Model Compression Based on Differentiable Network Channel Pruning

Yu-Jie Zheng, Si-Bao Chen, Chris H. Q. Ding, and Bin Luo, *Senior Member, IEEE*

***Abstract*— Although neural networks have achieved great success in various fields, applications on mobile devices are limited by the computational and storage costs required for large models. The model compression (neural network pruning) technology can significantly reduce network parameters and improve computational efficiency. In this article, we propose a differentiable network channel pruning (DNCP) method for model compression. Unlike existing methods that require sampling and evaluation of a large number of substructures, our method can efficiently search for optimal substructure that meets resource constraints (e.g., FLOPs) through gradient descent. Specifically, we assign a learnable probability to each possible number of channels in each layer of the network, relax the selection of a particular number of channels to a softmax over all possible numbers of channels, and optimize the learnable probability in an end-to-end manner through gradient descent. After the network parameters are optimized, we prune the network according to the learnable probability to obtain the optimal substructure. To demonstrate the effectiveness and efficiency of DNCP, experiments are conducted with ResNet and MobileNet V2 on CIFAR, Tiny ImageNet, and ImageNet datasets.**

***Index Terms*— Channel pruning, convolutional neural network, differentiable method, model compression, neural network pruning.**

I. Introduction

IN THE application of computer vision including image classification [1]–[4], object detection [5]–[8], semantic segmentation [9]–[12] and so on, deep neural networks have achieved great success. However, these deep neural networks require huge computing and storage costs yet, which severely hinder their applications under resource constraints. There are many model compression methods, such as pruning [13]–[26], quantization [27]–[29], lightweight network design [30]–[33], and so on. Among them, pruning is a widely recognized efficient network compression and acceleration method, which significantly improves the inference efficiency of the network by subtracting unimportant weights or channels.

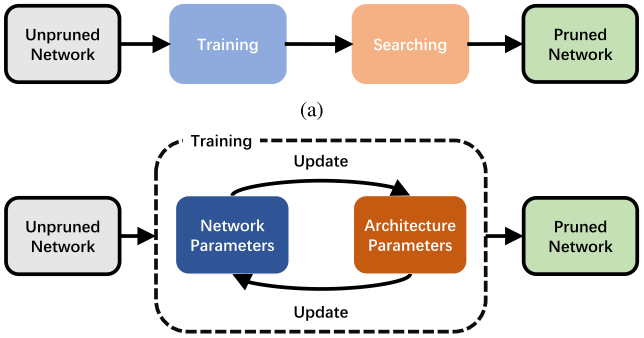

Fig. 1. Comparison between (a) existing AutoML-based channel pruning methods and (b) our proposed DNCP method.

Traditional channel pruning methods [13]–[18] can be mainly divided into three steps: first train a large model with redundant parameters, then prune the unimportant channels and finally fine-tune the pruned model. The second step is the key to network channel pruning methods. The previous methods are mostly based on manually designed measurement methods or sparse training of the network to complete the channel pruning.

Recent work [34] has proposed a new understanding of channel pruning. It is believed that the network architecture obtained after pruning is the key to determining network performance compared to the weight inherited from unpruned networks and compared with fine-tuning, training the pruned model from scratch can achieve similar or even better accuracy. This shows that channel pruning can be regarded as a kind of network architecture search. Based on this view, the existing AutoML-based methods as indicated in Fig. 1(a), first train the unpruned network, and then directly use the reinforcement learning algorithm [21], evolutionary algorithm [23], or greedy algorithm [22] to search for the optimal substructure from the unpruned network. However, these methods have a disadvantage that the search space of the network channel architecture is so huge that they need to sample and evaluate a large number of substructures during the search phase.

DARTS [35] proposed a differentiable approach to solve a similar problem in the field of network architecture search.

Manuscript received March 22, 2021; revised August 23, 2021 and November 26, 2021; accepted March 31, 2022. This work was supported in part by the NSFC Key Project of International (Regional) Cooperation and Exchanges under Grant 61860206004, in part by the National Natural Science Foundation of China under Grant 61976004, and in part by the University Synergy Innovation Program of Anhui Province under Grant GXXT-2019-025. *(Corresponding author: Si-Bao Chen.)*

Yu-Jie Zheng, Si-Bao Chen, and Bin Luo are with the IMIS Laboratory of Anhui Province, the Anhui Provincial Key Laboratory of Multimodal Cognitive Computation, the MOE Key Laboratory of Intelligent Computing and Signal Processing (ICSP), and the School of Computer Science and Technology, Anhui University, Hefei 230601, China (e-mail: 1064926284@qq.com; sbchen@ahu.edu.cn; luobin@ahu.edu.cn).

Chris H. Q. Ding is with the Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX 76019 USA (e-mail: chqding@uta.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2022.3165123.

Digital Object Identifier 10.1109/TNNLS.2022.3165123

They relaxed the choice of a particular operation to the result of the softmax of all possible operations to make the search space continuous. So that the network architecture can be optimized through gradient descent. Inspired by their work, we define the search space of network channel pruning as all possible numbers of channels for each layer, and use a differentiable method to search for the optimal number of channels.

In this article, we propose a differentiable network channel pruning (DNCP) method as explained in Fig. 1(b) that can efficiently search for the optimal substructure that meets the resource constraints (e.g., FLOPs). Our method relaxes the choice of a particular number of channels in each layer of the network to the result of a softmax over all possible numbers of channels to make the search space continuous and searches for the optimal number of channels through gradient descent. Specifically, we use a set of learnable probabilities to represent the distribution of all possible numbers of channels in each layer of the network. Then we calculate the probability of each channel being retained based on these learnable probabilities. In the network feedforward, the output of each channel is multiplied by the corresponding probability as the actual output. In this way, the learnable probability can be optimized in an end-to-end manner through gradient descent with respect to classification loss and computational cost loss. After the network parameters are optimized, we perform channel pruning according to the learnable probabilities. Inspired by previous work [34], after getting the pruned substructure, we train it from scratch to get the final pruned model.

II. Related Work

A. Network Pruning

Network pruning can significantly reduce network parameters and speed up network reasoning. Traditional pruning can be roughly divided into three categories according to their realization methods. Metric-based pruning usually calculates a value that measures the importance of neurons based on the designed metric, and then cuts out unimportant neurons. The values used for measurement mainly include weight [13], activation [36] and gradient [15]. Pruning based on reconstruction error [14], [17] mainly determines which channels or filters are pruned by minimizing the reconstruction error of feature output. Pruning based on sparse training [16], [37], [38] mainly combines various regularities in training to make the network weights sparse, and then the weights close to 0 will be pruned.

The traditional network pruning algorithm has some obvious drawbacks. First, it needs to rely on a lot of experience to design appropriate metrics, and secondly, it needs to go through a lot of attempts to set an appropriate compression ratio. Different from the traditional pruning method, our method can automatically search for the optimal subnetwork through gradient descent.

B. AutoML

In order to solve the problem of manually setting the compression ratio in the traditional method, some pruning methods based on AutoML gradually appeared to automatically search for the optimal substructure. AMC [21] used reinforcement learning algorithms to transform the pruning process into a serialization decision that determines the compression ratio of each layer in turn. NetAdapt [39] proposed a progressive network pruning algorithm that progressively compresses each layer of the network to produce a final model. MetaPruning [23] introduced meta-learning and evolutionary algorithm into pruning. It trained a PruningNet to predict the weights of the network and then used evolutionary algorithm to search subnetwork. AutoSlim [22] first trained a slimmable network, and then gradually pruned the network in a greedy way.

DMCP [25] models the pruning process as a Markov model. The architecture parameters are introduced to calculate the retention probability of each group of channels in each layer of the network. Then the architecture parameters are optimized through gradient descent. Finally, the pruning is completed according to the probability distribution, eliminating the time overhead of sampling and evaluating a large number of substructures. By comparison, inspired by DARTS [35], which proposed a differentiable search algorithm that uses gradient descent to optimize architecture parameters, our method defines the search space of each layer of the network as the number of channels, and then introduces a set of parameters to express the probability that the network layer retains different numbers of channels. Our method relaxes the selection of the number of specific channels in each layer of the network to the softmax output of all possible channel numbers, which makes the search space continuous, and searches for the optimal number of channels through gradient descent. Compared with DMCP [25], the advantage of our method is that when training network parameters, we can sample higher probability network substructures for training according to the probability of each layer of the network retaining a different number of channels, so that higher probability substructures can be trained thoroughly to find better substructures.

C. Neural Architecture Search

Zoph and Le [40] first proposed the use of reinforcement learning for network architecture search. Pham *et al.* [41] proposed the use of weight sharing to accelerate the search of network architecture, that is, the parameters of the subgraph are directly inherited from the DAG, without training the subgraph from scratch. DARTS [35] proposed a differentiable search algorithm that uses gradient descent to optimize architecture parameters. The previous methods all have a drawback. They can only search for the network architecture on CIFAR10 and transfer to large-scale datasets such as ImageNet. ProxylessNAS [42] directly searched for the optimal network architecture on the target hardware on the target dataset, and at the same time added the hardware latency loss item to the objective function as a multi-target NAS. The search space of network architecture search is a set of predefined operations, and the search space of channel pruning is the number of channels in each layer of the network. Similarly, the selection of the number of channels is a discrete operation.

Inspired by the methods in [35] and [42], we use relaxation to make the search space of channel pruning continuous,

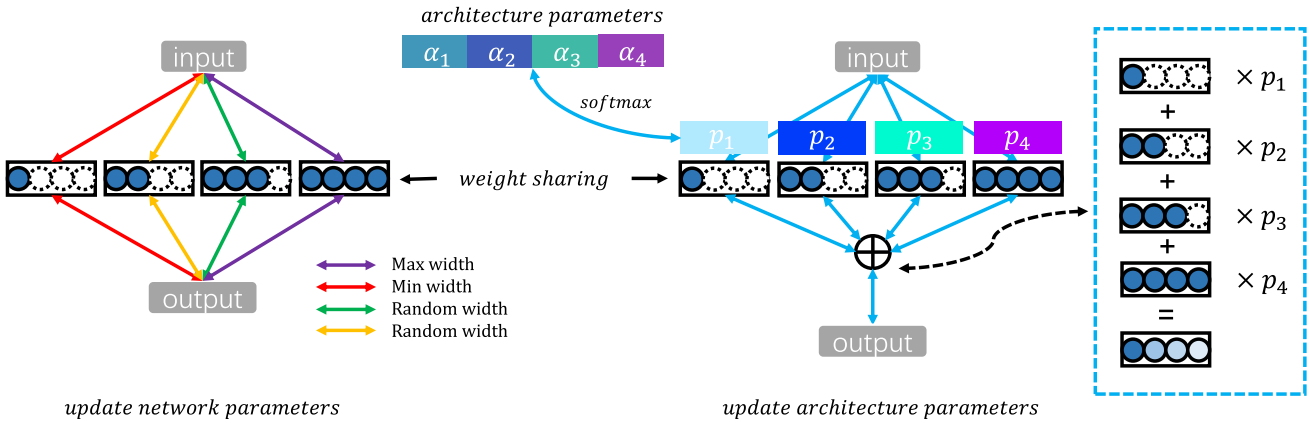

Fig. 2. Illustration of our DNCP method. During the training process, we alternately update network parameters and architecture parameters. When updating network parameters, we sample four subnetworks, which are max-width network that retains all channels, min width network that retains the fewest channels, and two random width networks that are randomly sampled according to the architecture parameters. Four subnetworks share weights and are forwarded independently. The gradients of the four subnetworks are accumulated to update the network parameters. When updating architecture parameters, they participate in the feedforward of the unpruned network and are updated through gradient descent.

then add network cost loss to the loss function, and use gradient descent to efficiently search for subnetwork that meets resource constraints.

III. Proposed DNCP Method

In this section, we elaborate on our DNCP method. The whole procedure of DNCP is shown in Fig. 2 which will be explained in detail in Section III-C. Our goal is to search for the optimal number of channels in each layer of the network. We first parameterize the probabilities of retaining different numbers of channels in each layer of the network by architecture parameters. Then we make the architecture parameters participate in the network feedforward and optimize the architecture parameters by gradient descent in an end-to-end manner with respect to the loss we proposed in Section III-B. After that, we calculate the optimal number of channels for each layer of the network according to the architecture parameters and get the final pruned structure.

A. Channel Pruning Process

Suppose we have a convolutional network with L layers. In the lth$(1 \leq l \leq L)$ layer with c^l output channels, We have c^l possible numbers of channels after pruning. We define the search space of channel pruning as the number of channels in each layer of the network. At the same time, in order to optimize the search space, we adopt the method of retaining the first i channels. In the previous methods [21], [23], in order to find the optimal number of channels in the search space, they all use heuristic algorithms such as reinforcement learning algorithms or evolutionary algorithms to select a particular number of channels. This choice is an either-or discrete operation. Discrete operations are not differentiable, so we relax the choice of a particular number of channels to a softmax over all possible numbers of channels to make the search space continuous.

We use $p_i^l (1 \leq i \leq c^l)$ to represent the probability of retaining only the first i channels in the lth layer. In this way, p_i^l can be formulated as

$$p_i^l = \frac{\exp(\alpha_i^l)}{\sum_{j=1}^{c^l} \exp\left(\alpha_j^l\right)} \tag{1}$$

where $\{\alpha_1^l, \alpha_2^l, \ldots, \alpha_{c^l}^l\}$ is a set of learnable parameters called architecture parameters. In this way, we parameterize the probabilities of selecting different numbers of channels by architecture parameters.

In order to optimize the architecture parameters by gradient descent, we need to make p_i^l participate in the network feedforward. In each branch, we multiply the output feature map for each channel with the corresponding probability of being retained as the actual output feature map. For example, in the branch that only retains the first i channels the probability of each channel in the first i channels being retained is p_i^l, and the probability of each channel in the last $c^l - i$ channels being retained is 0. Therefore, in this branch, the actual output feature map of the kth channel in the lth layer is formulated as

$$\hat{O}^l(k|i) = O^l(k|i) \times \hat{p}^l(k|i) \tag{2}$$

$$\hat{p}^l(k|i) = \begin{cases} p_i^l, & k \leq i \\ 0, & k > i \end{cases} \tag{3}$$

where $O^l(k|i)$ is the original output feature map of the kth channel in the branch that only retains the first i channels, $\hat{p}^l(k|i)$ represents the probability of the kth channel being retained in this branch. We accumulate the actual output feature map of each branch as the input to the next layer of the network. After accumulation, the actual output feature map of the kth channel in the lth layer is formulated as

$$\hat{O}_k^l = \sum_{i=1}^{c^l} \hat{O}^l(k|i) = \sum_{i=1}^{c^l} \left(O^l(k|i) \times \hat{p}^l(k|i)\right). \tag{4}$$

It can be noticed that the feature map of each branch will take up a lot of memory. So, in order to reduce memory overhead, we adopted a weight sharing strategy. In this way, the output feature maps of the corresponding channels in

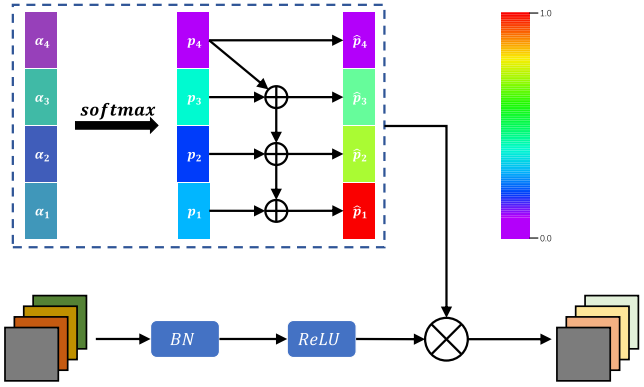

Fig. 3. Detail illustration of architecture parameters participating in network feedforward.

different branches are the same, and we can merge them. We use O_k^l to represent the output feature map of the kth channel in the lth layer, then its actual output feature map $\hat{O}_k^l$ can be formulated as

$$\hat{O}_k^l = O_k^l \times \sum_{i=1}^{c^l} \hat{p}^l(k|i). \tag{5}$$

Therefore, we do not need to use multiple branches when we participate in the architecture parameters in the network feedforward. We only need to calculate the probability of each channel being retained according to the architecture parameters, and then multiply it with the output feature map of the corresponding channel, as shown in Fig. 3. For example, the probability of the kth channel in the lth layer of the network being retained is formulated as

$$\hat{p}_k^l = \sum_{i=1}^{c^l} \hat{p}^l(k|i) \tag{6}$$

and the actual output feature map can be calculated as

$$\hat{O}_k^l = O_k^l \times \hat{p}_k^l. \tag{7}$$

Notice that $\hat{p}_1^l$ is 1 because the network must reserve at least one channel. Through the above method, the gradient can be backpropagated from $\hat{p}_i^l$ to α_i^l. The architecture parameters can be optimized by gradient descent.

In addition, it is worth noting that we need to put the above operations after the batch normalization layer as depicted in Fig. 3. Because the scaling factor of the BN layer may offset the influence of our operations on subsequent network layers.

Finally, We use $N^l(c)$, the softmax over all possible numbers of channels, as the optimal number of channels in the lth layer we searched for. $N^l(c)$ can be computed as

$$N^l(c) = \sum_{i=1}^{c^l} \frac{\exp(\alpha_i^l)}{\sum_{j=1}^{c^l} \exp\left(\alpha_j^l\right)} \times i. \tag{8}$$

After the training of the architecture parameters, we compute the optimal number of channels for each layer of the network to obtain the final optimal substructure.

B. Loss Function

When training the network parameters, we use the cross-entropy loss for classification. When training architecture parameters, our goal is to optimize the architecture parameters through gradient descent, and finally get a pruned model that not only has the best performance but also whose computation cost meets the resource constraints. As a result, the loss function takes the following form:

$$\text{Loss} = \text{Loss}_{\text{cls}} + \lambda_{\text{cost}} \text{Loss}_{\text{cost}} \tag{9}$$

where Loss_{cls} is the classification loss, $\text{Loss}_{\text{cost}}$ is the cost loss and λ_{cost} is the weight of the cost loss.

We use the soft target [43] calculated from the output of the unpruned network to compute the classification loss. We perform ablation study in Section IV-C to verify the effectiveness of using soft target. Loss_{cls} is formulated as

$$\hat{x}_i = \frac{\exp(x_i/T)}{\sum_j \exp(x_j/T)} \tag{10}$$

$$\hat{y}_i = \frac{\exp(y_i/T)}{\sum_j \exp(y_j/T)} \tag{11}$$

$$\text{Loss}_{\text{cls}} = T^2 \cdot \text{KL}(\hat{X}, \hat{Y}) \tag{12}$$

where KL denotes the Kullback–Leibler Divergence [44], X is the output of the network after architecture parameters participating in the feedforward, Y is the output of the unpruned network and T is a temperature that set to 1 in our experiment. When computing the network cost loss, we use the FLOPs of the network as the cost indicator of the network. Our goal is to make the FLOPs of the network converge to our target R. Since the FLOPs of the network cannot be optimized using gradient descent, we use the $N(\text{FLOPs})$ computed based on the architecture parameters as our optimization object. $N(\text{FLOPs})$ is formulated as

$$N(\text{FLOPs}) = \sum_{l=1}^{L} N^l(\text{FLOPs}) \tag{13}$$

$$N^l(\text{FLOPs}) = \sum_{i=1}^{c^{l-1}} \sum_{j=1}^{c^l} p_i^{l-1} \times p_j^l \times N^l(\text{FLOPs}|i, j) \tag{14}$$

where L is the number of layers of the network, c^{l-1} and c^l are the number of channels in the $(l-1)$th layer and the lth layer of the network, respectively. $N^l(\text{FLOPs}|i, j)$ represents the FLOPs of the lth layer when only i channels are reserved in the $(l-1)$th layer and j channels are reserved in the lth layer. The $\text{Loss}_{\text{cost}}$ is as follows:

$$\text{Loss}_{\text{cost}} = \log(|N(\text{FLOPs}) - R|). \tag{15}$$

Inspired by previous work [25], we also set a tolerance $\gamma \in (0, 1)$, when $N(\text{FLOPs})$ is between $\gamma \times R$ and R, $\text{Loss}_{\text{cost}}$ will be set to zero. This makes $N(\text{FLOPs})$ strictly lower than the target FLOPs we set while not being too sensitive to the target FLOPs.

Algorithm 1 Alternate Training Process for DNCP

Input: Train data D_{train}, unpruned model M.

1: **for** $t = 0 : T_{iters}$ **do**
2: Get next batch data D_t from D_{train};
3: Sample max width subnet, min width subnet and two random width subnets from M as in Fig. 2 and add them to $subnets$;
4: **for** $subnet$ in $subnets$ **do**
5: Calculate cross entropy loss for classification and accumulate gradients;
6: **end for**
7: Update network parameters;
8: Calculate $Loss$ via Eqn. (9) to update architecture parameters and obtain model $M^{(t)}$;
9: **end for**

Output: The trained model $M^* = M^{(T_{iters})}$.

C. Network Training Process

Next, we will introduce the detailed training process of our pruning algorithm. As illustrated in Fig. 2, in our training process, we not only need to update the network parameters but also the architecture parameters. Therefore, we adopt an alternate training method to alternately update network parameters and architecture parameters in each batch as shown in Algorithm 1.

1) Updating Network Parameters: For each layer of the network, since we give priority to preserving the first k channels, we need to ensure that the left channels are more important than the right channels. Previous work [45] proposed a network training method called "sandwich rule," which used the parameter sharing strategy to sample networks of different widths and train together so that the importance of each channel is no longer equal, that is, the channel on the left will be more important than the channel on the right. In the original method, the random width network uses a uniform random width for sampling.

However, the width of each layer of the pruned substructure is not the same, and we hope that the subnetwork with high probability can get more training. Therefore, we replace random sampling with importance sampling based on importance score. Our architecture parameters can learn the probabilities that the network retains different numbers of channels. We express these probabilities as the importance of the different widths of each layer of the network so that we can sample the subnetwork according to the learned probability distribution. In this way, the architecture parameters will affect the training of the network parameters, and the subnetwork with high probability will get more training.

In addition, before training the architecture parameters, we must first ensure that the importance of each channel group in each layer of the network is different. When optimizing architecture parameters through gradient descent, this can prevent the network from falling into a local minimum. Therefore, before alternate training, we need to train the network parameters for several epochs.

2) Updating Architecture Parameters: At this stage, we fix the network parameters and only update the architecture parameters. We make the architecture parameters participate in the feedforward of the network, so the gradient can be back-propagated to the architecture parameters by the following formulae:

$$\frac{\partial \text{Loss}}{\partial \alpha_i^l} = \sum_{j=1}^{c^l} \frac{\partial \text{Loss}}{\partial \hat{p}_j^l} \frac{\partial \hat{p}_j^l}{\partial \alpha_i^l} = \sum_{j=1}^{c^l} \frac{\partial \text{Loss}}{\partial \hat{p}_j^l} \sum_{k=1}^{c^l} \frac{\partial \hat{p}_j^l}{\partial p_k^l} \frac{\partial p_k^l}{\partial \alpha_i^l} \tag{16}$$

$$\frac{\partial \hat{p}_j^l}{\partial p_k^l} = \begin{cases} 1, & j \le k \\ 0, & j > k \end{cases} \tag{17}$$

$$\frac{\partial p_j^l}{\partial \alpha_i^l} = \frac{\partial \frac{\exp\left(\alpha_j^l\right)}{\sum_k \exp\left(\alpha_k^l\right)}}{\partial \alpha_i^l} = \begin{cases} p_j^l\left(1 - p_i^l\right), & i = j \\ -p_i^l p_j^l, & i \ne j. \end{cases} \tag{18}$$

In this way, we can optimize the architecture parameters in an end-to-end manner through gradient descent.

It is worth noting that architectural parameters of all network layers are jointly optimized during the training procedure. Although the architectural parameters of each layer of the network are independent of each other, we need to optimize them together in the training procedure. We use (7) to make the architectural parameters participate in network inference. The output feature map of each layer of the network is the input feature map of the next layer. Therefore, the connection between neighboring layers have an impact on the optimization of the architecture parameters. We use the symbol F^l to represent all operations of the lth layer of the network. The input feature map of the network layer is $\hat{O}^{l-1}$, that is, the output feature map of the previous layer, then the output feature map of the network layer can be calculated as

$$\hat{O}^l = F^l\left(\hat{O}^{l-1}\right). \tag{19}$$

The architecture parameters of current network layer and those of previous network layer are both involved in the calculation of the output feature map of current network layer. Therefore, when optimizing the classification loss of the network, the architecture parameters of all network layers are optimized together. At the same time, as shown in (14), when calculating the FLOPs of each layer of the network, the architecture parameters of current layer and those of previous layer are required. Therefore, when optimizing the cost loss of the network, all network layer architecture parameters are also optimized together. Therefore, in our method, the pruning rate of successive layers are jointly optimized.

In addition, in order to further reduce the search space of optimization, we group the channels of each layer of the network uniformly and define the search space as the number of channel groups in each layer. Therefore, each $\hat{p}$ represents the probability of each group of channels being retained and the output of each channel will be multiplied by the probability of the corresponding group being retained as the input of the next network layer.

IV. Experiments

In this section, we conduct extensive experiments to demonstrate the effectiveness and efficiency of our DNCP

TABLE I
TOP-1 ACCURACY OF THE PROPOSED DNCP METHOD AND UNIFORM PRUNING BASELINES AS WELL AS OTHER STATE-OF-THE-ART METHODS ON CIFAR DATASET

Network	Model	FLOPs	Top-1	ΔTop-1
Dataset		CIFAR100		
ResNet50	Uniform 1.00x	1.29G	79.01	-
	Uniform 0.75x	730M	78.32	-0.68
	SFP [18]	730M	78.30	-0.71
	MetaPruning [23]	730M	78.64	-0.37
	Autoslim [22]	730M	78.60	-0.41
	DMCP [25]	730M	78.99	-0.02
	DNCP	730M	79.22	+0.21
MobileNet V2	Uniform 1.00x	89M	75.91	-
	Uniform 0.75x	52M	74.36	-1.55
	AMC [21]	52M	75.08	-0.83
	MetaPruning [23]	52M	75.12	-0.79
	Autoslim [22]	52M	74.99	-0.92
	DMCP [25]	52M	75.33	-0.58
	DNCP	52M	75.69	-0.22
Dataset		CIFAR10		
ResNet18	Uniform 1.00x	555M	95.35	-
	Uniform 0.75x	313M	95.21	-0.14
	DNCP	313M	95.31	-0.04
	Uniform 0.50x	140M	94.57	-0.78
	DNCP	140M	94.97	-0.35
MobileNet V2	Uniform 1.00x	89M	94.15	-
	DNCP	89M	94.25	+0.10
	Uniform 0.75x	52M	93.48	-0.67
	DNCP	52M	93.71	-0.44
	Uniform 0.50x	25M	92.68	-1.47
	DNCP	25M	93.30	-0.85

TABLE II
TOP-1 ACCURACY OF THE PROPOSED DNCP METHOD AND UNIFORM PRUNING BASELINES AS WELL AS OTHER STATE-OF-THE-ART METHODS ON TINY IMAGENET DATASET

Network	Model	FLOPs	Top-1	ΔTop-1
ResNet18	Uniform 1.00	561M	61.40	-
	Uniform 0.50x	142M	55.79	-5.61
	Autoslim [22]	142M	56.60	-4.80
	DMCP [25]	142M	56.78	-4.62
	DNCP	142M	56.81	-4.59
ResNet50	Uniform 1.00x	1.30G	65.96	-
	Uniform 0.75x	735M	64.97	-0.99
	SFP [18]	735M	65.67	-0.29
	MetaPruning [23]	735M	66.20	+0.24
	Autoslim [22]	735M	66.12	+0.16
	DMCP [25]	735M	66.37	+0.41
	DNCP	735M	66.65	+0.69
MobileNet V2	Uniform 1.00x	102M	59.96	-
	Uniform 0.75x	60M	57.32	-2.64
	AMC [21]	60M	57.32	-2.64
	MetaPruning [23]	60M	58.24	-1.72
	Autoslim [22]	60M	58.30	-1.66
	DMCP [25]	60M	58.66	-1.30
	DNCP	60M	58.89	-1.07

method. First, we introduce the implementation details of our experiments on ResNet [3] and MobileNet V2 [31]. Then, we compare our results with the uniform pruning baselines as well as state-of-the-art channel pruning methods. After that, we perform ablation experiments on each component of our method. Finally, we visualize and analyze the results of our experiments.

A. Implementation Details

1) Description of Datasets: In order to demonstrate the efficiency of our method, we conduct experiments on the CIFAR, Tiny ImageNet, and ImageNet datasets with ResNet and MobilenetV2. CIFAR-10 consists of 50k training images and 10k testing images with ten classes. The size of each image is 32×32. CIFAR100 is similar to CIFAR10 but has 100 classes. Tiny ImageNet has 100k training images, with 500 images from 200 different classes, 10k testing images. The size of each image is 64×64. ImageNet contains 1.28 million training images and 50k test images with 1000 classes.

2) Experimental Settings: For ResNet, we set the minimum width to $0.1\times$, the maximum width to $1.0\times$, and the offset width to $0.1\times$. For MobileNetV2, the maximum width is set to $1.5\times$, and the other settings remain unchanged. Both network weights and architecture parameters are optimized via SGD with a momentum of 0.9. For ResNet, the weight decay is set to 5e-4, and for MobileNetV2, the weight decay is set to 1e-4. We do not add weight decay to the architecture parameters. The tolerance rate of all networks is set to 0.95. The weight of the cost loss is set to 0.1. The initial learning rate for updating network parameters is 0.1, for architecture parameters it is 0.5, and will eventually decay to one-tenth by cosine scheduler. We first train the network parameters for 20 epochs and then alternately train the network parameters and architecture parameters for 20 epochs.

After getting the pruned network, it will be trained from scratch on the training dataset. All pruned models will be trained for 200 epochs, the initial learning rate is 0.1 and reduced to 0 by cosine scheduler finally.

3) ResNet [3]: There are many residual modules with shortcut connections in the ResNet. For each residual module with identity shortcut, the number of input channels and the number of output channels must be equal. We use the strategy of weight sharing, that is, sharing the same set of architectural parameters to ensure this.

4) MobileNet V2 [31]: In MobileNet V2, there are many inverted residual modules with shortcut connections. We use the same strategy as in ResNet. In addition, there is a depth-wise convolution in each inverted residual module, and it is necessary to ensure that it has the same number of input channels and the same number of output channels. Here we also use the strategy of weight sharing.

B. Compared to the State-of-the-Art

We compare our method on CIFAR, Tiny ImageNet and ImageNet datasets with uniform pruning baselines as well as various pruning methods including traditional pruning method SFP [18] and FPGA [46], reinforcement learning method AMC [21], meta learning method MetaPruning [23], one-shot method AutoSlim [22], and Markov process-based method DMCP [25]. Uniform pruning means that the pruning rate of each layer of the network is the same when pruning the network. Taking Uniform $0.50\times$ as an example, it means that

TABLE III

TOP-1 ACCURACY OF THE PROPOSED DNCP METHOD AND UNIFORM PRUNING BASELINES AS WELL AS OTHER STATE-OF-THE-ART METHODS ON IMAGENET DATASET

Network	Model	FLOPs	Top-1	ΔTop-1
ResNet18	Uniform 1.00x	1.8G	70.1	-
	Uniform 0.75x	1.04G	67.6	-2.5
	FPGA [46]	1.04G	68.4	-1.7
	DMCP [25]	1.04G	69.1	-1.0
	DNCP	1.04G	69.2	-0.9
ResNet50	Uniform 1.00x	4.1G	76.6	-
	Uniform 0.75x	2.3G	74.6	-2.0
	FPGA [46]	2.4G	75.6	-1.0
	SFP [18]	2.4G	74.6	-2.0
	MetaPruning [23]	2.3G	75.4	-1.2
	Autoslim [22]	2.0G	75.6	-1.0
	DMCP [25]	2.2G	76.2	-0.4
	DNCP	2.2G	76.3	-0.3
	Uniform 0.50x	1.1G	71.9	-4.7
	MetaPruning [23]	1.1G	73.4	-3.2
	Autoslim [22]	1.1G	74.0	-2.6
	DMCP [25]	1.1G	74.4	-1.8
	DNCP	1.1G	74.3	-1.8
MobileNet V2	Uniform 1.00x	300M	72.3	-
	Autoslim [22]	300M	73.1	+0.8
	DMCP [25]	300M	73.5	+1.2
	DNCP	300M	73.6	+1.3
	Uniform 0.75x	210M	70.1	-2.2
	AMC [21]	211M	70.8	-1.5
	MetaPruning [23]	211M	71.2	-1.1
	Autoslim [22]	211M	72.2	-0.1
	DMCP [25]	211M	72.4	+0.1
	DNCP	211M	72.7	+0.4
	Uniform 0.50x	97M	64.8	-7.5
	MetaPruning [23]	87M	63.8	-8.5
	DMCP [25]	97M	66.1	-6.2
	DNCP	97M	66.5	-5.8

TABLE IV

TRAINING TIME AND GPU MEMORY USAGE OF OUR PROPOSED DNCP METHOD AND OTHER STATE-OF-THE-ART METHODS WHEN SEARCHING RESNET-325M NETWORK ON CIFAR100 DATASET

Method	Training Time	Training Memory
MetaPruning [23]	9.6h	10.9GB
Autoslim [22]	5.8h	5.9GB
DMCP [25]	2.9h	7.4GB
DNCP	2.5h	7.8GB

only half of the channels are reserved for each layer of the network. All methods are implemented on ResNet [3] and MobileNet V2 [31].

1) Results on CIFAR Dataset: As shown in Table I, our method exceeds the uniform pruning baselines of all networks and matches or surpasses the state-of-the-art pruning methods on CIFAR100 dataset. In addition to the CIFAR100 dataset, we also compare our method on CIFAR10 dataset with the uniform pruning baselines. We use the same hyperparameters as in the CIFAR100 experiment. Our method has better accuracy than the uniform pruning baselines on the CIFAR10 dataset.

2) Results on Tiny ImageNet Dataset: As shown in Table II, our method exceeds the uniform pruning baselines of all networks and matches or surpasses the state-of-the-art pruning methods on Tiny ImageNet dataset.

3) Results on ImageNet Dataset: As shown in Table III, our method exceeds the uniform pruning baselines of all networks and matches or surpasses the state-of-the-art pruning methods on ImageNet dataset. For example, compared to latest pruning methods such as AutoSlim [22] and DMCP [25] our DNCP-MobileNet V2 at 211MFLOPs achieves 72.7% top-1 accuracy on Tiny ImageNet, 0.3% better than DMCP-MobileNet V2 and 0.5% better than AutoSlim-MobileNet V2, under the same FLOPs.

4) Training Time and Training Memory: We also compare the training time and GPU memory usage of the proposed DNCP method and other state-of-the-art methods when searching ResNet-325M network on CIFAR100 dataset. All experiments are performed on a 1080Ti GPU. According to the results in Table IV, our method has the shortest training time but requires a larger training memory due to the need to train the architecture parameters.

TABLE V

ACCURACY OF DNCP ON CIFAR100 DATASET WITH DIFFERENT TRAINING SCHEMES

Model	Arch Params	Net Params	Top-1
ResNet18-313M	✓		78.30
	✓	✓	78.62
ResNet50-730M	✓		78.26
	✓	✓	79.22
MBV2-52M	✓		75.16
	✓	✓	75.99

TABLE VI

IMPACT OF PRUNED MODEL TRAINING STRATEGY ON THE ACCURACY OF THE PROPOSED DNCP ON CIFAR100 DATASET

Dataset	Model	Fine-tuned	Scratch
CIFAR100	ResNet18-140M	76.32	76.83
	ResNet50-730M	78.54	79.22
	MobileNet V2-52M	75.33	75.99

C. Ablation Study

1) Training Scheme: We conduct comparative experiments on two training schemes: only updating architecture parameters and alternately updating network parameters and architecture parameters. In order to ensure that the network parameters have the same amount of training in the two training schemes, for the experiments that only update the architecture parameters, we double the epochs only to update network parameters. The learning rate of updating the network parameters in the two training schemes remains the same. The experimental results are shown in Table V, MBV2 is short for MobileNet V2. We can find that the pruned model obtained by alternately updating network parameters and architecture parameters has higher accuracy. We can conclude that when the architecture parameters change, updating the network parameters can search for a better substructure.

TABLE VII
INFLUENCE OF DIFFERENT SAMPLING RULES ON THE ACCURACY OF THE PROPOSED DNCP ON CIFAR100 DATASET

Model	Sampling rule	Top-1
ResNet18-140M	random	76.58
	importance	76.83
ResNet50-730M	random	78.71
	importance	79.22
MobileNet V2-52M	random	75.53
	importance	75.99

TABLE VIII
INFLUENCE OF TARGET STRATEGY ON THE ACCURACY OF DNCP PRUNED MODELS

Dataset	Model	Target Strategy	Top-1
CIFAR100	ResNet18-313M	hard target	78.10
		soft target	78.62
	ResNet50-730M	hard target	78.34
		soft target	79.22
	MBV2-52M	hard target	75.32
		soft target	75.99
Tiny ImageNet	ResNet18-317M	hard target	60.42
		soft target	60.81
	ResNet50-735M	hard target	66.37
		soft target	66.65
	MBV2-60M	hard target	58.55
		soft target	58.89

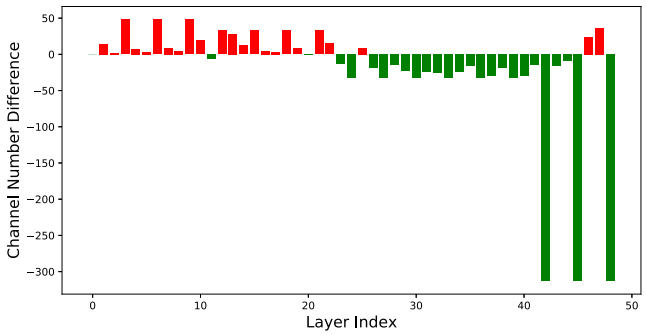

Fig. 4. Difference in the number of channels between two pruned ResNet50 models at 325MFLOPs obtained with soft target and hard target. The x-axis indicates the layer index and the y-axis is the difference computed by subtracting the number of channels each layer in the substructure obtained with hard target from that in the substructure obtained with soft target.

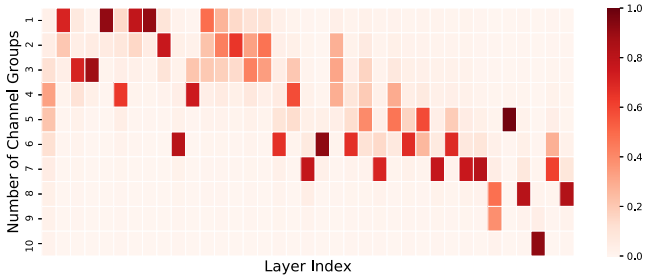

Fig. 5. Heatmap of p at the last epoch when training the architecture parameters of ResNet50 by our DNCP method at 325MFLOPs on CIFAR100 dataset.

2) Pruned Model Training Strategy: We conducted ablation experiments on the selection of training strategies for the pruned model. We use the pruned model to inherit the network weights of the unpruned model and then fine-tune them and train the pruned model from scratch to conduct experiments. As shown in Table VI, compared to fine-tuning, training the pruned network from scratch can achieve higher accuracy. We think the possible reason is when training network parameters, we deliberately make the left channel more important than the right channel. If the pruned network directly inherits the weight of the unpruned network, it may cause the network to fall into a local minimum.

3) Sampling Rule: We ablate the influence of sampling rules in network training. We use random sampling rule and importance sampling rule to train the network for comparison in our method. The comparison result is as shown in Table VII. We can find that the pruned model obtained by training the network using importance sampling rule has higher accuracy. The possible reason is that the use of importance sampling rule can allow the subnetwork with higher probability to be more fully trained, thereby making the pruned model perform better.

4) Influence of Soft Target: We compare the influence of whether using soft target or not on the performance of pruned models. According to Table VIII, we can find that the accuracy of the pruned models obtained using soft target are significantly better than that of the pruned models obtained using hard target. The possible reason is that using soft target allows the output of the substructure to match the output of the unpruned model as much as possible so that the searched optimal substructure has a better performance. It can also be noted that the soft target improves the accuracy more on the CIFAR100 dataset than on the Tiny ImageNet dataset.

We also visualize the substructures obtained using hard target and soft target. As shown in Fig. 4, we can find that the pruned substructure obtained using hard targets will retain too many channels in the deep layers of the network, leaving too few channels in the shallow layers of the network, resulting in poor model performance.

D. Visualization

1) Heatmap of Softmax Values: We visualize the softmax values of the final architecture parameters of each layer of the unpruned network. As shown in Fig. 5, we can see that the maximum value of most network layers is close to 1, but the maximum value of some network layers is not very obvious. Therefore, we take the softmax over all possible numbers of channels in the network layer as the number of channels we will eventually retain, instead of choosing the number of channels with the highest probability.

2) Channel Configurations: We visualize the channel configuration of the pruned models and compare them with the unpruned model. The pruned models at 313M and 140M we searched from ResNet50-1.00× correspond to ResNet50-0.75× and ResNet50-0.50×, respectively. As shown in Fig. 6, compared to the uniform pruning ratio, the

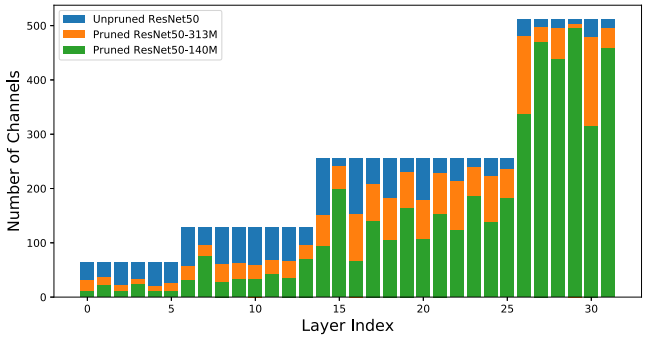

Fig. 6. Number of channels in the middle two layers of each residual module of pruned ResNet50 at 140M, 313M and unpruned ResNet50.

pruning ratio of each layer of our pruned model is different. At the same time, we can find that more channels are reserved at the network layer where downsampling occurs. We guess that because the downsampling operation will reduce the size of the feature map, the network needs more channels to retain valid features.

V. Conclusion and Future Work

In this article, we propose a DNCP method for model compression. Different from existing methods that require sampling and evaluation of a large number of substructures, our method relaxes the search space to make it continuous, and efficiently searches for the optimal substructure in an end-to-end manner through gradient descent. Then we train the substructure from scratch to get the pruned model. Our proposed method is able to match or outperform the state-of-the-art pruning algorithms on CIFAR, Tiny-ImageNet, and ImageNet datasets.

One limit of our proposed DNCP work is that it could only tune the number of channels. It would be great if this method could also be applied to choose the number of layers, which will be investigated in the near future.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[2] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[4] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7132–7141.

[5] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[8] W. Liu *et al.*, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.*, Cham, Switzerland: Springer, 2016, pp. 21–37.

[9] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent*. Cham, Switzerland: Springer, 2015, Pp. 234–241.

[10] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, Apr. 2017.

[11] G. Lin, A. Milan, C. Shen, and I. Reid, "RefineNet: Multi-path refinement networks for high-resolution semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1925–1934.

[12] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2881–2890.

[13] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*.

[14] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1389–1397.

[15] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016, *arXiv:1611.06440*.

[16] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2736–2744.

[17] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 5058–5066.

[18] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," 2018, *arXiv:1808.06866*.

[19] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, and W. Lin, "ThiNet: Pruning CNN filters for a thinner net," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 10, pp. 2525–2538, Oct. 2019.

[20] H.-J. Kang, "Accelerator-aware pruning for convolutional neural networks," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 7, pp. 2093–2103, Jul. 2020.

[21] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 784–800.

[22] J. Yu and T. Huang, "AutoSlim: Towards one-shot architecture search for channel numbers," 2019, *arXiv:1903.11728*.

[23] Z. Liu *et al.*, "MetaPruning: Meta learning for automatic neural network channel pruning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 3296–3305.

[24] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, and Y. Yang, "Asymptotic soft filter pruning for deep convolutional neural networks," *IEEE Trans. Cybern.*, vol. 50, no. 8, pp. 3594–3604, Aug. 2020.

[25] S. Guo, Y. Wang, Q. Li, and J. Yan, "DMCP: Differentiable Markov channel pruning for neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1539–1547.

[26] Z. Chen, Z. Chen, J. Lin, S. Liu, and W. Li, "Deep neural network acceleration based on low-rank approximated channel pruning," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 4, pp. 1232–1244, Apr. 2020.

[27] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 365–382.

[28] A. Zhou, A. Yao, K. Wang, and Y. Chen, "Explicit loss-error-aware quantization for low-bit deep neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9426–9435.

[29] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, "Structured binary neural networks for accurate image classification and semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 413–422.

[30] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[32] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.

[33] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 116–131.

[34] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," 2018, *arXiv:1810.05270*.

[35] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–12. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX
[36] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016, *arXiv:1607.03250*.
[37] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.
[38] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 304–320.
[39] T.-J. Yang *et al.*, "NetAdapt: Platform-aware neural network adaptation for mobile applications," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 285–300.
[40] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.
[41] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018, *arXiv:1802.03268*.
[42] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*.
[43] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.
[44] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.
[45] J. Yu and T. Huang, "Universally slimmable networks and improved training techniques," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1803–1811.
[46] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4340–4349.

Yu-Jie Zheng received the B.S. degree in electronic science and technology from Anhui University, Hefei, China, in 2019, where he is currently pursuing the master's degree in computer technology.

His research interests include pattern recognition and computer vision.

Si-Bao Chen received the B.S. and M.S. degrees in probability and statistics and the Ph.D. degree in computer science from Anhui University, Hefei, China, in 2000, 2003, and 2006, respectively.

From 2006 to 2008, he was a Post-Doctoral Researcher with the Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei. Since 2008, he has been a Teacher with Anhui University. From 2014 to 2015, he was a Visiting Scholar with The University of Texas at Arlington, Arlington, TX, USA. His current research interests include image processing, pattern recognition, machine learning, and computer vision.

Chris H. Q. Ding received the Ph.D. degree from Columbia University, New York, NY, USA, in 1987.

After that, he joined the California Institute of Technology, Pasadena, CA, USA, and then the Jet Propulsion Laboratory, California Institute of Technology. From 1996 to 2007, he was with the Lawrence Berkeley National Laboratory, University of California at Oakland, Oakland, CA, USA. Since 2007, he has been with The University of Texas at Arlington, Arlington, TX, USA. He has published about 200 articles that were cited more than 13 000 times (Google scholar). His main research areas are machine learning, data mining, bioinformatics, information retrieval, weblink analysis, and high-performance computing.

Bin Luo (Senior Member, IEEE) received the B.Eng. degree in electronics and the M.Eng. degree in computer science from Anhui University, Hefei, China, in 1984 and 1991, respectively, and the Ph.D. degree in computer science from the University of York, York, U.K., in 2002.

He is currently a Professor with Anhui University. He has published more than 200 papers in journal and refereed conferences. His current research interests include random graph-based pattern recognition, image and graph matching, and spectral analysis.

Dr. Luo has served as a Peer Reviewer for international academic journals, such as IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, *Pattern Recognition*, and *Pattern Recognition Letters*. At present, he is the Chair of the IEEE Hefei Subsection.