IDARTS: Interactive Differentiable Architecture Search

Song Xue[1,2]*, Runqi Wang[1]*, Baochang Zhang[1,5], Tian Wang[1,2]†, Guodong Guo[3], David Doermann[4]
[1] Beihang University, Beijing, China
[2] Jiangsu Key Laboratory of Image and Video Understanding for Social Safety,
Nanjing University of Science and Technology, Nanjing, China
[3]National Engineering Laboratory for Deep Learning Technology and Application;
Institute of Deep Learning, Baidu Research, Beijing, China
[4]University at Buffalo, USA
[5]Lobachevsky State University of Nizhni Novgorod, Nizhni Novgorod, Russian Federation
{songxue, runqiwang, bczhang, wangtian}@buaa.edu.cn,
guoguodong01@baidu.com, doermann@buffalo.edu

Abstract

Differentiable Architecture Search (DARTS) improves the efficiency of architecture search by learning the architecture and network parameters end-to-end. However, the intrinsic relationship between the architecture's parameters is neglected, leading to a sub-optimal optimization process. The reason lies in the fact that the gradient descent method used in DARTS ignores the coupling relationship of the parameters and therefore degrades the optimization. In this paper, we address this issue by formulating DARTS as a bilinear optimization problem and introducing an Interactive Differentiable Architecture Search (IDARTS). We first develop a backtracking backpropagation process, which can decouple the relationships of different kinds of parameters and train them in the same framework. The backtracking method coordinates the training of different parameters that fully explore their interaction and optimize training. We present experiments on the CIFAR10 and ImageNet datasets that demonstrate the efficacy of the IDARTS approach by achieving a top-1 accuracy of 76.52% *on ImageNet without additional search cost vs.* 75.8% *with the state-of-the-art PC-DARTS.*

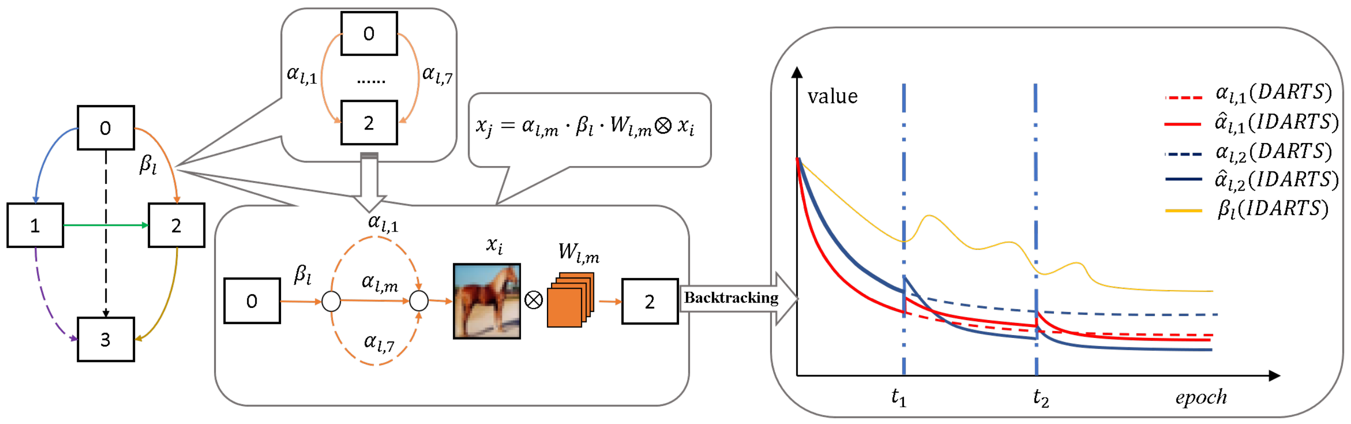
1. Introduction

The goal of Neural Architecture Search (NAS) is to automatically design neural architectures to replace traditional manual architecture design. NAS has had a significant impact on computer vision, in part by reducing this need for manual work. Recently, Liu et al. [18] proposed differentiable architecture search (DARTS) as an alternative that makes architecture search more efficient. DARTS relaxes the search space to be continuous and differentiable. DARTS learns the weight of each operation with gradient descent so that the architecture can be optimized with respect to its validation set performance by gradient descent. Despite its sophisticated design, DARTS is still subject to a large yet redundant space of network architectures and thus suffers from significant memory and computation overhead. To address the problems of DARTS, researchers have proposed alternative formulations [3, 13, 30, 2, 33, 4, 11]. Among them, PC-DARTS [30] reduces redundancy in the network space by performing a more efficient search without compromising the performance. PC-DARTS only samples a subset of channels in a super-net during the search to reduce computation and introduces edge normalization to stabilize the search for network connectivity by explicitly learning an extra set of edge-selection parameters.

However, these DARTS alternatives ignore the intrinsic relationship between different kinds of parameters, and as a result, the selected architecture is sub-optimal due to an insufficient training process. The reason lies in the fact that the coupling relationship will affect the training of the network architecture to its limit before it is selected or left out. To address this issue, we introduce a bilinear model into DARTS and develop a new backpropagation method to decouple the hidden relationships among variables to facilitate the optimization process. To the best of our knowledge, few works have formulated DARTS as a bilinear problem.

In the paper, we address these issues by formulating DARTS as a bilinear optimization problem and developing the efficient Interactive Differentiable Architecture

*Co-first author.
†Corresponding author.



(a) ***α and β*** **are coupled** (b) **Backtracking in back propagation**

Figure 1. An overview of IDARTS. (a) α and β are coupled in IDARTS. The edge and operation (β_l and α_l) are coupled during the neural architecture search. x_i and x_j represent node 0 and node 2, respectively. $x_j = \alpha_{l,m} \cdot \beta_l \cdot W_{l,m} \otimes x_i$ is specifically described in Eq. 2. (b) A backtracking method is introduced to coordinate the training of different parameters, which can fully explore their interaction during training. The dotted line results indicate that the lack of backtracking leads to the inefficient training of α, and the solid line indicates an efficient training of IDARTS.

Search (IDARTS). Fig. 1 shows the framework of IDARTS. Fig. 1(b) shows that the dotted line results are inefficient compared with IDARTS shown in the solid line. t_1 and t_2 mark the results where the architecture parameter α is backtracked. IDARTS coordinates the training of different parameters and fully explores the interaction between them based on the backtracking method. Our method allows operations to be selected only when they are sufficiently trained. We evaluate our IDARTS on image classification and conduct experiments on the CIFAR10 and ImageNet datasets. The experimental results show that IDARTS achieves superior performance compared to existing DARTS approaches [30, 33, 4]. Our contributions are summarized as follows:

- We provide the first attempt to formulate DARTS as a bilinear optimization problem. IDARTS decouples the relationship of different kinds of parameters to sufficiently train them in the same framework.
- We introduce a backtracking method to coordinate the training of different parameters. The backtracking fully explores the potential of the parameters in the architecture search through their interaction during the optimization process.
- Extensive experiments demonstrate that IDARTS achieves better performance than prior arts on the CIFAR10 and ImageNet datasets, with a top-1 accuracy of 97.68% on CIFAR10 and 76.52% on ImageNet.

2. Related works

2.1. Neural Architecture Search (NAS)

NAS is one of the most promising technologies for optimizing deep learning paradigms. Early NAS approaches focus on searching a network by either reinforcement learning [41, 40, 1, 38] or evolution [23, 28, 25]. However, most of these methods require a tremendous amount of computation and memory resources. *One-shot* architecture search methods [18, 22, 29, 37, 36] have been proposed to implement efficient architecture search, making it possible to identify an optimal architecture within a few GPU days. Liu et al. [18] proposed differentiable architecture search (DARTS) that enables NAS to use gradient descent for search. As a result, DARTS is able to identify good convolutional architectures at a fraction of the computational cost, making NAS broadly accessible. DARTS has achieved promising performance by using orders of magnitude less computation, but still has some drawbacks. Liang et al. [13] proposed DARTS+ to avoid collapse and improve the original DARTS by stopping the search procedure early when certain conditions occur. PDARTS [3] presents an efficient algorithm that attempts to overcome the depth gap issue between search and evaluation. This is accomplished by increasing the depth of the searched architectures gradually during the training procedure. ProxylessNAS [2] argues that the optimization objectives of the search and evaluation networks are inconsistent in DARTS. ProxylessNAS adopts the differentiable framework and searches for architectures on the target task instead of adopting the conventional proxy-based framework.

Recently, there have been methods proposed to improve DARTS. Fair DARTS [4] suggests that the reason for the collapse of DARTS performance lies in an unfair advantage of exclusive competition and lets each operation's architecture be weighed independent of others, which relaxes the exclusive competition to be collaborative. CDARTS [33] builds a cyclic feedback mechanism between the search and evaluation networks to enable the evolution of the topology to fit the final evaluation network. PC-DARTS [30] addresses the problem of high GPU memory cost by introducing a partially-connected strategy for network optimization. They introduce edge normalization to stabilize the search for network connectivity by explicitly learning an extra set of edge-level parameters. However, the relationship with the edge-level parameters is neglected, leading to an insufficient training process and a sub-optimal solution.

2.2. Bilinear Optimization

Bilinear optimization models are widely used in many computer vision algorithms. Often, the optimization objectives or models are influenced by two or more hidden factors that interact to produce the observations [6, 31]. Bilinear models can be embedded in CNNs [15, 14, 19, 39]. Bilinear models in CNNs can be performed by iterative methods such as the Accelerated Proximal Gradient (APG) [9] and the iterative shrinkage-thresholding algorithm (ISTA) [32, 15]. A number of deep learning applications, such as fine-grained categorization [16, 12], visual question answering (VQA) [34], and person re-identification [26], attempt to embed bilinear models into CNNs to model pairwise feature interactions and fuse multiple features with attention. To update the parameters, they directly utilize gradient descent and back-propagate the gradients of the loss.

In this paper, we formulate DARTS as a bilinear optimization problem and introduce IDARTS for an efficient search. The experimental results validate the effectiveness of our method on CIFAR10 and ImageNet without additional search costs.

3. Interactive Differentiable Architecture Search

3.1. Bilinear Models for DARTS

We first show how DARTS can be formulated as a bilinear optimization problem. Assume that there are L edges in a cell, and the edge between node N_i and node N_j is the lth edge. Following [18, 30], we take the lth edge, which is formulated as:

$$f_l(\mathbf{W}_{l,m}, \mathbf{x_i}) = \sum_{o_{l,m} \in \mathcal{O}_{(l)}} \alpha_{l,m} \cdot o_{l,m}(\mathbf{W}_{l,m} \otimes \mathbf{x_i}), \tag{1}$$

where $\mathbf{W}_{l,m}$ denotes the kernels of the mth convolution operation. We assume that there are M operations on one edge. M refers to the number of all operations. $\mathbf{x_i}$ denotes the feature map of N_i, $\mathcal{O}_l$ denotes the set of operations, and $\alpha_{l,m}$ is the parameter of operation $o_{l,m}$ on lth edge processed by softmax operation.

$$\begin{aligned} \mathbf{x_j} &= \sum_{i<j} \{\beta_l\} \cdot f_l(\mathbf{x_i}) \\ &= \sum_{i<j} \sum_{o_{l,m} \in \mathcal{O}_l} \beta_l \alpha_{l,m} \cdot o_{l,m}(\mathbf{W_{l,m}} \otimes \mathbf{x_i}), \end{aligned} \tag{2}$$

where β_l denotes the parameter of lth edge. The softmax is defined on β and α to calculate the final architecture. For each intermediate node, we will choose two edges, which are jointly determined by α and β. In Fig. 1, we see α and β are coupled in the inference process as shown in Eq. 2. $\mathbf{x_j}$ is linearly dependent on both α and β. If an improper operation is selected, it will affect the selection of the edge and vice versa. It suggests that we should consider their relationship for better optimization. A basic bilinear optimization problem attempts to optimize the following objective function in the architecture search:

$$\arg\min_{\beta,\alpha} G(\mathbf{W}, \beta, \alpha) = \arg\min_{\beta,\alpha} (\mathcal{L}(\mathbf{W}, \beta, \alpha) + R(\beta)), \tag{3}$$

where $\alpha \in \mathbb{R}^{L \times M}$ and $\beta \in \mathbb{R}^{L \times 1}$ are variables to be optimized, L is the number of edges, M is the number of operations at each edge, and $R(\cdot)$ represents the constraint about backtracking. $\mathcal{L}(\cdot)$ denotes the loss function in the original DARTS models.

Following [18, 30], the weights of the kernels $\mathbf{W}$ and the architectural parameters α, β are optimized sequentially. The learning procedure for the architectural parameters involves an optimization as:

$$\begin{aligned} \mathbf{W^{t+1}} &= \arg\min_{\mathbf{W}} \mathcal{L}_{train}(\mathbf{W^t}, \alpha^t, \beta^t), \\ \alpha^{t+1} &= \arg\min_{\alpha} \mathcal{L}_{val}(\mathbf{W^{t+1}}, \alpha^t, \beta^t), \\ \beta^{t+1} &= \arg\min_{\beta} \mathcal{L}_{val}(\mathbf{W^{t+1}}, \alpha^t, \beta^t), \end{aligned} \tag{4}$$

where α^{t+1} and β^{t+1} denote the parameters of operation and edge in the $(t+1)$th step, and $\mathbf{W^{t+1}}$ denotes the kernel of the convolution at the $(t+1)$th step.

In Eq. 4, α and β are updated independently. However, it is improper to optimize α and β independently due to their coupling relationship. We consider the search process of differentiable architecture search as a bilinear optimization problem and solve the problem using a new backtracking method. The details will be shown in Section 3.3.

3.2. Search Space

By simplifying the architecture search to find the best cell structure, cell-based NAS methods try to learn a scalable and transferable architecture. Following [18, 30], we

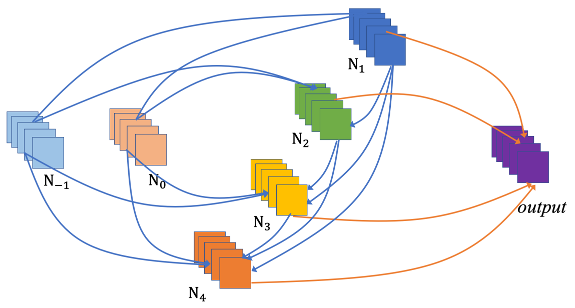


Figure 2. A cell contains seven nodes, which are two input nodes N_{-1} and N_0, four intermediate nodes N_1, N_2, N_3, N_4, and one output node.

search for normal and reduction computation cells to build the final architecture. The reduction cells are located at $1/3$ and $2/3$ of the total depth of the network, and the rest of the cells are normal cells. A normal cell uses operations with a stride of 1 to keep the size of the input feature map unchanged. The number of output channels is identical to the number of input channels. A reduction cell uses operations with a stride of 2 to reduce the spatial resolution of feature maps, and the number of output channels is twice the number of input channels. The set of operations include 3×3 and 5×5 separable convolution, 3×3 and 5×5 dilated separable convolution, 3×3 max pooling, 3×3 average pooling, a zero(none), and a skip connection. A cell (Fig. 2) is a fully-connected directed acyclic graph (DAG) of 7 nodes. Each $\mathbf{x_i}$ is a latent representation (e.g., a feature map in convolutional networks). Each directed edge (i, j) between node N_i and node N_j denotes the set of operations $\mathcal{O}_l = \{o_{l,1}, ..., o_{l,M}\}$. Following [18], there are 2 input nodes, 4 intermediate nodes, 1 output node, and 14 edges per cell during the search. Each cell takes the outputs of the two previous cells as the input. The output node of a cell is the depth-wise concatenation of all of the intermediate nodes.

3.3. Backtracking Backpropagation

We consider the problem from a new perspective where the β and α are coupled in Eq. 3. We note that the calculation of the derivative of α should consider its coupling relationship with β. Based on the chain rule [21] and its notations, we have:

$$\hat{\alpha}^{t+1} = \alpha^t + \eta_1(-\frac{\partial G(\alpha)}{\partial \alpha} + \eta_2 Tr((\frac{\partial G(\beta)}{\partial \beta})^T \frac{\partial \beta}{\partial \alpha})), \quad (5)$$

where η_1 represents the learning rate, η_2 represents the coefficient of backtracking, $\hat{\alpha}_{t+1}$ denotes the value backtracked from α_{t+1}. and $Tr(\cdot)$ represents the trace of the matrix. Here, $\mathbf{W}$ is omitted for simplicity, and only structure parameters α, β are considered during the backpropagation process. We further define:

$$\hat{G} = (\frac{\partial G(\beta)}{\partial \beta})^T / \alpha, \quad (6)$$

where $\hat{G}$ is defined by considering the bilinear optimization problem as in Eq. 3. Note that $R(\cdot)$ is only considered when backtracking. Then we have:

$$\frac{\partial G(\beta)}{\partial \alpha} = Tr[\alpha \hat{G} \frac{\partial \beta}{\partial \alpha}]. \quad (7)$$

We denote $\hat{G} = [\hat{g}_1, ..., \hat{g}_L]$. Assuming that β_l and α_m are independent when $l \neq m$, α_m denotes a column vector, and $\alpha_{1,m}$ denotes an element in matrix α, we have:

$$\frac{\partial \beta}{\partial \alpha} = \begin{bmatrix} 0 & ... & \frac{\partial \beta_m}{\partial \alpha_{1,m}} & ... & 0 \\ . & & . & & . \\ . & & . & & . \\ . & & . & & . \\ 0 & ... & \frac{\partial \beta_m}{\partial \alpha_{L,m}} & ... & 0 \end{bmatrix}, \quad (8)$$

and

$$\alpha \hat{G} = \begin{bmatrix} \alpha_1 \hat{g}_1 & ... & \alpha_1 \hat{g}_l & ... & \alpha_1 \hat{g}_L \\ . & & . & & . \\ . & & . & & . \\ . & & . & & . \\ \alpha_L \hat{g}_1 & ... & \alpha_L \hat{g}_l & ... & \alpha_L \hat{g}_L \end{bmatrix}. \quad (9)$$

We combine Eq. 8 and Eq. 9, and get:

$$\alpha \hat{G} \frac{\partial \beta}{\partial \alpha} = \begin{bmatrix} 0 & ... & \alpha_1 \sum_{l=1}^{L} \hat{g}_l \frac{\partial \beta_m}{\partial \alpha_{l,m}} & ... & 0 \\ . & & . & & . \\ . & & . & & . \\ . & & . & & . \\ 0 & ... & \alpha_L \sum_{l=1}^{L} \hat{g}_l \frac{\partial \beta_m}{\partial \alpha_{l,m}} & ... & 0 \end{bmatrix}. \quad (10)$$

After that, the trace of Eq. 5 is then calculated by:

$$Tr[\alpha^t \hat{G} \frac{\partial \beta}{\partial \alpha_m}] = \alpha_m \sum_{l=1}^{L} \hat{g}_l \frac{\partial \beta_m}{\partial \alpha_{l,m}}. \quad (11)$$

Remembering that $\alpha^{t+1} = \alpha^t - \eta_1 \frac{\partial G(\alpha)}{\partial \alpha}$, IDARTS combines Eq. 5 and Eq. 11:

$$\begin{aligned} \hat{\alpha}^{t+1} &= \alpha^{t+1} + \eta \begin{bmatrix} \sum_{l=1}^{L} \hat{g}_l \frac{\partial \beta_1}{\partial \alpha_{l,1}} \\ . \\ . \\ . \\ \sum_{l=1}^{L} \hat{g}_l \frac{\partial \beta_L}{\partial \alpha_{l,L}} \end{bmatrix} \odot \begin{bmatrix} \alpha_1 \\ . \\ . \\ . \\ \alpha_L \end{bmatrix} \\ &= \alpha^{t+1} + \eta \begin{bmatrix} < \hat{G}, \frac{\partial \beta_1}{\partial \alpha_1} > \\ . \\ . \\ . \\ < \hat{G}, \frac{\partial \beta_L}{\partial \alpha_L} > \end{bmatrix} \odot \begin{bmatrix} \alpha_1 \\ . \\ . \\ . \\ \alpha_L \end{bmatrix} \\ &= \alpha^{t+1} + \eta \gamma \odot \alpha^t, \end{aligned} \quad (12)$$

Algorithm 1: IDARTS Interactive Differentiable Architecture Search

```
Input: Training data, validation data, searching
       hyper-graph, hyper-parameters $K = 0$, $T = 25$,
       $S = 50$;
Create architectural parameters $\alpha = \alpha_l$, edge level
  parameters $\beta = \beta_l$ and supernet weights $W$
Create a mixed operation $o_l$ parameterized by $\alpha_l$ and $\beta_l$
  for each edge $l$;
Output: The structure;
Search for an architecture for S epochs;
while ($K \leq S$) do
    Update parameters $\alpha$ and $\beta$;
    if ($K \geq T$) then
        According to Eq. 13, we select $\alpha$ that should be
          backtracked;
        backtracking $\alpha$ by Eq. 12;
    end
    Update weights $W$;
    $K \leftarrow K + 1$;
    Find the final architecture based on the learned $\alpha$ and
      $\beta$;
end
```

where $\odot$ represents the Hadamard product and $\eta = \eta_1\eta_2$. To simplify the calculation, $\frac{\partial\beta}{\partial\alpha}$ can be approximated by $\frac{\Delta\beta}{\Delta\alpha}$. Eq. 12 shows our method is actually based on a projection function to solve the coupling problem of the bilinear optimization by γ. In this method, we consider the influence of α^t and backtrack the optimized state at the $(t+1)th$ step to form $\hat{\alpha}^{t+1}$. We first decide when the optimization should be backtracked, and the update rule of the proposed IDARTS is defined as:

$$\hat{\alpha}^{t+1} = \begin{cases} P(\alpha^{t+1}, \alpha^t) & if\ R(\beta) < \zeta, \\ \alpha^{t+1} & otherwise, \end{cases} \tag{13}$$

where $P(\alpha^{t+1}, \alpha^t) = \alpha^{t+1} + \eta\gamma \odot \alpha^t$. $R(\beta)$ represents the ranking of $|\beta_l|$ and ζ represents the threshold. We then have:

$$\zeta = \lfloor (S - T) \cdot \lambda \cdot L \rfloor, \tag{14}$$

where T and S denote the beginning and ending epoch of backtracking, λ denotes the coefficient, and L denotes the number of edges in a cell. As shown in 14, ζ will be increased during searching. By doing so, α will be backtracked, according to β.

4. Experiments

We use our IDARTS to automatically find CNN architectures. The CNN architectures discovered by IDARTS outperform the state-of-the-art (SOTA) on image classification on the ImageNet dataset [5].

4.1. Datasets

CIFAR10 [10] is a small popular dataset containing 60K images. 50K are used for the training set, and the remaining 10K are used for the test set. The images belong to 10 different categories and have a resolution of 32×32.

ImageNet is currently the world's largest image recognition database. It consists of 1,000 categories with 1.2M training images and 50K validation images. We follow the general setting on the ImageNet dataset where the images are resized to 224×224 for training and testing.

4.2. Search and Training Settings

In our experiments, we search for architectures with an over-parameterized network on CIFAR10 and ImageNet and then evaluate on the corresponding datasets. Following DARTS [18] as well as conventional architecture search approaches, we use an individual stage for architecture search, and after the optimal architecture is obtained, we retrain the network.

Search and Training Settings on CIFAR10. As is common practice, we first search for normal cells and reduction cells with a small network for image classification on CIFAR10. In the search process, the over-parameterized network is constructed with eight cells, where the $3rd$ and $6th$ cells are reduction cells. The initial number of channels is 16. We employ the SGD optimizer with an initial learning rate of 0.1, a momentum of 0.9, a weight decay of 3×10^{-4}, and a gradient clipping at 5. We use an Adam optimizer for α and β, with a fixed learning rate of $\eta_1 = 6 \times 10^{-4}$, a momentum of (0.5, 0.999), and a weight decay of 10^{-3}. We set the weight of the backtracking to $\eta_2 = 0.04$ and the hyper-parameter $\lambda = 0.015$. We train 50 epochs using 50% of the training set as the training data in the search phase and 50% of the training set as the validation data. Following [3, 30], we freeze the parameters, α and β, and only allow the network parameters to be tuned in the first 15 epochs. We then start the training of α, β, and the network parameters. When the training reaches $T = 25$ epochs, we start backtracking to coordinate the training of different parameters and to fully explore their interaction. After the search, we train the final architecture for 600 epochs on CIFAR10. We set the total number of cell layers to 20, the batch size to 128, the initial learning rate to 0.025, and the initial number of channels to 36. The rest of the settings are the same as in the search phase.

Search and Training Settings on ImageNet. For ImageNet, we search and evaluate using the same settings provided in [30]. The optimization method and search strategy are the same as for our experiments on CIFAR10. Following [30], the over-parameterized network starts with three convolution layers of stride 2 to reduce the input image resolution from 224×224 to 28×28. We stack 6 normal cells and 2 reduction cells to form a network. To reduce the

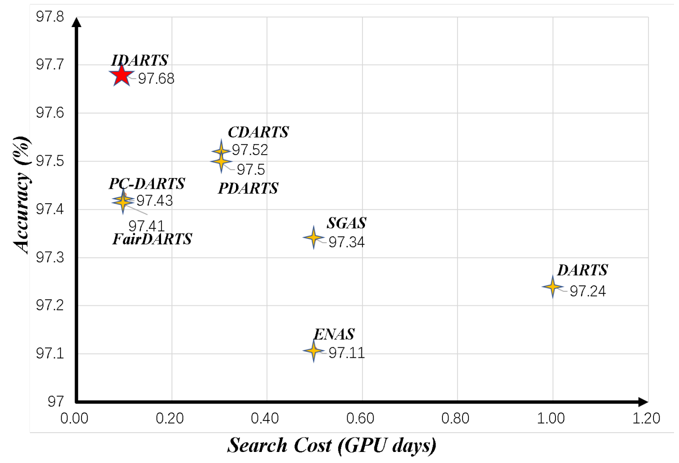


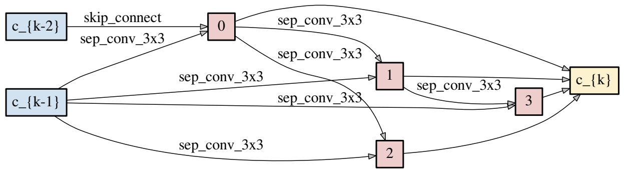
Figure 3. Comparison with SOTA architectures on CIFAR10

search time and improve the search efficiency, we sample 10% of the data in the ImageNet dataset for training network weights and 2.5% of the data for updating the hyper-parameters. We perform architecture search for 50 epochs with a batch size of 1024. The architecture parameters, α and β, are frozen during the first 35 epochs. We then start the training α, β, and the network parameters. When the training reaches $T = 40$ epochs, we start backtracking. We use a momentum SGD with an initial learning rate of 0.5, a momentum of 0.9, and a weight decay of 3×10^{-5}. For architecture parameters α and β, we use the Adam optimizer with an initial learning rate $\eta_1 = 6 \times 10^{-3}$, momentum (0.5, 0.999), and weight decay 10^{-3}. We set the weight of the backtracking to $\eta_2 = 0.04$ and the hyper-parameter $\lambda = 0.05$. After the search, we built a large network with 14 cells and 48 initial channels and train for 250 epochs with a batch size of 1024.

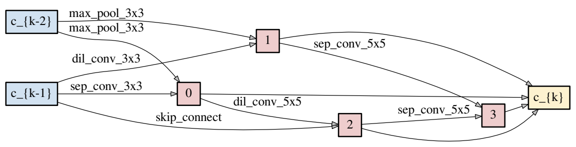
4.3. Results

4.3.1 Results on CIFAR10

On CIFAR10, we use a single NVIDIA Titan V GPU to search for approximately 0.1 GPU days. The results and a comparison to recent approaches are summarized in Table 1. We observe that our IDARTS achieves superior performance compared to some other manually or automatically designed CNNs. For instance, IDARTS surpasses the manually designed DenseNet-BC [8] by 1.14% (96.54% vs. 97.68%). It is worth noting that the performance of ProxylessNAS[2] is slightly better than our IDARTS, but its search time is 40× larger that of IDARTS. Compared with PC-DARTS[30], IDARTS achieves better performance (97.43% vs. 97.68%) with a similar search time. Compared with CDARTS[33], IDARTS achieves not only a better performance(97.52% vs. 97.68%), but also has a faster search speed (0.3 vs. 0.1 GPU days). We show the performance of IDARTS and other advanced NAS methods in Fig. 3. The red star represents IDARTS, and the remainder represent other advanced search methods. We clearly see that IDARTS has the highest accuracy and the shortest search time among all search methods other than ProxylessNAS. Fig. 4 shows a detailed representation of the best cells discovered. We observe that the network prefers to choose separable convolutions [7] in the normal cells as it is a key component of network construction and can increase the size of the model.



(a) normal



(b) reduction

Figure 4. The detailed structure of the best cells discovered on CIFAR10. (a) The normal cell found on CIFAR10. (b) The reduction cell found on CIFAR10.

4.3.2 Results on ImageNet

On ImageNet, we use eight Tesla V100 GPUs for search, and the total batch size is 1024. The entire search process takes around 11.5 hours. Fig. 6 shows the detailed structure of the best cells discovered. The evaluation results of the searched architectures are reported in Table 2. The architectures searched on CIFAR10 and ImageNet itself are both evaluated. IDARTS (CIFAR10) achieved 76.14% top-1 accuracy, which demonstrates the generalization potential of the IDARTS approach. IDARTS (ImageNet) achieved 76.52% top-1 accuracy. We compare our models with the SOTA architectures designed manually and models obtained with other NAS methods. The architectures discovered by our IDARTS are better than the human-designed ShuffleNet 2x (v2) [35] (74.9% vs. 76.52%). IDARTS surpasses PC-DARTS[30] by 0.72% (75.8% vs. 76.52%) which demonstrates the effectiveness of our method. We show the IDARTS method and other advanced methods in Fig. 5. The blue stars represent search methods that use the architecture searched on CIFAR10, and the red and yel-

Architecture	Accuracy (%)	# Params (M)	Search Cost (GPU days)	Search Method
DenseNet-BC [8]	96.54	25.6	-	Manual
NASNet-A [41]	97.35	3.3	1800	RL
AmoebaNet-A [24]	96.66 ± 0.06	3.2	3150	evolution
AmoebaNet-B [24]	97.45 ± 0.05	2.8	3150	evolution
PNAS [17]	96.59 ± 0.09	3.2	225	SMBO
ENAS [22]	97.11	4.6	0.5	RL
DARTS (1st order) [18]	97.00 ± 0.14	3.3	0.4	gradient
DARTS (2nd order) [18]	97.24 ± 0.09	3.3	1	gradient
SNAS (mild) [29]	97.02	2.9	1.5	gradient
ProxylessNAS [2]	97.92	3.27	4	gradient
P-DARTS [3]	97.5	3.27	0.3	gradient
SGAS [11]	97.34 ± 0.24	3.7	0.5	gradient
FairDARTS [4]	97.41 ± 0.14	3.8	0.1	gradient
PC-DARTS [30]	97.43 ± 0.07	3.27	0.1	gradient
CDARTS [33]	97.52 ± 0.04	3.8	0.3	gradient
IDARTS	**97.68**	**4.159**	**0.1**	gradient

Table 1. Comparison with state-of-the-art architectures on CIFAR10.

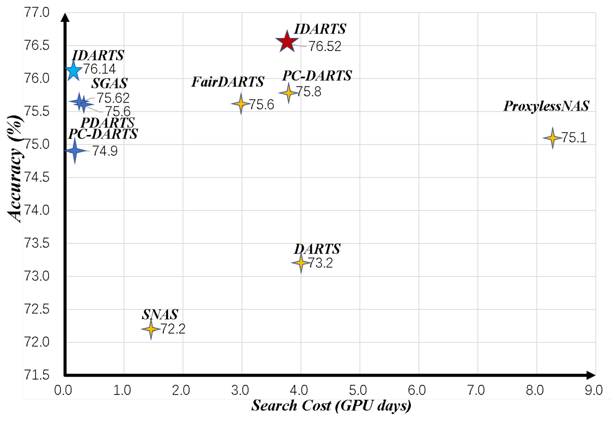
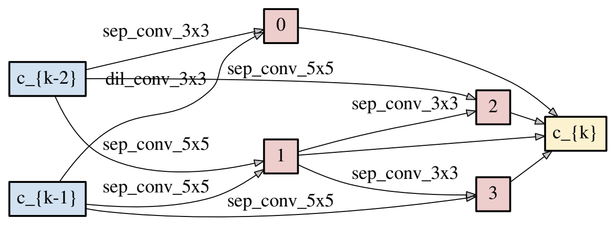
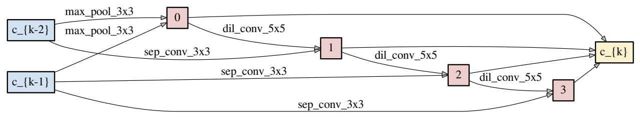


Figure 5. Comparison with SOTA architectures on ImageNet



(a) normal



(b) reduction

Figure 6. The detailed structure of the best cells discovered on ImageNet. (a) The normal cell found on ImageNet. (b) The reduction cell found on ImageNet.

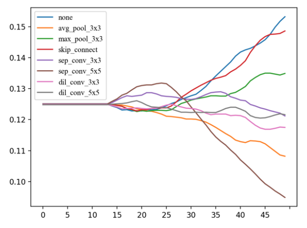
low stars represent the methods that search directly on ImageNet. We can clearly see that IDARTS (CIFAR10) has the highest accuracy and the shortest search time, compared with these advanced methods in blue stars.

4.4. Ablation Study

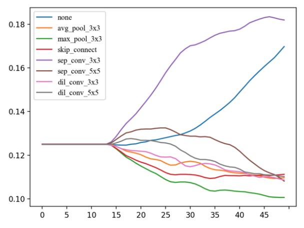
Fig. 7 illustrates the comparison of α for IDARTS and PC-DARTS in the shallowest edge. The label of x-axis is epoch in searching, the label of y-axis is value of α. We freeze the hyper-parameters, α and β, in the first 15 epochs (only network parameters are updated), α remains unchanged. As the shortage of interaction between α and β in PC-DARTS, α and β might easily fall into the local minima. However, we backtrack the insufficiently trained operations on this edge to escape from the local minimal to select a better operation and thus a better architecture by considering the intrinsic relationship between α and β. Due to the backtracking of α, the competition between different operations is intensified in the IDARTS search process as shown in Fig. 7. As a result, it is more conducive to choose the most valuable operation than PC-DARTS. In Fig. 8, the label of y-axis is $\mathcal{L}_{val}$. We also show that the convergence of IDARTS is similar to that of PC-DARTS. Although the two have the same convergence rate, we can clearly see that the final loss of IDARTS converges to a smaller value. The main reason is that IDARTS has explored the relation-

Architecture	**Accuracy(%)**		**# Params**	$+\times$	**Search Cost**	**Search**
	top-1	top-5	**(M)**	**(M)**	**(GPU days)**	**Method**
Inception-v1 [27]	69.8	89.9	6.6	1448	-	Manual
MobileNet [7]	70.6	89.5	4.2	569	-	Manual
ShuffleNet 2x (v1) [35]	73.6	89.8	~ 5	524	-	Manual
ShuffleNet 2x (v2) [20]	74.9	-	~ 5	591	-	Manual
NASNet-A [41]	74.0	91.6	5.3	564	1800	RL
AmoebaNet-C [24]	75.7	92.4	6.4	570	3150	evolution
PNAS [17]	74.2	91.2	5.1	588	225	SMBO
DARTS (2nd order) [18]	73.2	91.3	4.7	574	4.0	gradient
SNAS (mild) [29]	72.7	90.8	4.3	522	1.5	gradient
P-DARTS [3]	75.6	92.6	4.9	557	0.3	gradient
PC-DARTS [30]	74.9	92.2	5.3	586	0.1	gradient
SGAS [11]	75.9	92.7	5.4	598	0.25	gradient
IDARTS(CIFAR10)	**76.14**	**92.87**	**5.81**	**657**	**0.1**	gradient
ProxylessNAS [2]	75.1	92.5	7.1	465	8.3	gradient
PC-DARTS [30]	75.8	92.7	5.3	597	3.8	gradient
FairDARTS [4]	75.6	92.6	4.3	440	3	gradient
IDARTS(ImageNet)	**76.52**	**93.00**	**6.18**	**714**	**3.8**	gradient

Table 2. A comparison with state-of-the-art architectures on ImageNet. IDARTS(CIFAR10) means the architecture was searched on CIFAR10. IDARTS(ImageNet) means the architecture was searched on ImageNet directly.



(a) IDARTS



(b) PC-DARTS

Figure 7. Comparison of α values in the shallowest edge of IDARTS and PC-DARTS on CIFAR10

.

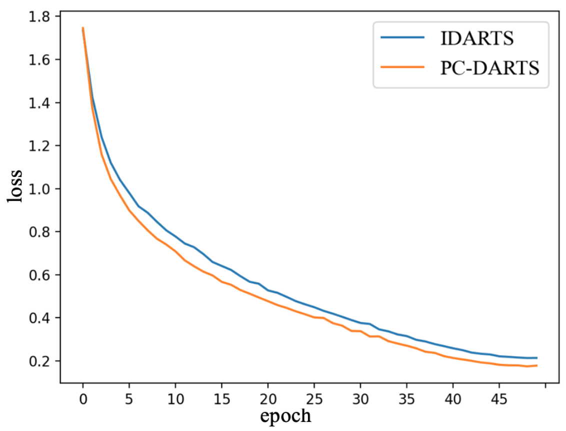


Figure 8. Comparison of searching loss on CIFAR10 with IDARTS and PC-DARTS

ship between different parameters and used our backtracking method to fully train the architecture parameter α. We theoretically derive our method under the framework of gradient descent, which provides a solid foundation for the convergence analysis of our method.

5. Conclusion

In this work, we proposed the IDARTS method for an efficient architecture search, motivated by Bilinear Models. IDARTS trains architecture parameters by decoupling the relationship between different types of parameters and using the backtracking method to coordinate the training of different parameters. The decoupling allows IDARTS to search for the best network structure. Experiments demonstrate the efficacy of the proposed algorithm and show that IDARTS achieves state-of-the-art performance on the ImageNet dataset.

Acknowledgment

This study was supported by the National Key Research and Development Program of China (2018AAA0101400), National Natural Science Foundation of China (62076016, 61972016).

References

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *Proc. of ICLR*, 2017.

[2] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *Proc. of ICLR*, 2019.

[3] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proc. of ICCV*, 2019.

[4] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. In *Proc. of ECCV*, 2020.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. of CVPR*, 2009.

[6] Felix Heide, Wolfgang Heidrich, and Gordon Wetzstein. Fast and flexible convolutional sparse coding. In *Proc. of CVPR*, 2015.

[7] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv*, 2017.

[8] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proc. of CVPR*, 2017.

[9] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proc. of ECCV*, 2018.

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[11] Guohao Li, Guocheng Qian, Itzel C Delgadillo, Matthias Muller, Ali Thabet, and Bernard Ghanem. Sgas: Sequential greedy architecture search. In *Proc. of CVPR*, 2020.

[12] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Factorized bilinear models for image recognition. In *Proc. of ICCV*, 2017.

[13] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. DARTS+: improved differentiable architecture search with early stopping. *arXiv*, 2019.

[14] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Shao Ling. Hrank: Filter pruning using high-rank feature map. In *Proc. of CVPR*, 2020.

[15] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proc. of CVPR*, 2019.

[16] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *Proc. of ICCV*, 2015.

[17] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proc. of ECCV*, 2018.

[18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *Proc. of ICLR*, 2019.

[19] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proc. of ICCV*, 2017.

[20] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proc. of ECCV*, 2018.

[21] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008.

[22] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *Proc. of ICLM*, 2018.

[23] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proc. of AAAI*, 2019.

[24] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proc. of AAAI*, 2019.

[25] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proc. of ICML*, 2017.

[26] Yumin Suh, Jingdong Wang, Siyu Tang, Tao Mei, and Kyoung Mu Lee. Part-aligned bilinear representations for person re-identification. In *Proc. of ECCV*, 2018.

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc. of CVPR*, 2015.

[28] Lingxi Xie and Alan Yuille. Genetic cnn. In *Proc. of ICCV*, 2017.

[29] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. In *Proc. of ICLR*, 2019.

[30] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *Proc. of ICLR*, 2020.

[31] Linlin Yang, Ce Li, Jungong Han, Chen Chen, Qixiang Ye, Baochang Zhang, Xianbin Cao, and Wanquan Liu. Image reconstruction via manifold constrained convolutional sparse coding for image sets. *JSTSP*, 11(7):1072–1081, 2017.

[32] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *Proc. of ICLR*, 2018.

[33] Hongyuan Yu and Houwen Peng. Cyclic differentiable architecture search. *arXiv*, 2020.

[34] Zhou Yu, Jun Yu, Jianping Fan, and Dacheng Tao. Multi-modal factorized bilinear pooling with co-attention learning for visual question answering. In *Proc. of ICCV*, 2017.

[35] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proc. of CVPR*, 2018.

[36] Xiawu Zheng, Rongrong Ji, Lang Tang, Yan Wan, Baochang Zhang, Yongjian Wu, Yunsheng Wu, and Ling Shao. Dynamic distribution pruning for efficient network architecture search. *arXiv*, 2019.

[37] Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. Multinomial distribution learning for effective neural architecture search. In *Proc. of ICCV*, 2019.

[38] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proc. of CVPR*, 2018.

[39] Li'an Zhuo, Baochang Zhang, Linlin Yang, Hanlin Chen, Qixiang Ye, David Doermann, Rongrong Ji, and Guodong Guo. Cogradient descent for bilinear optimization. In *Proc. of CVPR*, 2020.

[40] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[41] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proc. of CVPR*, 2018.