DARTS-GAL during the training for CIFAR10, which may be because its network has more parameters, it converges significantly faster than the latter for CIFAR100. This again indicates that DARTS-GAL generally converges slower than DARTS, requiring training the pruned network from scratch.
2. In terms of test accuracy, DARTS-GAL attains a 99.60% test accuracy on the CIFAR10 dataset, which is 7.35%, 6.11% and 6.49% higher than that of DARTS, P-DARTS and PC-DARTS, respectively. The CIFAR100 dataset, DARTS-GAL, also outperforms DARTS and PC-DARTS, achieving a 74.47% test accuracy, which is 1.15% and 10.06% higher than that of DARTS and PC-DARTS, while is 0.56% lower than that of P-DARTS. This suggests that pruning the architectures searched by DARTS simplifies the network architectures and improves the accuracy of network classification.
3. Regarding the number of network parameters, the network searched by DARTS has 0.37 M parameters on the CIFAR10 dataset, while the network pruned by DARTS-GAL only has 0.30 M parameters, which is equivalent to trimming 18.9% of the original network parameters. Meanwhile, the number of parameters of DARTS-GAL is less than that of P-DARTS and PC-DARTS. On the CIFAR100 dataset, the DARTS network had 1.12 M parameters, while the new network optimized by DARTS-GAL had 0.91 M parameters, reducing the original network parameters by 18.75%. Besides, its parameters are also less than that of both P-DARTS and PC-DARTS. This confirms redundancy in the network architecture obtained by splicing the cells obtained by DARTS and that pruning is effective to simplify the network architecture.

In summary, our DARTS-GAL can simplify the network architecture searched by DARTS and enhance the classification accuracy. On the CIFAR10 dataset, the state-of-the-art result is yielded by the proposed DARTS-GAL.

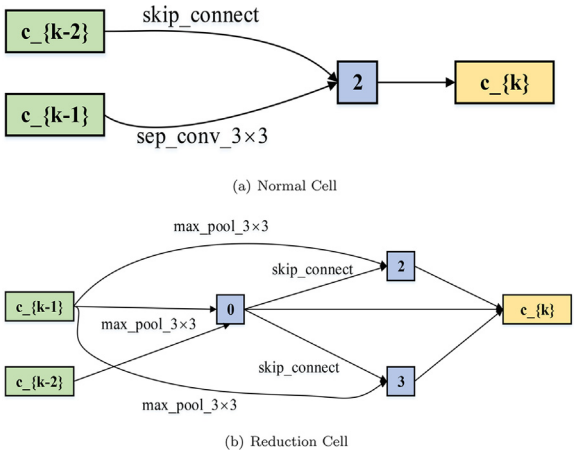


Fig. 10. Cell structure of CIFAR100 before pruning.

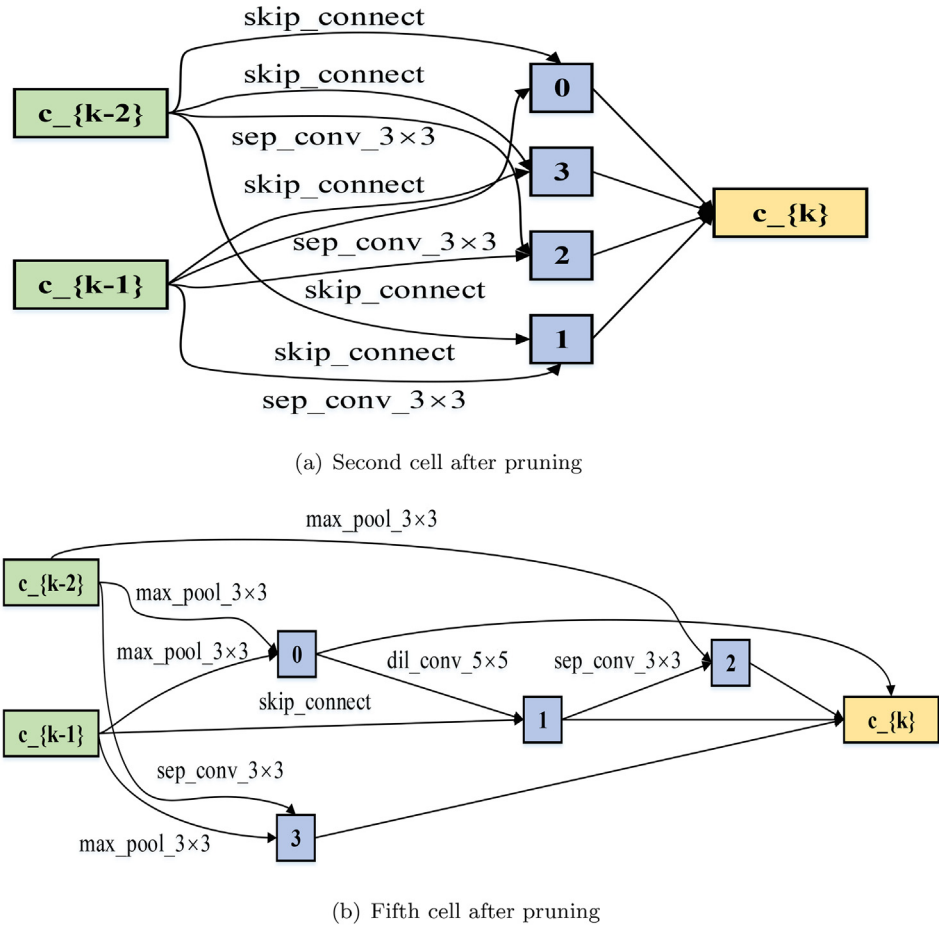
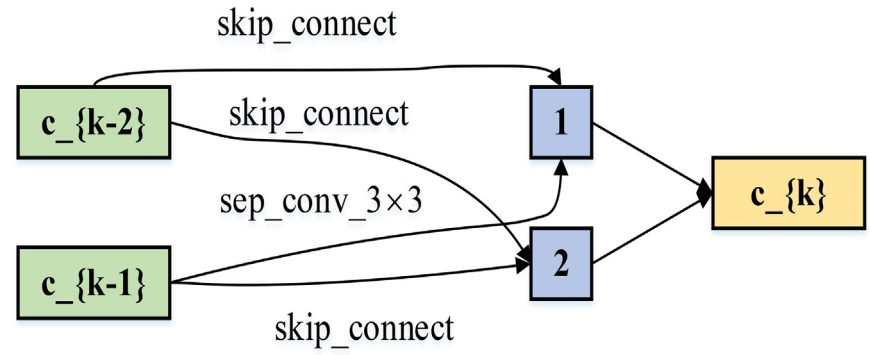


Fig. 11. Examples of cells of CIFAR10 after pruning.

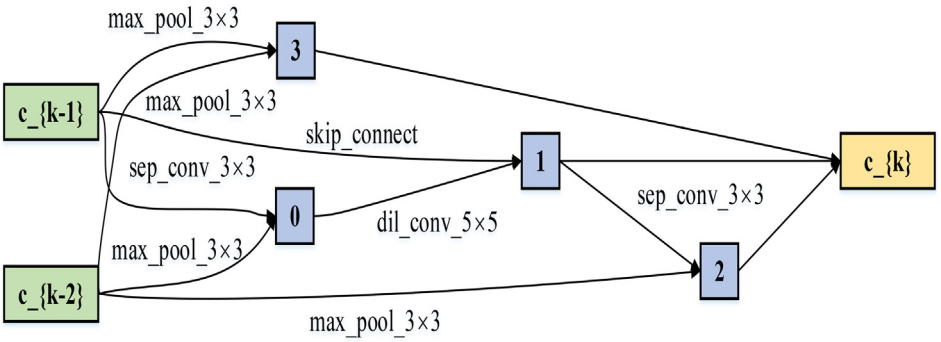
4.3. Large-Size Color Image Experiment

This subsection analyzes the performance of DARTS, P-DARTS, PC-DARTS and DARTS-GAL on a task of large-size color image classification. The dataset used in the experiment is the binary classification dataset of Cats vs Dogs, which is provided by the Kaggle competition platform[2]. Fig. 15 gives some example images of this dataset. The dataset contains 12,500 images of

[2] Kaggle. [Online]. Available:https://www.kaggle.com/c/dogs-vs-cats/data.



(a) First cell after pruning



(b) Fourth cell after pruning

Fig. 12. Example cells of CIFAR100 after pruning.

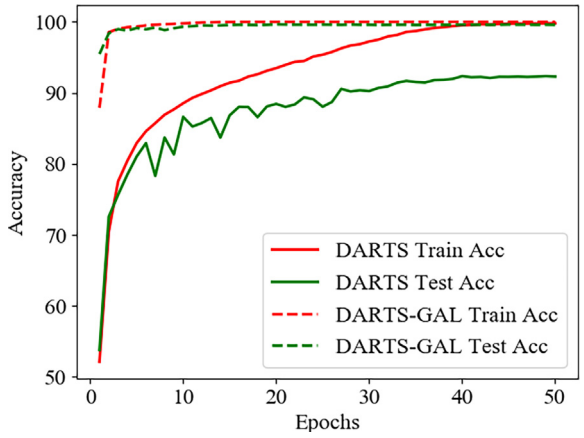


Fig. 13. Training process on CIFAR10.

cats and 12,500 images of dogs, totalling 25,000 images. Given the varying sizes and shapes of the images in the dataset, all the images are scaled to 256×256 prior to training, and then the middle 224×224 regions are cropped. The training set and the test set are divided at a ratio of $8:2$. Hence, the training set contains 20,000 images, whereas the test set contains 5,000 images.

Compared to the MNIST, CIFAR10 and CIFAR100 datasets, the images in the Cats vs Dogs dataset are considerably more prominent, which are RGB color images. However, when NAS is performed on the large-size images by directly using DARTS, the memory of a single GPU will overflow due to the enormous feature map dimensions in the subsequent network. Hence, when performing NAS on the large-size images, a two-stride 3×3 convolution operation is implemented on the input

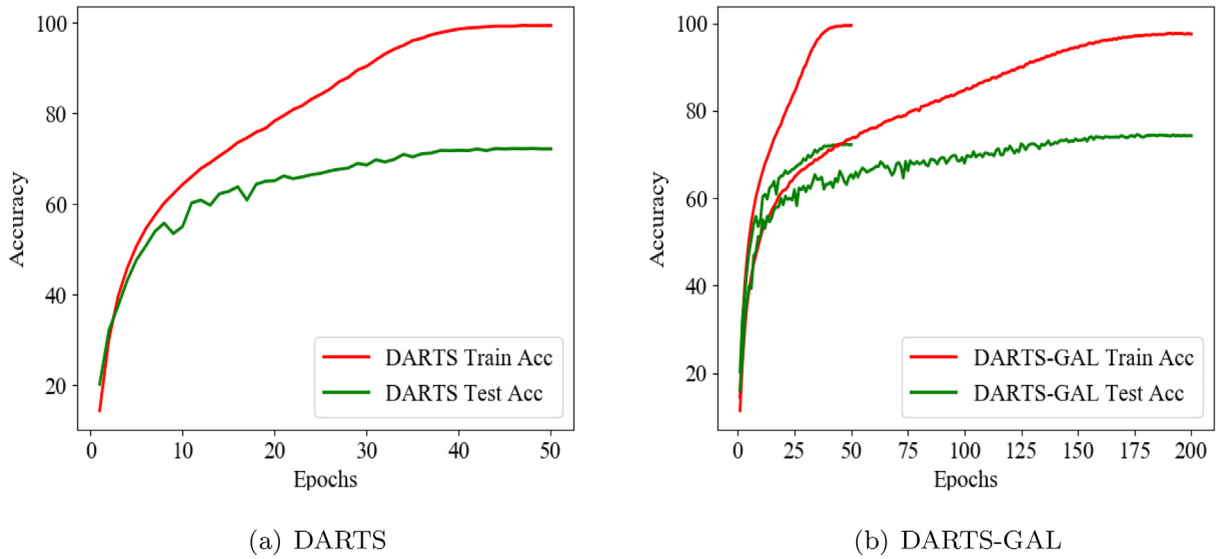


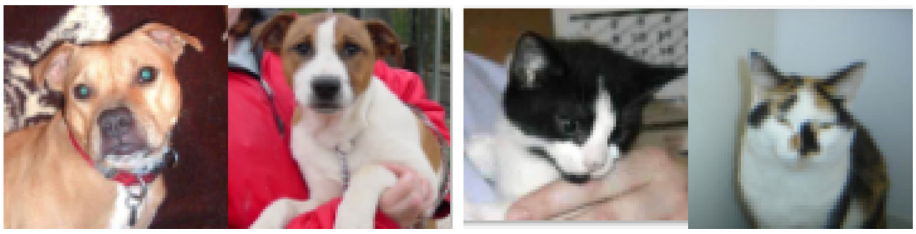
Fig. 14. Training process on CIFAR100.

Table 3
Network pruning on CIFAR10 and CIFAR100.

Datasets	Cell	Type	Reserved intermediate nodes
CIFAR10	1	Normal	None
	2	Normal	2
	3	Normal	2
	4	Normal	1
	5	Reduction	0, 1, 3
	6	Normal	1
	7	Normal	3
	8	Normal	0, 1, 2
	9	Reduction	1, 2, 3
	10	Normal	0
	11	Normal	0, 2
	12	Normal	1, 3
CIFAR100	1	Normal	1, 2
	2	Normal	0, 1, 3
	3	Normal	1
	4	Reduction	1, 2, 3
	5	Normal	1, 2, 3
	6	Normal	0, 1, 2, 3
	7	Reduction	3
	8	Normal	0, 1, 2, 3
	9	Normal	1
	10	Normal	2, 3

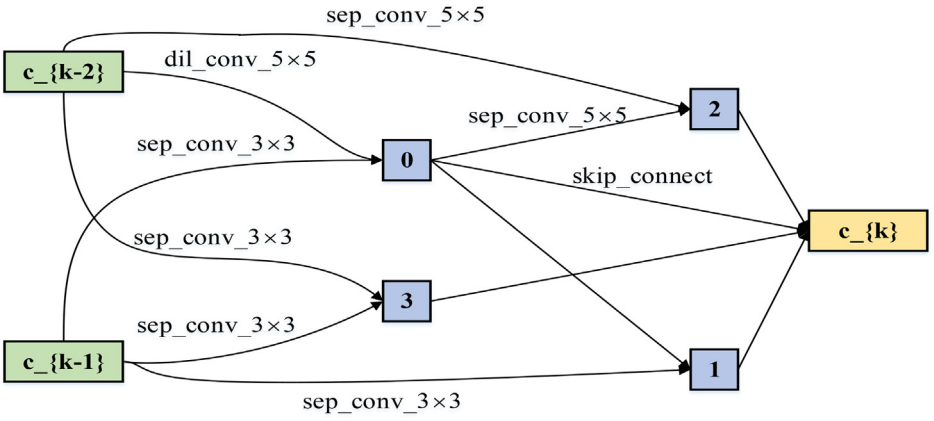
Table 4
Performance comparison of different networks on CIFAR10 and CIFAR100.

Datasets	Methods	Tese Accuracies(%)	Param(M)
CIFAR10	DARTS	92.35	0.37
	P-DARTS	93.49	0.40
	PC-DARTS	93.11	0.42
	DARTS-GAL	**99.60**	**0.30**
CIFAR100	DARTS	72.32	1.12
	P-DARTS	**75.03**	2.34
	PC-DARTS	64.41	3.97
	DARTS-GAL	74.47	**0.91**

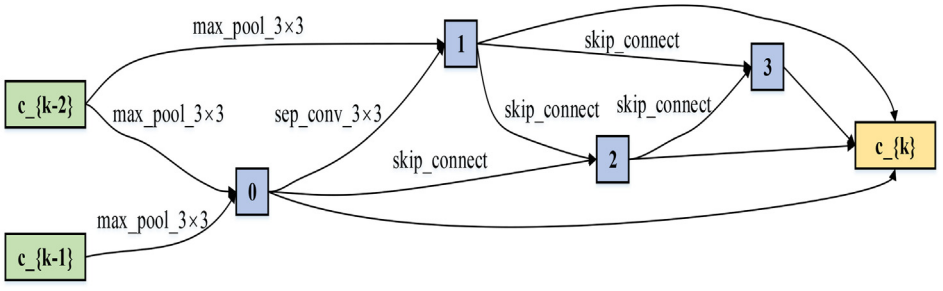


(a) Dog (b) Cat

Fig. 15. Examples of Cats vs. Dogs dataset.



(a) Normal Cell



(b) Reduction Cell

Fig. 16. Cell structure of Cats vs. Dogs dataset before pruning.

images. As a result, the input feature dimensions of the first cell can turn half of the original image size. This practice differs slightly from the direct cell splicing adopted in the previous small-size image experiments. In Fig. 16, the network stacking effect is demonstrated.

During the NAS process by DARTS, the number of searched cells is set to 8, the input channel number of the first cell is set to 16, the number of search training rounds is set to 50, and the number of intermediate state nodes in each cell is set to 4. The architectures of the searched normal, and reduction cells are displayed in Fig. 16. For the searched cell architectures, 8 cells are spliced as shown in Fig. 17. The third and fifth cells in the network are reduction cells, while the rests are normal cells.

The intermediate state nodes in the original 8 cells are subjected to structural pruning through the GAL algorithm-based adversarial training. In Table 5, the retention states of intermediate state nodes for various cells in the network are displayed, while in Fig. 18, the cells structures after pruning by DARTS-GAL are presented. The test results of DARTS, P-DARTS, PC-DARTS and DARTS-GAL are compared in Table 6, and the training processes of DARTS and DARTS-GAL are illustrated in Fig. 19. The following observations can be drawn.

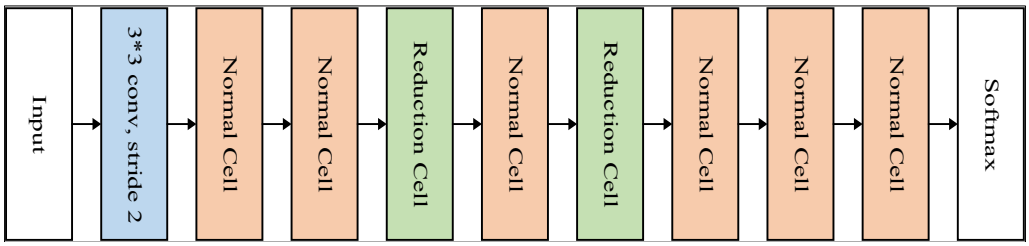
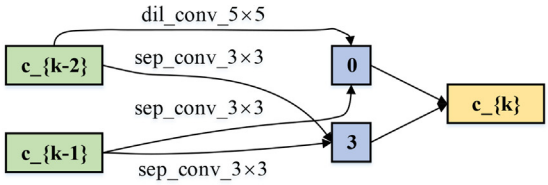


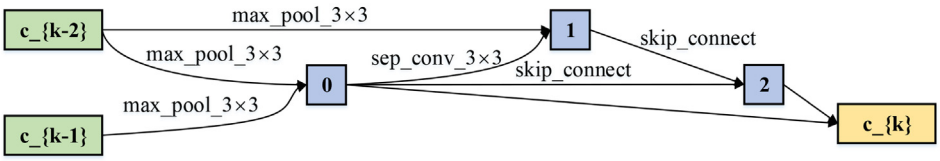
Fig. 17. Large size input images DARTS network.

Table 5
Network pruning on Cats vs. Dogs

Cell	Type	Reserved intermediate nodes
1	Normal	None
2	Normal	0, 3
3	Reduction	0, 1, 2
4	Normal	0
5	Reduction	None
6	Normal	0, 1, 3
7	Normal	2
8	Normal	1, 2, 3



(a) The second cell after pruning



(b) The third cell after pruning

Fig. 18. Example cells of Cats vs. Dogs dataset after pruning.

Table 6
Performance comparison of different networks on Cats vs. Dogs dataset

Methods	Test accuracies(%)	Param(M)
DARTS	90.60	0.30
P-DARTS	90.44	3.43
PC-DARTS	90.53	2.73
DARTS-GAL	**90.70**	0.13

1. In terms of the number of network parameters, the network searched by DARTS has 0.30 M parameters. In comparison, the DARTS-GAL method yields the pruned network with only 0.13 M parameters, 56.67% less parameters than the former network. Besides, its parameters are significantly less than those of both P-DARTS and PC-DARTS. This again indicates the redundancy in the network obtained by directly stacking the cells searched by DARTS, and further pruning of the network can simplify the architecture.

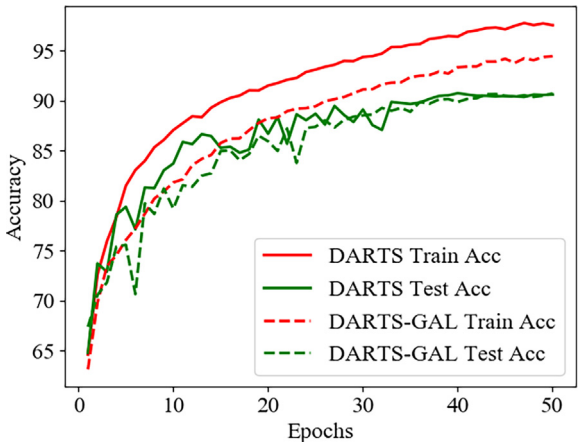


Fig. 19. Training process of DARTS and DARTS-GAL on Cats vs. Dogs dataset.

2. The test accuracy of the network formed by directly stacking the cell architectures searched by DARTS is 90.60%. In contrast, the network simplified and optimized by the proposed DARTS-GAL exhibits a 0.1% improvement in classification accuracy, greater than those both P-DARTS and PC-DARTS. This shows that by removing the redundancy in the architectures searched by DARTS. Thus, our DARTS-GAL is capable of enhancing network performance.
3. Again DARTS-GAL converges slower than DARTS during the training.

4.4. Voiceprint Recognition Experiment

This subsection compares the performance of our DARTS-GAL with VGG16 [1], ResNet18 [2], DARTS [21], P-DARTS [22] and PC-DARTS [23] on the voiceprint recognition task. The data of 100 speakers with the smallest differences in utterance time are extracted from VoxCeleb1 dataset [50] to form our experiment dataset, which is denoted as VoxCeleb1-Min. The voice sampling rate is unified at 16 kHz, monophonic [50].

Since the total utterance duration of the dataset is 1,647 min, the average utterance duration is 16 min per speaker. The utterance of each speaker is segmented into a voice file every 3 sec. Then, these 3-s voice files are converted into 360×360 spectrograms. In Fig. 20 (a) and Fig. 20 (b), the 3-s spectrograms of two different speakers are illustrated. From the spectrogram features showing in Fig. 20, it is obvious that the voiceprints of these two speakers are distinguishable, which is conducive by neural networks to recognizing different speakers. The dataset converted into spectrograms is partitioned into two sets. The training set contains 30,931 spectrogram images, with an average of 310 spectrograms per speaker, while the test set is designed to have 20 spectrograms per speaker, with 2,000 spectrogram images.

Since the extracted spectrogram features are 360×360 large-size color images, we add a two-stride convolution operation before the first cell in the network as shown in Fig. 17, to reduce the dimensionality of the cell input features. This enables the DARTS to search for the network architectures on a single GPU. We employed the DARTS to search for two types of cell architectures, normal cells and reduction cells, as displayed in Fig. 21. During the DARTS-based search, the number of cells is set to 10, the initial channel number of the first cell is 16, and the number of intermediate state nodes is 4.

Afterwards, the 10 cell architectures searched are stacked and trained, where the fourth and seventh cells in the network are reduction cells, while the rests are normal cells. The stacked network is used as the baseline network for the GAL algorithm. Then, the structured pruning of the DARTS network is performed. In Table 7, the retention states of intermediate state nodes for the 10 cells in the network are displayed, while the example cells after pruning are displayed in Fig. 22. The training processes of various methods are illustrated in Fig. 23, and the final classification results are presented in Table 8. Finally, we draw the following observations from the experimental results.

1. In terms of network parameters, VGG16 has hundreds of millions of parameters, and ResNet18 has tens of millions of parameters, which is less than 10% of VGG16. By contrast, the directly spliced DARTS network has only 1.1 million parameters, which is less than 10% of ResNet18. Besides, P-DARTS and PC-DARTS have 0.85 and 0.80 million parameters, both lower than that of DARTS. Moreover, Our DARTS-GAL network only has 0.68 million, which is 61.8% of the DARTS network parameters. This suggests that on the voiceprint recognition dataset, directly splicing the intermediate state nodes in the depth direction as the output of cells in the original DARTS network still produces plenty of redundant intermediate state nodes. Indeed with the DARTS-GAL, the 23 intermediate state nodes of the original DARTS network are pruned, retaining only 17 intermediate state nodes.

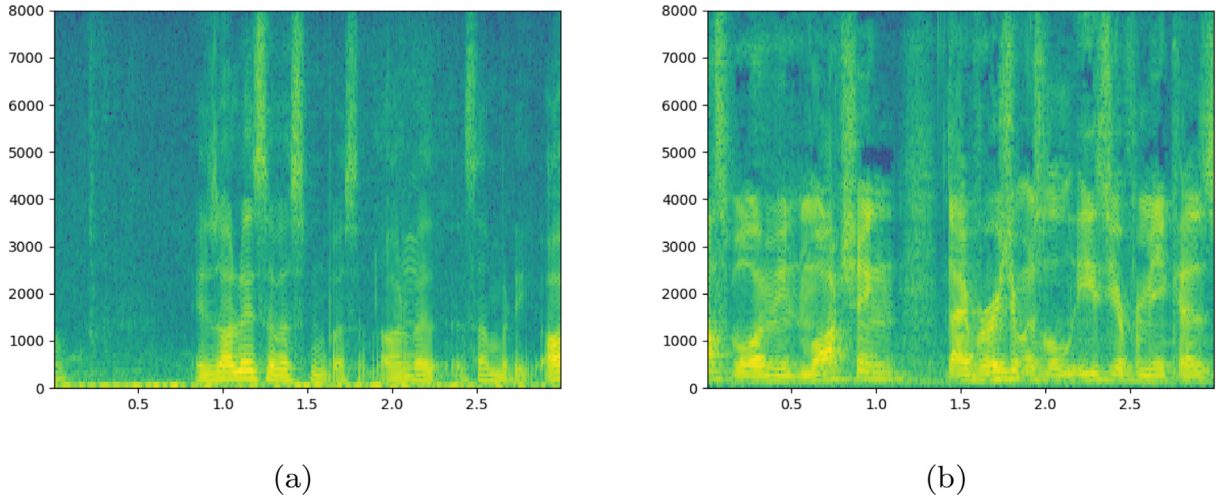


Fig. 20. Two examples of spectrogram.

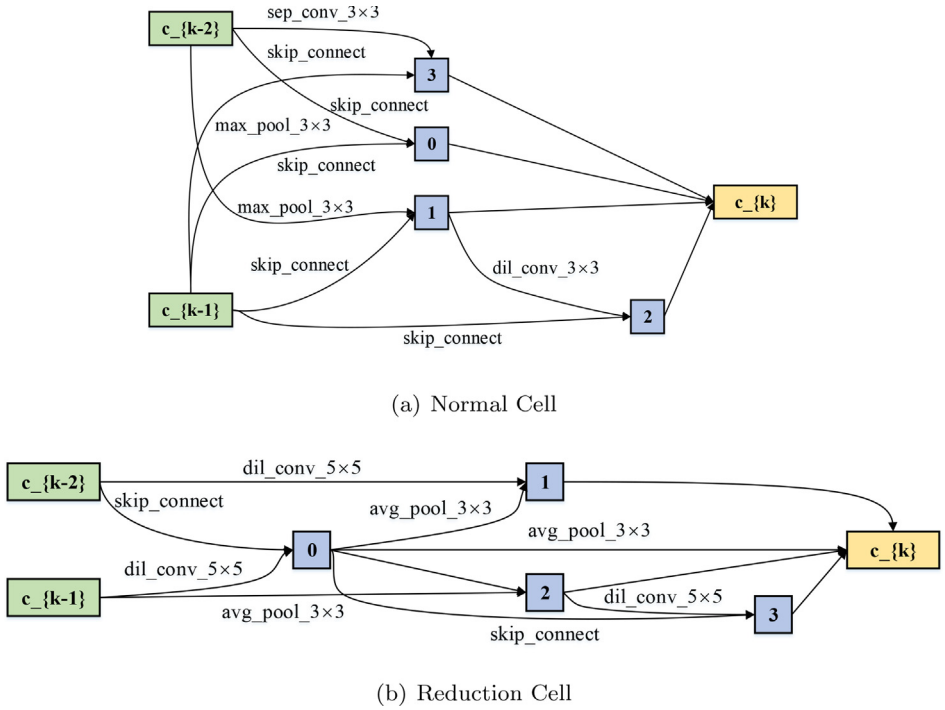


Fig. 21. Cell structures of VoxCeleb1-Min dataset.

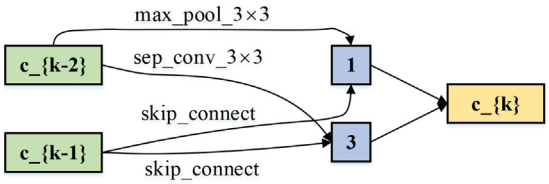
2. In terms of classification performance, VGG16 attains an 87.30% test accuracy, and ResNet18 attains a test accuracy of 88.95%. DARTS, P-DARTS, PC-DARTS and DARTS-GAL achieve much better test performance, with test accuracies of 94.05%, 92.50%, 92.13% and 93.30%, respectively. This experiment, therefore, indicates that classification accuracy can be improved by using the searched network instead of the manually designed network. It also shows that the network architecture searched by DARTS have redundancy, which can be pruned without degrading the classification performance markedly.
3. DARTS-GAL converges slightly slower than DARTS during training.

5. Conclusions

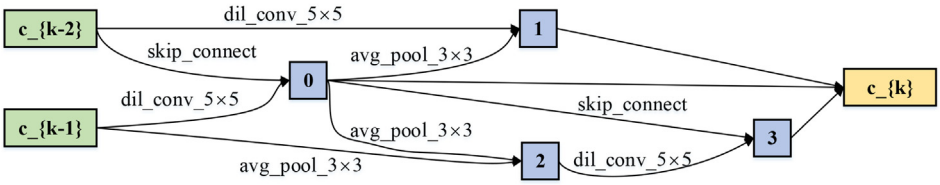
This work has presented a novel neural network optimization method based on DARTS and GAL to simplify the network architectures search (NAS). The proposed DARTS-GAL initially employs the DARTS-based NAS method to search two cell

Table 7
Network pruning on VoxCeleb1-Min by DARTS-GAL

Cell	Type	Reserved intermediate nodes
1	Normal	1
2	Normal	1, 3
3	Normal	0, 3
4	Reduction	0, 1, 3
5	Normal	0
6	Normal	3
7	Reduction	0, 1
8	Normal	2
9	Normal	2, 3
10	Normal	1, 2

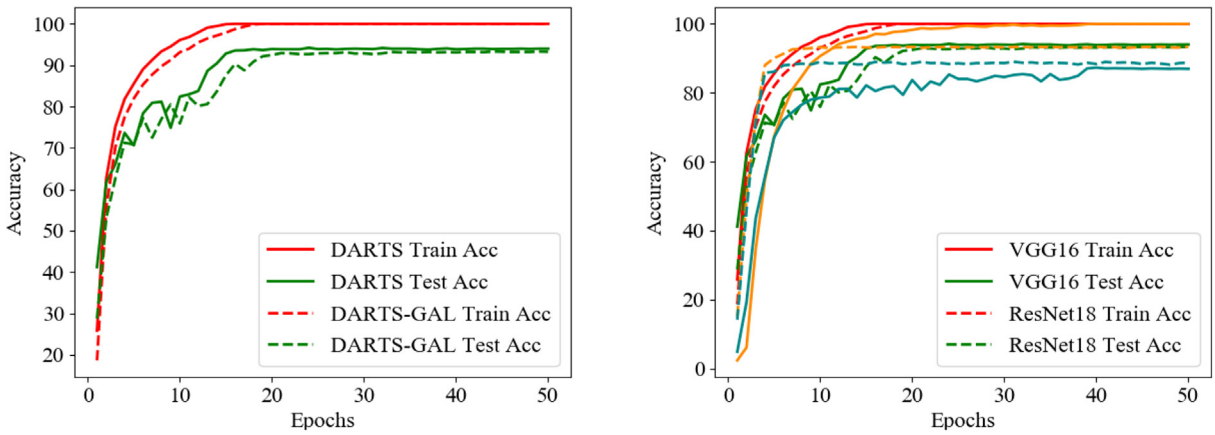


(a) The second cell after pruning



(b) The fourth cell after pruning

Fig. 22. Example cells of VoxCeleb1-Min after pruning.



(a) VGG16 and ResNet18 (b) DARTS and DARTS-GAL

Fig. 23. Training processes of four different methods on VoxCeleb1-Min dataset.

structures, namely, normal cells and reduction cells. Afterwards, it stacks these two cell types alternately in a particular order by borrowing the idea of the Inception network, thereby forming a complete neural network. Finally, it utilizes the GAL to prune the intermediate state nodes of cells for the stacked network, thereby obtaining the final architecture. Experimental results on MNIST, FashionMNIST, CIFAR10, CIAFR100, Cats vs Dogs, and VoxCeleb1-Min datasets have demonstrated

Table 8
Performance comparison of different networks on VoxCeleb1-Min dataset

Methods	Test accuracies(%)	Param(M)
VGG16	87.30	134.67
ResNet18	88.95	11.23
DARTS	**94.05**	1.10
P-DARTS	92.50	0.85
PC-DARTS	92.13	0.80
DARTS-GAL	93.30	**0.68**

that the architectures searched by DARTS after pruning. Thus, the proposed DARTS-GAL reduces the number of parameters considerably and ensures the recognition performance of the pruned network. Among the studied datasets, DARTS-GAL attains the state-of-the-art result on the CAFAR10 dataset and outperforms the original DARTS slightly on several other datasets. Nevertheless, it exhibits slightly decreased recognition accuracies on some datasets. This can probably be attributed to the node-level structured pruning implemented by the DARTS-GAL. Our future work will consider adopting fine-grained structured pruning.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by National Natural Science Foundation of China (61906005, 61806013, 61876010), Program of College of Computer Science, Beijing University of Technology (2019JSJKY006), and Open project of key laboratory of oil and gas resources research, Chinese academy of sciences (KLOR2018-9).

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in Proc. ICLR 2015 (San Diego, CA, USA), May 7–9, 2015, pp. 1–14.
[2] C. Szegedy, et al., "Going deeper with convolutions," in Proc. CVPR 2015 (Boston, MA, USA), Jun. 7–12, 2015, pp. 1–9.
[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. CVPR 2016 (Las Vegas, NV, USA), Jun. 27–30, 2016, pp. 770–778.
[4] G. Huang, Z. Liu, L.V. Der Maaten, and K.Q. Weinberger, "Densely connected convolutional networks," in Proc. CVPR 2017 (Honolulu, HI, USA), Jul. 21–26, 2017, pp. 2261–2269.
[5] J. Donahue, et al., "Long-term recurrent convolutional networks for visual recognition and description," in Proc. CVPR 2015 (Boston, MA, USA), Jun. 7–12, 2015, pp. 2625–2634.
[6] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: A survey, J. Machine Learning Research 20 (55) (2019) 1–21.
[7] Y. Weng, T. Zhou, Y. Li, X. Qiu, NAS-Unet: Neural architecture search for medical image segmentation, IEEE Access 7 (2019) 44247–44257.
[8] B. Zoph and Q.V. Le, "Neural architecture search with reinforcement learning," in Proc. ICLR 2017 (Toulon, France), Apr. 24–26, 2017, pp. 1–16.
[9] X. He, K. Zhao, and X. Chu, ‘‘AutoML: A survey of the state-of-the-art," arXiv:1908.00709, pp. 1–33, 2019.
[10] C.C. Tutum, S. Chockchowwat, E. Vouga, and R. Miikkulainen, "Functional generative design: An evolutionary approach to 3D-printing," in Proc. GECCO 2018 (Kyoto, Japan), Jul. 15–19, 2018, pp. 1379–1386.
[11] D. Szwarcman, D. Civitarese, and M. Vellasco, "Q-NAS revisited: Exploring evolution fitness to improve efficiency," in Proc. BRACIS 2019 (Salvador, Brazil), Oct. 15–18, 2019, pp. 509–514.
[12] M. Sulaiman, Z. Halim, M. Lebbah, M. Waqas, S. Tu, An Evolutionary Computing-Based Efficient Hybrid Task Scheduling Approach for Heterogeneous Computing Environment, J. Grid Computing 19 (11) (Mar. 2021).
[13] M. Sulaiman, Z. Halim, M. Waqas, and D. Aydin, "A hybrid list-based task scheduling scheme for heterogeneous computing," The Journal of Supercomputing, Mar. 2021.
[14] M. Tan, et al., "MnasNet: Platform-aware neural architecture search for mobile," in Proc.CVPR 2019 (Long Beach, CA, USA), Jun. 15–20, 2019, pp. 2820–2828.
[15] S. Tu et al, Reinforcement Learning Assisted Impersonation Attack Detection in Device-to-Device Communications, IEEE Trans. Vehicular Technol. 70 (2) (Feb. 2021) 1474–1479.
[16] S. Tu et al, Security in Fog Computing: A Novel Technique to Tackle an Impersonation Attack, IEEE Access 6 (May 2019) 74993–75001.
[17] Z. Zhong, et al., "Practical block-wise neural network architecture generation," in Proc. CVPR 2018 (Salt Lake City, UT, USA), Jun. 18–22, 2018, pp. 2423–2432.
[18] T. Elsken, J.H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via Lamarckian evolution," in Proc. ICLR 2019 (New Orleans, LA, USA), May 6–9, 2019, pp. 1–23.
[19] B. Zoph, V.K. Vasudevan, J. Shlens, and Q.V. Le, "Learning transferable architectures for scalable image recognition," in Proc. CVPR 2018 (Salt Lake City, UT, USA), 2018, pp. 8697–8710.
[20] E. Real, et al., "Large-scale evolution of image classifiers," in Proc. 34th Int. Conf. Machine Learning (Sydney, Australia), Aug. 6–1, 2017, pp. 1–10.
[21] H. Liu, K. Simonyan, and Y. Yang, ‘‘DARTS: Differentiable architecture search," arXiv:1806.09055, 2018.
[22] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in Proc. ICCV 2019 (Seoul, Korea), Oct. 27-Nov.2, 2019, pp. 1294–1303.
[23] Y. Xu, et al., "PC-DARTS: Partial channel connections for memory-efficient differentiable architecture search," in Proc. ICLR 2020, Apr. 26-May 1, 2020, pp. 1–13.
[24] I.J. Goodfellow, et al., "Generative adversarial nets", in Proc. NIPS 2014 (Montreal, Canada), Dec. 8–13, 2014, pp. 2672–2680.
[25] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, in Proc. ICLR 2017 (Toulon, France) Apr. 24–26, 2017.

[26] S. Rehman, S. Tu, M. Waqas, Y. Huang, O. Rehman, B. Ahmad, S. Ahmad, Unsupervised pre-trained filter learning approach for efficient convolution neural network, Neurocomputing 365 (Jul. 2019) 171–190.
[27] Z. Zhong, Yan J. Yan, W. Wu, et al., Practical block-wise neural network architecture generation, in Proc. CVPR 2018 (Salt Lake City, United States) Jun. 19–21, 2018, pp. 2423–2432.
[28] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, J. Evolutionary Computation 10 (2) (2002) 99–127.
[29] Z. Liu, X. Zhang, T. Jiang, B. Liu, M. Waqas, Y. Li, Infrared salient object detection based on global guided lightweight non-local deep features, Infrared Physics & Technology 115 (2021) 103672.
[30] Z. Liu, M. Waqas, J. Yang, A. Rashid, Z. Han, A Multi-Task CNN for Maritime Target Detection, IEEE Signal Processing Letters 28 (Feb. 2021) 434–438.
[31] K. Kandasamy, W. Neiswanger, J. Schneider, et al., Neural architecture search with Bayesian optimization and optimal transport, in Proc. NIPS 2018 (Montreal, Canada) Dec. 3–8, 2018, pp. 2016–2025.
[32] A. Klein, S. Falkner, J.T. Springenberg, et al., Learning curve prediction with Bayesian Neural Networks, in Proc. ICLR 2017 (Toulon, France) Apr. 24–26, 2017.
[33] H. Zhou, M. Yang, J. Wang, et al., BayesNAS: A Bayesian Approach for Neural Architecture Search, in Proc. ICML 2019 (California, USA) Jun. 9–15, 2019.
[34] W. Grathwohl, E. Creager, S.K.S. Ghasemipour, et al., Gradient-based Optimization of Neural Network Architecture, in Proc. ICLR 2018 (Vancouver, Canada) Apr. 30-May 3, 2018.
[35] H. Liang, S. Zhang, J. Sun, et al., DARTS+: Improved Differentiable Architecture Search with Early Stopping, arXiv:1909.06035, pp. 1–15, 2019.
[36] X. Chu, B. Zhang, X. Li, Noisy Differentiable Architecture Search, arXiv:2005.03566, pp. 1–14, 2020.
[37] S. Han, J. Pool, J. Tran, et al., Learning both weights and connections for efficient neural network, in Proc. NIPS 2015 (Vancouver, Canada), Dec. 7–12, 2015, pp. 1135–1143.
[38] B. Hassibi, D.G. Stork, Second order derivatives for network pruning: Optimal brain surgeon, in Proc. NIPS 1993 (Colorado, USA), 1993, pp. 164–171.
[39] Y. LeCun, J.S. Denker, S.A. Solla, Optimal brain damage, in Proc. NIPS 1990, 1990, pp. 598–605.
[40] J. Luo, J. Wu, W. Lin, Thinet: A filter level pruning method for deep neural network compression, in Proc. ICCV 2017 (Venice, Italy), Oct. 22-29, 2017, pp. 5058–5066.
[41] S. Lin, R. Ji, Y. Li, et al., Accelerating convolutional networks via global and dynamic filter pruning, in Proc. IJCAI 2018 (Stockholm, Sweden), Jul. 13–19, 2018, pp. 2425–2332.
[42] Z. Huang, N. Wang, Data-driven sparse structure selection for deep neural networks, in Proc. ECCV 2018 (Munich, Gernmany), Sep. 8–14, 2018, pp. 304–320.
[43] Z. Liu, J. Li, Z. Shen, et al, Learning efficient convolutional networks through network slimming, in Proc, 22–29, (Venice, Italy), Oct, 2017, pp. 2755–2763.
[44] Y. Li, Y. Song, L. Jia, et al, Intelligent Fault Diagnosis by Fusing Domain Adversarial Training and Maximum Mean Discrepancy via Ensemble Learning, IEEE Trans. Ind. Inform. 17 (4) (2020) 2833–2841.
[45] Q. Guo, Y. Li, Y. Song, et al, Intelligent fault diagnosis method based on full 1-D convolutional generative adversarial network, IEEE Trans. Ind. Inform. 16 (11) (2019) 2044–2053.
[46] S. Lin, et al., "Towards optimal structured CNN pruning via generative adversarial learning," in Proc. CVPR 2019 (Long Beach, CA, USA), Jun. 16–20, 2019, pp. 2790–2799.
[47] L. Yann, B. Leon, B. Yoshua, H. Patrick, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[48] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-NMIST: A novel image dataset for benchmarking machine learning algorithms," arXiv:1708.07747, pp. 1–6, 2017.
[49] A. Krizhevsky, "Learning multiple layers of features from tiny images," Technical Report TR-2009, University of Toronto, Toronto, 2009.
[50] A. Nagrani, J.S. Chung, and A. Zisserman, "VoxCeleb: A large-scale speaker identification dataset," arXiv:1706.08612, pp. 1–6, 2017.