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], Shanshan Tu[a], Yujian Li[a], Zahid Halim[b], Sheng Chen[d,e]

[a]*Faculty of Information Technology, Beijing University of Technology, Beijing, China, 100124*
[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*

Abstract

Neural network architecture search automatically configures a set of network architectures according to the targeted rules. It not only relieves the human-dependent effort and repetitive resources consumption for designing neural network architectures but also makes the task of finding the optimum network architecture with better performance much easier. 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 finds the module structures utilizing the DARTS algorithm. Afterwards, the retrieved modules are stacked to derive the initial neural network architecture. Next, the GAL is utilized 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 to reduce the network parameters without compromising the network performance. Experimental results on benchmark datasets, MNIST, FashionMNIST, CIFAR10, 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 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.

Keywords: Deep learning, neural network architecture search, differentiable architecture search (DARTS), pruning, generative adversarial learning (GAL)

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, 6, 7]. These novel neural network architectures, which have achieved breakthrough progress, are developed however by professionals through persistent experimentation as well as attempts based on baseline theories and empirical analysis. This process requires domain expertise and consumes huge amount of computational resources. The introduction of network architecture search (NAS) [8, 9] has significantly alleviated this problem. The architecture of neural network can now be designed by an algorithm, and no human expert needs to involve in the designing process. The performance of networks derived by an NAS algorithm has exceeded the networks designed by experts on certain specific tasks' datasets [10, 11]. Interestingly, NAS algorithms are also capable of designing some novel network architectures that have not been proposed by a human expert in the past [12, 13]. For example, Google uses reinforcement learning approaches to devise the neural network architectures [13], and the devised neural networks outperformed the manually designed networks in both the 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 [14]. In the early days, the process of finding the neural network hyperparameters was often referred to as the hyperparameter optimization. Common methods included the random search [15], Bayesian optimization [16], and evolutionary algorithm [17], to name a few. The conventional network hyperparameters typically only included the learning rate, batch size, and regularization coefficient. However, in general, hyperparameters are high-dimensional, which also include 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 period of research, most of the works focused on evolutionary algorithms [18, 19, 20]. 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 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. Traditional evolutionary algorithms could no longer meet the requirements on 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. Majority of these algorithms were inspired by 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 use them to automatically find architectures and to look into the in-depth details of network performance. For instance, a Google research team proposed NASNet [21]. 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 the reinforcement learning. 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 [22, 23, 24, 25], in order 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 [26]. It innovatively converted the discrete NAS optimization problem into a differentiable process, so that it can be solved by the gradient descent method. This original DARTS, however, 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 P-DARTS method [27], 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 PC-DARTS method [28]. This method just performs the differentiable architecture search in some channels, which reduces the video memory chip's memory in the DARTS training substantially.

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, there remain redundant connections and state nodes within the cells. To address this problem, this work presents a novel network architecture optimization method based on DARTS and generative adversarial learning (GAL) [29], called DRATS-GAL. Our method performs structural pruning on the network architecture searched by DARTS, which removes some redundant connections and state nodes to re-optimize the original 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 reminder of this paper is arranged as follows. Section 2 details our DARTS-GAL based network architecture optimizer. Section 3 presents the experimental results and performance analysis. Section 4 concludes this work.

2. DARTS-GAL Based Network Architecture Optimizer

The DARTS algorithm constructs the network by splicing the searched cell architectures in a direct manner. 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 by using a GAL based neural network pruning algorithm. Finally, we retrain the newly pruned network. Figure 1 depicts the flowchart of the proposed method.

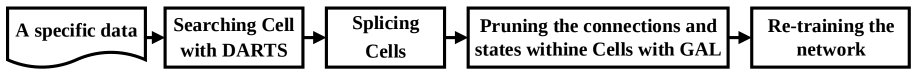

A specific data
Searching Cell with DARTS
Splicing Cells
Pruning the connections and states withine Cells with GAL
Re-training the network

Figure 1: Flowchart of DARTS-GAL

2.1. DARTS-Based Search

DARTS, as a gradient-based NAS algorithm, aims to search for two types of cell architectures: normal cell and reduction cell [26]. Normal cells refer to the cell computing units that do not alter the size of the input feature maps, while 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 that contain N ordered nodes, where the value of 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:

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

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 each two connected nodes, and each operator is assigned with a corresponding structural parameter for participating in the network learning and training together. The operator with the highest final structural parameter value is retained. 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:

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

To determine the internal cell architectures, a structural weight is assigned for each candidate operator between every two nodes during training. 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 intermediate state node in the search process can be expressed as:

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

$\mathcal{O}$ 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 continuous problem, which can then be solved by the gradient descent method.

2.2. Cell Splicing

With reference to the Inception network construction method [2], DARTS stacks the normal and reduction cells alternately to obtain the entire network architecture. To be specific, at the locations 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, which are the output of the adjacent previous cell and the output of the cell just prior to the previous cell. Each cell has an output value, which is 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 Figure 2, the specific connection modes between various cells in the DARTS network are illustrated.

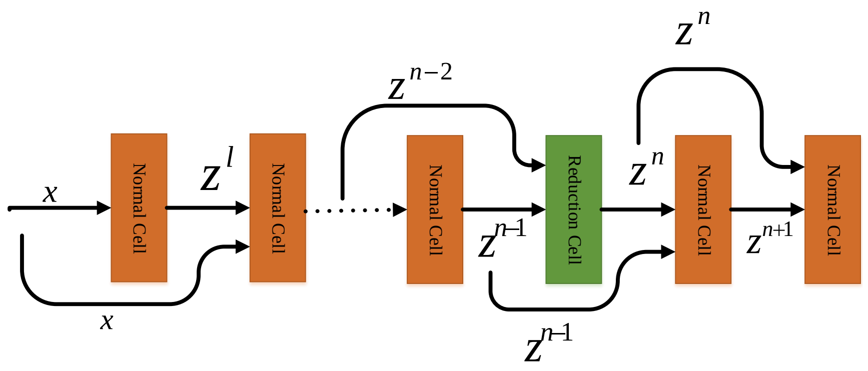


Figure 2: A model spliced by cells

2.3. 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 [30]. It trims the convolutional kernels, branch structures and block structures in the neural network. Figure 3 presents the flowchart of this GAL-based pruning algorithm.

As illustrated in Figure 3, the algorithm sets the network searched by the DARTS as a baseline network initially, 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

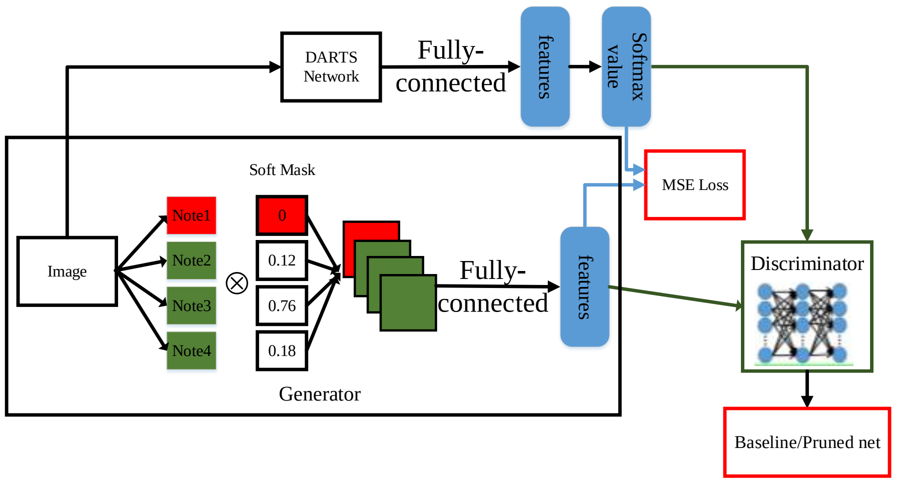


Figure 3: Flowchart of GAL-based pruning algorithm

as a generator G, and the Softmax prediction value output by G is marked as a false label. Next, 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 adversarial game. During the adversarial learning, part of the soft masks in the generator are sparsified to 0, so as 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, the structured pruning of the intermediate state nodes for each cell can be completed according to the final soft mask values.

During the adversarial learning, the parameters of the baseline network are fixed. By contrast, the parameters of the network (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 the course of learning. The overall optimization objective function can be expressed as:

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

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

$$\begin{aligned} L_{\mathrm{Adv}}\left(W_G, s, W_D\right) =& E_{f_b(x) \sim p_b(x)}\left[\log\left(D\left(f_b(x), W_D\right)\right)\right] \\ &+ E_{f_g(x,z) \sim (p_g(x), p_z(z))}\left[\log\left(1 - D\left(f_g(x,z), W_D\right)\right)\right], \quad (5) \\ L_{\mathrm{data}}\left(W_G, s\right) =& \frac{1}{2n} \sum_x \left\| f_b(x) - G\left(x, W_G, s\right) \right\|_2^2, \quad (6) \\ L_{\mathrm{reg}}\left(W_G, s, W_D\right) =& R\left(W_G\right) + R_\lambda(s) + R\left(W_D\right). \quad (7) \end{aligned}$$

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, $D\left(f_b(x), W_D\right)$ 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, $R\left(W_G\right) = \frac{1}{2}\left\|W_G\right\|_2^2$ is the l_2 regularizer of the network to be pruned, $R_\lambda(s) = \lambda \left\|s\right\|_1$ is the sparse regularizer of s with regularization parameter λ, and $R(W_D) = E_{f_g(x) \sim p_g(x)}\left[\log\left(D\left(f_g(x), W_D\right)\right)\right]$ represents the discriminator regularizer to prevent the discriminator from dominating the training.

2.4. Training the Pruned Network

After the pruning optimization of the network architecture, we obtain a new network with both less states and less connections. Compared with the network architecture before pruning, the model after pruning is obviously different. Then, we load this new model to train with the original dataset, while keeping the hyper-parameters, such as learning rate, the size of batch, dropout, unchanged to have a clear comparison with the network before pruning.

3. Experimental Results

To validate the performance of our proposed method, performance comparisons are made first between our DARTS-GAL and other methods on two tasks of small-size image classification. Afterward, 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.

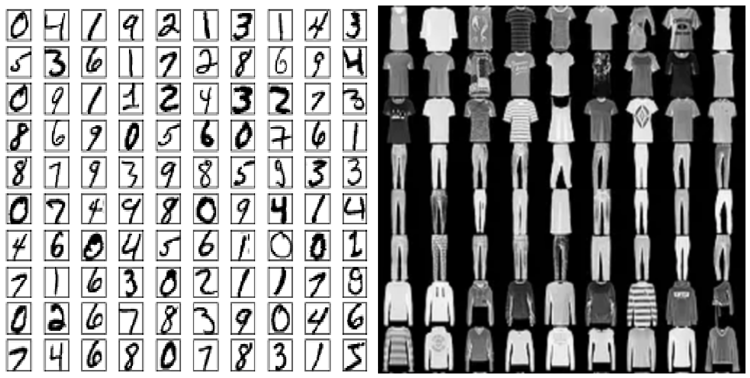
3.1. Small-Size Grayscale Image Classification Experiment

This subsection compares the classification performance of our DARTS-GAL with LeNet [31] and DARTS [26] for the small-size grayscale images. The datasets adopted are MNIST [31] and FashionMNIST [32].

MNIST is a handwritten digit dataset [31], 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 Figure 4(a) displays some example images. FashionMNIST is a product image dataset containing ten categories of products [32], and Figure 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 exactly identical to the MNIST.

LeNet is a human-designed full convolutional neural network for handwritten digit recognition [31]. We train this LeNet with the training sets of MNIST and FashionMNIST, and evaluate the corresponding test performance.

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

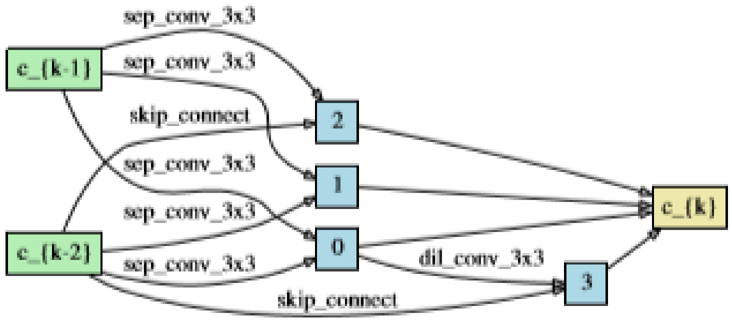


(a) MNIST (b) FashionMNIST

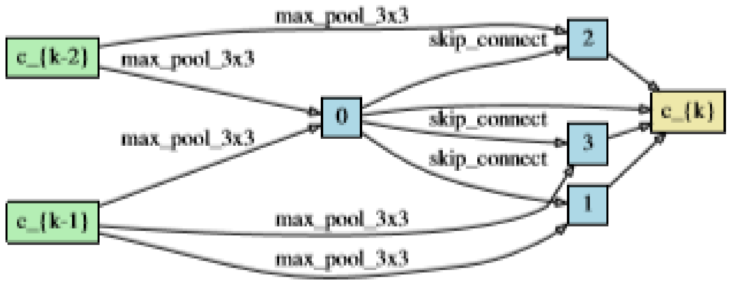
Figure 4: Examples of MNIST series datasets

For the DARTS, we do not use the training sets of MNIST and Fashion-MNIST to search for the network architecture. Rather, we use the network architecture searched by the DARTS on CIFAR10 dataset [33], as described in the original DARTS paper. The reason is that these two datasets may be too 'simple' and a NAS may tend to result in shallow network architectures [9]. With the architecture found by the DARTS on CIFAR10 dataset, we then use the training sets of MNIST and FashionMNIST to train the parameters of this network. The test performance are then evaluated on the test datasets of MNIST and FashionMNIST. In Figure 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 splice 5 cell architectures as the baseline network of the both datasets. The first, fourth and fifth cells of the spliced network are normal cells, 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 Figure 5 on the MNIST and FashionMNIST datasets after pruning. Figures 6 and 7 show the example cells after pruning for the two datasets, respectively. The pruned network is then retrained, and the test performance is evaluated.

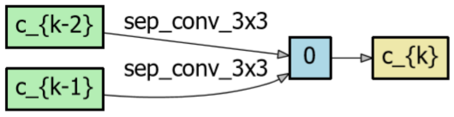


(a) Normal Cell

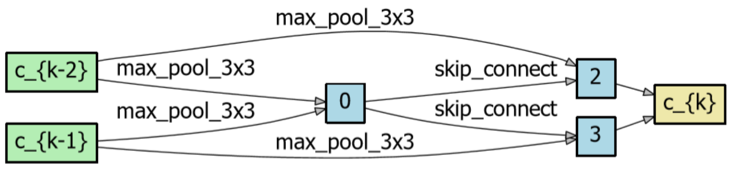


(b) Reduction Cell

Figure 5: Cell structure search on CIFAR10 by DARTS



(a) First cell after pruning



(b) Second cell after pruning

Figure 6: Example cells for MNIST after pruning

Table 2 summarizes the test accuracies of LeNet, DARTS and DARTS-GAL
240 on MNIST and FashionMNIST datasets, where Param(M) stands for the num-

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

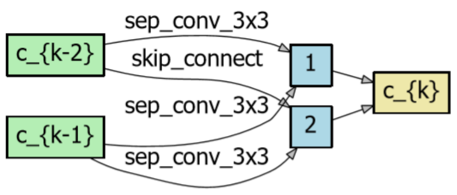
Datasets	Cell	Type	Reserved intermediate nodes
MNIST	1	Normal	0
	2	Reduction	2, 3
	3	Reduction	1
	4	Normal	2
	5	Normal	1, 2
FashionMNIST	1	Normal	1, 2
	2	Reduction	0, 1
	3	Reduction	3
	4	Normal	0, 1, 2
	5	Normal	1

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

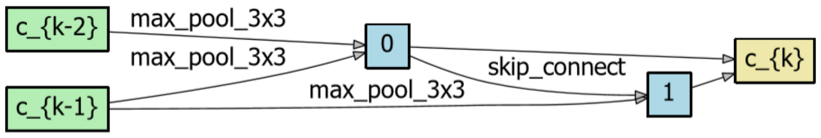
Datasets	Models	Test Accuracies(%)	Param(M)
MNIST	LeNet	**99.60**	1.20
	DARTS	99.41	0.13
	DARTS-GAL	99.32	**0.09**
FashionMNIST	LeNet	91.32	1.20
	DARTS	92.33	0.13
	DARTS-GAL	**92.50**	**0.10**

ber of parameters (millions). Figures 8 and 9 depict the accuracies 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 initial channel numbers of the first cell are both set to 16. From the results shown in Tables 1 to 2 and Figures 6 to 9, we can draw the following observations.

1. Not surprisingly, LeNet attains the best test accuracy of 99.60% on MNIST dataset, as it is a human-designed network specifically catering for hand-



(a) The first cell after pruning



(b) The second cell after pruning

Figure 7: Example cells of FashionMNIST after pruning

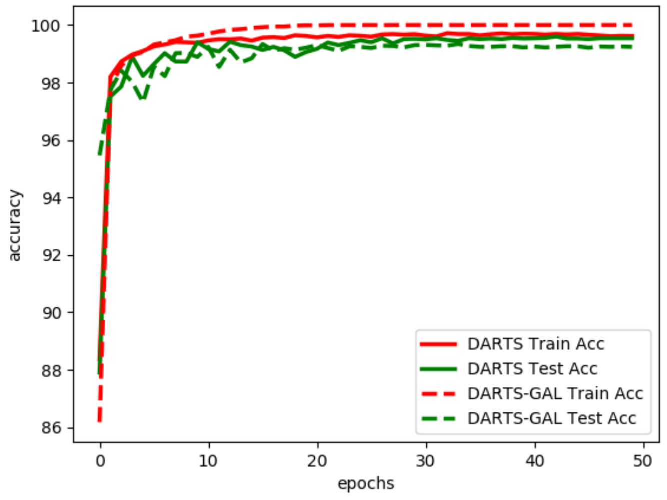


Figure 8: Training process of DARTS and DARTS-GAL on MNIST

written digit recognition. But it is a full convolutional neural network with a very large number of parameters (1.2 millions).

2. DARTS is capable of arriving a much sparser network architecture with only 0.13 millions of parameters, which is less than 11% of LeNet. Our DARTS-GAL further removes the redundancy in the DARTS network architecture. For MNIST dataset, DARTS-GAL achieves a network with only 0.09 millions of parameters, 0.04 millions less than DARTS, while for

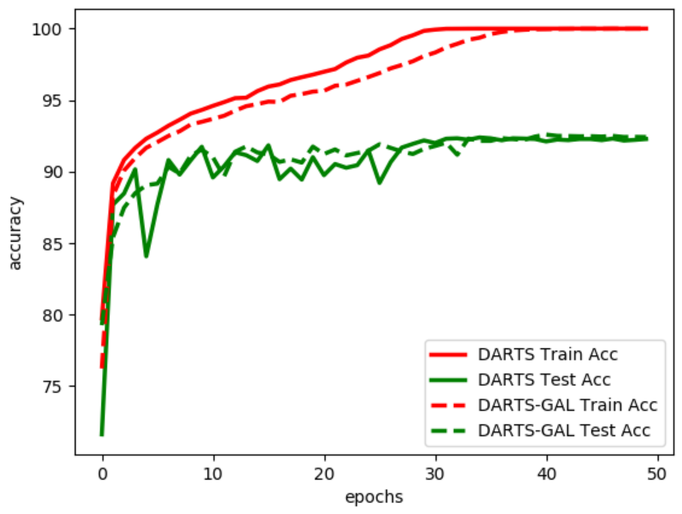


Figure 9: Training process of DARTS and DARTS-GAL on FashionMNIST

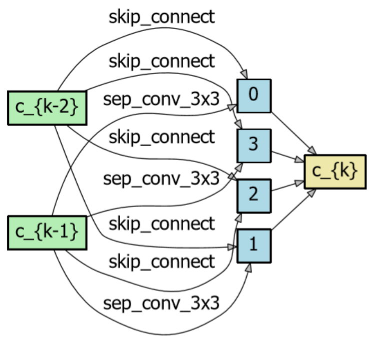
FashionMNIST dataset, DARTS-GAL arrives a network with only 0.10 millions of parameters, 0.03 millions less than DARTS.

3. For MNIST dataset, our DARTS-GAL attains a test accuracy of 99.32%, which is 0.28% and 0.09% lower than the results by LeNet and DARTS, respectively. For FashionMNIST dataset, our DARTS-GAL attains a test accuracy of 92.50%, which is actually 1.18% and 0.17% higher than the results of LeNet and DARTS, respectively.
4. DARTS converges slower than the DARTS during training, as can be seen from Figures 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.

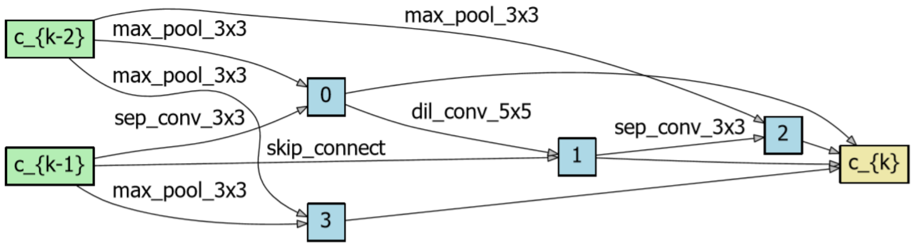
Combining 2. and 3., we can conclude that the proposed DARTS-GAL is capable of simplifying the network architecture searched by DARTS and reducing the network parameters, while maintaining the network performance.

3.2. Small-Size Color Image Classification Experiment

This subsection compares the performances of our DARTS-GAL and DARTS on the datasets of CIFAR10 [33] and CIFAR100 [33].

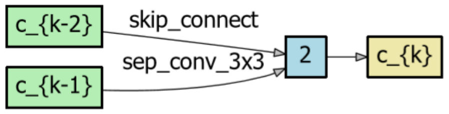


(a) Normal Cell

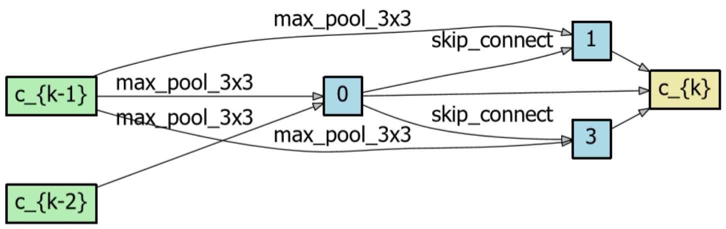


(b) Reduction Cell

Figure 10: Cell structure of CIFAR100 before pruning

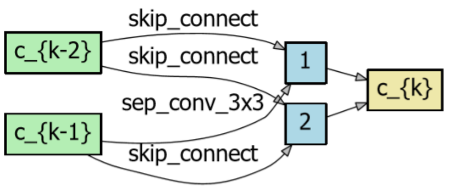


(a) Second cell after pruning

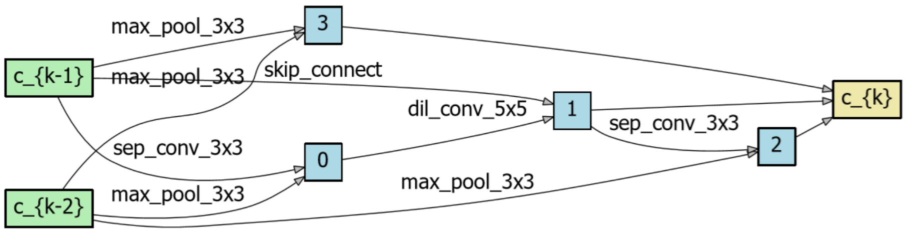


(b) Fifth cell after pruning

Figure 11: Examples of cells of CIFAR10 after pruning



(a) First cell after pruning



(b) Fourth cell after pruning

Figure 12: Example cells of CIFAR100 after pruning

CIFAR10 is a 10-class dataset including classes like 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 the both datasets, the initial channel number of

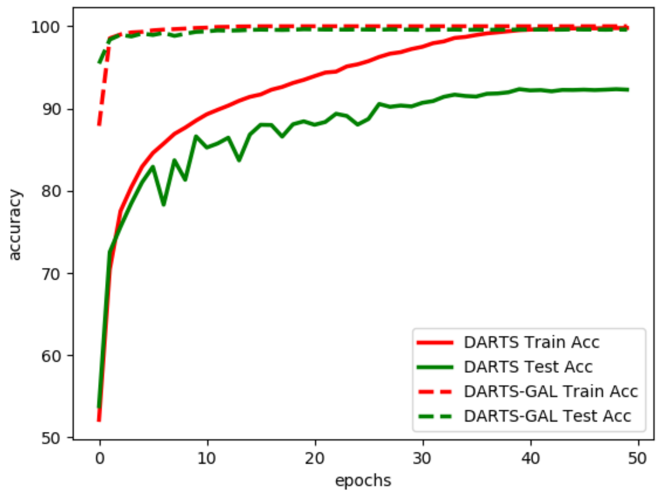
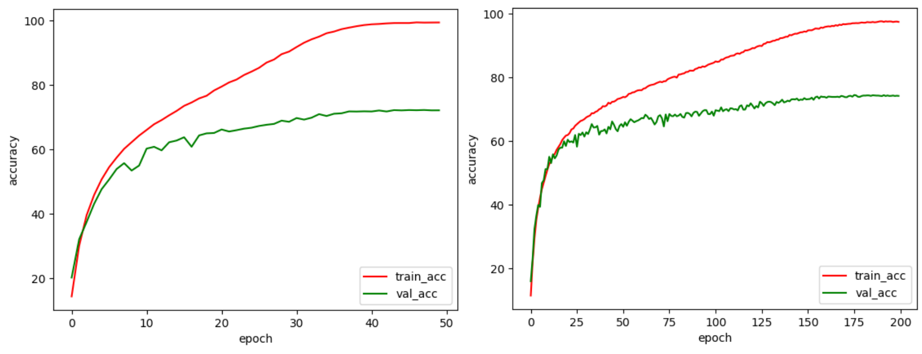


Figure 13: Training process on CIFAR10



(a) DARTS (b) DARTS-GAL

Figure 14: Training process on CIFAR100

the first cell is 16. For CIFAR10 dataset, we splice 12 cells of Figure 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 CIFAR100 dataset, we spliced 10 cell of Figure 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. The number of intermediate state nodes in each cell is 4.

In Figures 11 and 12, the cells structures after pruning by DARTS-GAL on these two datasets are presented. In Figures 13 and 14, the training processes of DARTS and DARTS-GAL 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, as it requires to train the pruned network from scratch.

2. In terms of test accuracy, DARTS-GAL attains a 99.60% test accuracy on CIFAR10 dataset, which is 7.35% higher than the accuracy of DARTS. This is really a state-of-the-art result. On CIFAR100 dataset, DARTS-GAL also outperforms DARTS, achieving a 74.47% test accuracy, which

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

is 1.15% higher than the DARTS result. This suggests that pruning the architectures searched by DARTS not only simplifies the network architectures, but also improves the accuracy of network classification.

3. With regarding the number of network parameters, the network searched by DARTS has 0.369 M parameters on CIFAR10 dataset, while the net-

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

Datasets	Methods	Tese Accuracies(%)	Param(M)
CIFAR10	DARTS	92.35	0.369
	DARTS-GAL	**99.60**	0.298
CIFAR100	DARTS	72.32	1.12
	DARTS-GAL	**74.47%**	0.91

work pruned by DARTS-GAL only has 0.298 M parameters, which is equivalent to trimming 19.2% of the original network parameters. On CIFAR100 dataset, the DARTS network had 1.12 M parameters, while the new network optimized by DARTS-GAL had 0.91 M parameters, a reduction of the original network parameters by 18.75%. This confirms that there remains 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 is capable of simplifying the network architecture searched by DARTS and enhances the classification accuracy. On CIFAR10 dataset, state-of-the-art result is yielded by the proposed DARTS-GAL.

3.3. Large-Size Color Image Experiment

This subsection analyzes the performance of 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 Kaggle competition platform[2]. Figure 15 gives some example images of this dataset. The dataset contains 12,500 images of cats and 12,500 images of dogs, totaling 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

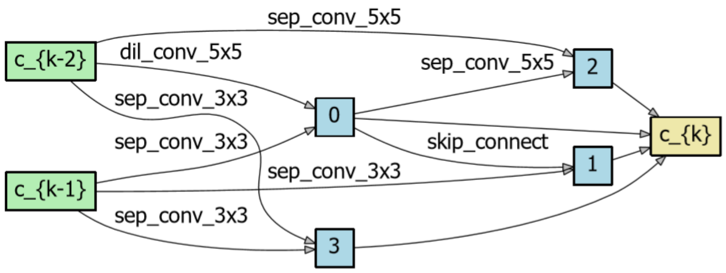
[2]Kaggle. [Online]. Available: `https://www.kaggle.com/c/dogs-vs-cats/data`.

(a) Dog (b) Cat

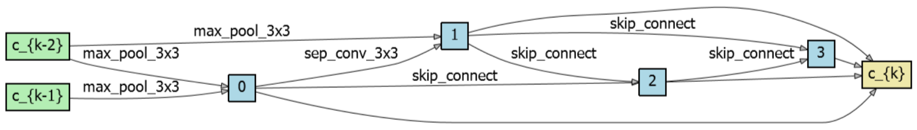
Figure 15: Examples of Cats vs. Dogs dataset

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 larger in size, which are RGB color images as well. However, when NAS is performed on the large-size images by directly using DARTS, the memory of single GPU will overflow due to the excessively large feature map dimensions in the subsequent network. Hence, when performing NAS on the large-size images, two-stride 3×3 convolution operation is implemented on the input images, so that 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



(a) Normal Cell



(b) Reduction Cell

Figure 16: Cell structure of Cats vs. Dogs dataset before pruning

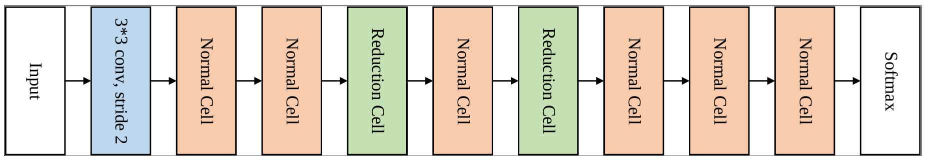


Figure 17: Large size input images DARTS network

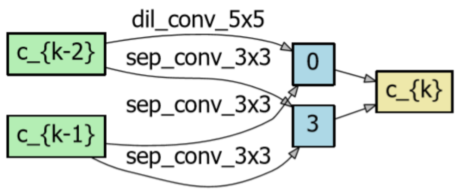
experiments. In Figure 16, the network stacking effect is demonstrated.

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 Figure 16. For the searched cell architectures, 8 cells are spliced as shown in Figure 17. The third and fifth cells in the network are reduction cells, while the rests are normal cells.

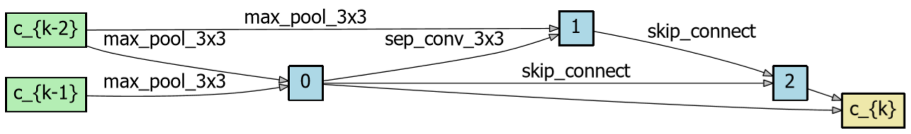
Through the GAL algorithm-based adversarial training, the intermediate state nodes in the original 8 cells are subjected to structural pruning. In Table 5, the retention states of intermediate state nodes for various cells in the network are displayed, while in Figure 18, the cells structures after pruning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

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

Figure 18: Example cells of Cats vs. Dogs dataset after pruning

1. In terms of number of network parameters, the network searched by DARTS has 0.302 M parameters, while the DARTS-GAL method yields the pruned network with only 0.132 M parameters, 62.3% less parameters than the former network. This again indicates that there exists the redundancy in the network obtained by directly stacking the cells searched by DARTS and further pruning of the network can simplify the architecture.
2. The test accuracy of the network formed by directly stacking the cell architectures searched by DARTS is 90.60%, while the network simplified and optimized by the proposed DARTS-GAL exhibits a 0.1% improvement in classification accuracy. This shows that by removing the redundancy in the architectures searched by DARTS, our DARTS-GAL is capable of enhancing the network performance.
3. Again DARTS-GAL converges slower than DARTS during the training.

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

Methods	Test accuracies(%)	Param(M)
DARTS	90.60	0.302
DARTS-GAL	**90.70**	0.132

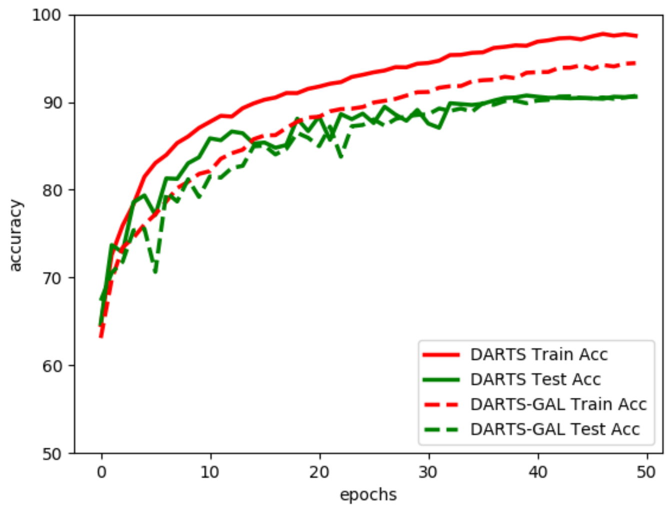


Figure 19: Training process of DARTS and DARTS-GAL on Cats vs. Dogs dataset

3.4. Voiceprint Recognition Experiment

This subsection compares the performance of our DARTS-GAL with VGG16 [1], ResNet18 [2] and DARTS on the voiceprint recognition task. The data of 100 speakers with the smallest differences in utterance time are extracted from VoxCeleb1 dataset [34] to form our experiment dataset, which is denoted as VoxCeleb1-Min. The voice sampling rate is unified at 16 kHz, monophonic [34].

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-sec voice files are converted

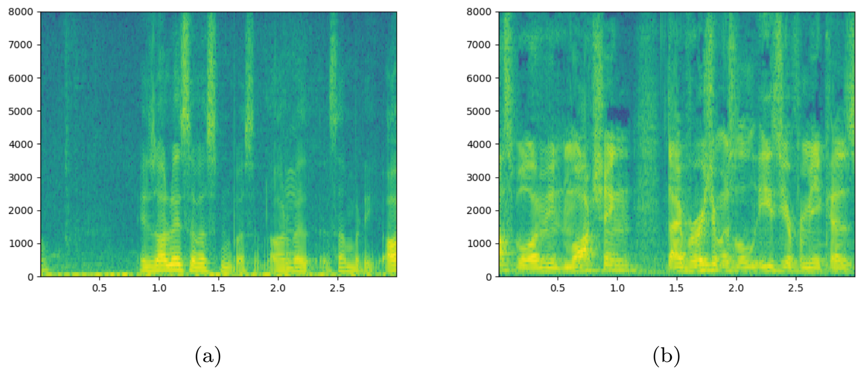
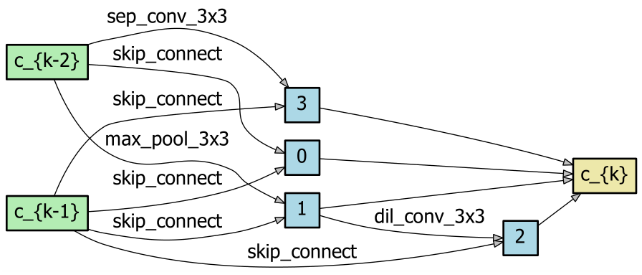


Figure 20: Two examples of spectrogram

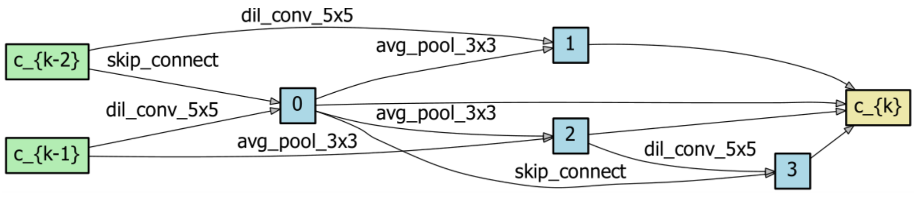
into 360 × 360 spectrograms. In Figure 20 (a) and Figure 20 (b), the 3-sec spectrograms of two different speakers are illustrated. From the spectrogram features showing in Figure 20, it is obvious that the voiceprints of these two speakers are distinguishable, which is conducive by neural networks to 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 total of 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 Figure 17, in order 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 Figure 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



(a) Normal Cell



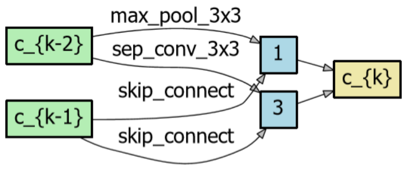
(b) Reduction Cell

Figure 21: Cell structures of VoxCeleb1-Min dataset

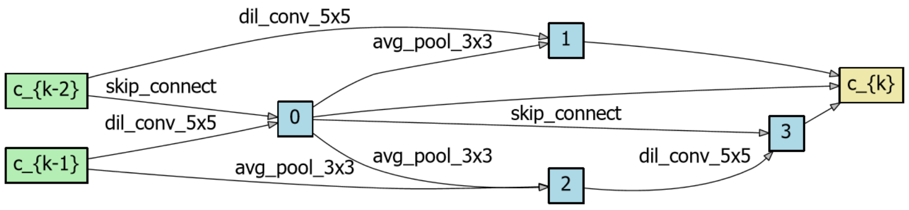
Table 7: Network pruning on VoxCeleb1-Min by DARTS-GAL

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

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, and 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

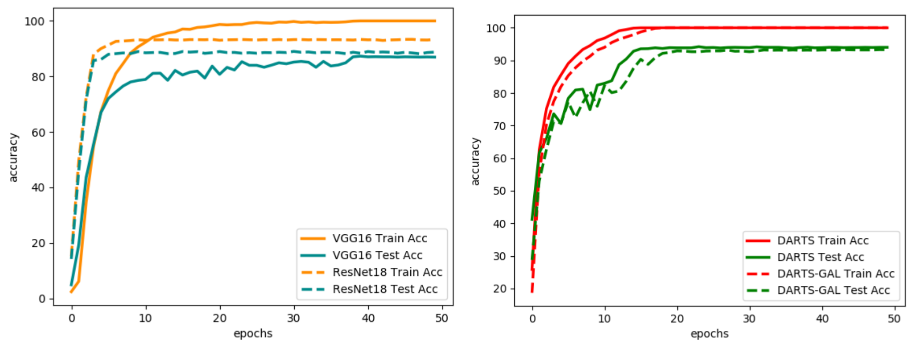


(a) The second cell after pruning



(b) The fourth cell after pruning

Figure 22: Example cells of VoxCeleb1-Min after pruning



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

Figure 23: Training processes of four different methods on VoxCeleb1-Min dataset

displayed in Figure 22. The training processes of various methods are illustrated in Figure 23, and the final classification results are presented in Table 8. 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 millions of parameters, which is less than 10% of ResNet18. Our DARTS-GAL network only has 0.68 millions of parameters, 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 a plenty of redundant intermediate state nodes. Indeed with the DARTS-GAL, the 23 intermediate state nodes of the original DARTS

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
DARTS-GAL	93.30	0.68

network are pruned, retaining only 17 intermediate state nodes.

2. In terms of classification performance, VGG16 attains a 87.30% test accuracy and ResNet18 attains the test accuracy of 88.95%. DARTS and DARTS-GAL achieve much better test performance, with test accuracies of 94.05% 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.

4. 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 types of cell structures, namely, normal cells and reduction cells. Afterward, it stacks these two cell types alternately in a certain order by borrowing the idea of Inception network, thereby forming a complete neural network. Finally, for the stacked network, it utilizes the GAL to prune the intermediate state nodes of cells, thereby obtaining the final architecture. Experimental results on MNIST, FashionMNIST, CIFAR10, CIAFR100, Cats vs. Dogs, and VoxCeleb1-Min datasets have demonstrated that after pruning the architectures searched by DARTS, the proposed DARTS-GAL not only reduces the number of parameters considerably, but also ensures the recognition performance of the pruned network. Among the studied datasets, DARTS-GAL attains the state-of-the-art result on CAFAR10 dataset and also 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 the fine-grained structured pruning.

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).

5. 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] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84—90, Jun. 2017.

[6] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. ICASSP 2013* (Vancouver, BC, Canada), May 26-31, 2013, pp. 6645–6649.

[7] 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.

[8] C. Liu, *et al.*, "Progressive neural architecture search," in *Proc. ECCV 2018* (Munich, Germany), Sep.8-14, 2018, pp. 19–34.

[9] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Machine Learning Research*, vol 20, no. 55, pp. 1–21, 2019.

[10] G. Ghiasi, T. Lin, and Q. V. Le, "NAS-FPN: Learning scalable feature pyramid architecture for object detection,"in *Proc. CVPR 2019* (Long Beach, CA, USA), Jun.16-20, 2019, pp. 7036–7045.

[11] Y. Weng, T. Zhou, Y. Li, and X. Qiu, "NAS-Unet: Neural architecture search for medical image segmentation,' *IEEE Access*, vol. 7, pp. 44247–44257, 2019.

[12] H. Pham, *et al.*, "Efficient neural architecture search via parameter sharing," in *Proc. 35th Int. Conf. Machine Learning* (Stockholm, Swede), Jul.10-15, 2018, pp. 1–10.

[13] 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.

[14] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *arXiv:1908.00709*, pp. 1–33, 2019.

[15] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Machine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012.

[16] A. Mohammad-Djafar, "Joint estimation of parameters and hyperparameters in a Bayesian approach of solving inverse problems," in *Proc. ICIP 1996* (Lausanne, Switzerland), Sep.19, 1996, pp. 473–477.

[17] 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.

[18] 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.

[19] J. D. Lohn, W. F. Kraus, D. S. Linden, and S. Colombano, “Evolutionary optimization of Yagi-Uda antennas,” in *Proc. ICES 2001* (Tokyo, Japan), Oct.3-5, 2001, pp. 236–243.

[20] I. B. D. De Andrade, T. Januario, G. L. Pappa, and G. R. Mateus, “An evolutionary algorithm to the density control, coverage and routing multi-period problem in wireless sensor networks,” in *Proc. 2010 IEEE Conf. Evolutionary Computation* (Barcelona, Spain), Jul.18-23, 2010, pp. 1–8.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] I. J. Goodfellow, *et al.*, "Generative adversarial nets", in *Proc. NIPS 2014* (Montreal, Canada), Dec.8-13, 2014, pp. 2672–2680.

[30] 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.

[31] L. Yann, B. Leon, B. Yoshua, and H. Patrick, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[32] 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.

[33] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Technical Report TR-2009*, University of Toronto, Toronto, 2009.

[34] A. Nagrani, J. S. Chung, and A. Zisserman, "VoxCeleb: A large-scale speaker identification dataset," *arXiv:1706.08612*, pp. 1–6, 2017.