CCPrune: Collaborative Channel Pruning for Learning Compact Convolutional Networks

Yanming Chen[a,*], Xiang Wen[a], Yiwen Zhang[a] and Weisong Shi[b]

[a]*Key Laboratory of Intelligent Computing and Signal Processing of Ministry of Education, School of Compute Science and Technology, Anhui University, Hefei, China*

[b]*Department of Computer Science, Wayne State University, Detroit, MI, USA*

ARTICLE INFO

Keywords:
Deep convolutional neural network
Model compression
Channel pruning

ABSTRACT

Deep convolutional neural networks (CNNs) is difficult to deploy on resource-constrained devices due to its huge amount of computation. Channel pruning is an effective method to reduce the amount of computation and accelerate network inference. Most of channels pruning methods use statistics from a single structure (convolutional layer or batch normalization layer) of the sparse network to evaluate the importance of channels. The limitation of these methods is that it may often mistakenly delete the important channels. In view of this, we propose a novel method, namely Collaborative Channel Pruning (CCPrune), to evaluate the importance of channels, which combines the convolution layer weights and the BN layer scaling factors. The proposed method first introduces the regularization on the convolution layer weights and the BN layer scaling factors respectively. Then combine the weight of the convolutional layer and the scaling factor of the BN layer to evaluate the importance of the channel. Finally, it can delete the unimportant channels without reduces the performance of the model. The experimental results well demonstrate the effectiveness of our method. On CIFAR-10, it can reduce the FLOPs of VGG-19 by 85.50% while only slightly reducing the accuracy of the model, and it can reduce the FLOPs of Resnet-50 by 78.31% without reducing the accuracy of the model, respectively.

1. Introduction

In recent years, the deep convolutional neural networks (CNNs) have achieved great success in computer vision such as image classification[17, 27, 47, 49], object detection[1, 7, 14, 43, 50], and semantic segmentation[3, 36], and so on. However, with the continuous improved performance of CNNs, the depth of the network model is deeper and deeper, and the storage space and computing cost are higher and higher. For example, the VGG[47] model requires about 500M of storage space and 15 billion floating point operations (FLOPs) for a single image classification. The network structure is very redundant. However, most devices cannot provide enough storage space and computing power required for CNNs, which makes it generally difficult to deploy CNNs on resource-constrained devices, such as mobile devices, Raspberry Pi, or IoT devices. Especially in recent years, the development of edge computing[46] and autonomous vehicle systems has become more and more popular, and applications in these fields usually require network models to have higher real-time performance and less resource occupancy.

In order to solve the above contradictions, many methods have been proposed to reduce the storage space and computation cost of network models. These methods include weight quantization[4, 6, 42, 51], low-rank decomposition[9, 25, 55], pruning[2, 16, 18, 20, 30, 39, 40] and knowledge distillation[5, 21, 44, 52]. The application scenarios of weight quantization and low-rank decomposition are very limited. Some of them require a specific acceleration framework to be implemented, such as TensorRT. The student network structure of knowledge distillation is difficult to determine, and generally requires professional artificial design. Relatively speaking, pruning is easier to obtain a compact network, and it can greatly reduce storage space and computational cost.

Pruning can be divided into two categories: weight pruning[2, 16] and channel pruning[18, 20, 30, 39, 40]. Weight pruning directly deletes unimportant connections, which easily produces irregular network structure and discontinuous storage space. Meanwhile, as discussed in [31] and [18], weight pruning often require specific software or hardware support, and they are difficult to use Basic Linear Algebra Subprograms (BLAS). Channel pruning can directly delete

*Corresponding author

cym@ahu.edu.cn (Y. Chen); e19301112@stu.ahu.edu.cn (X. Wen); zhangyiwen@ahu.edu.cn (Y. Zhang); weisong@wayne.edu (W. Shi)

http://weisongshi.org (W. Shi)

unimportant channels without changing the network structure, so that the existing framework can be used to achieve more effective compression and acceleration.

The method based on channel pruning is a very effective and popular pruning method. Most channel-based pruning methods introduce sparsity regularization term to train the model to zero out some channels, and then delete these channels. The compact network obtained by the channel-based pruning method can maintain the original structure, and can use existing libraries to achieve effective compression and acceleration.

Many previous methods only evaluate the importance of channels based on single structure (convolutional layer or Batch Normalization layer). Such as Li et al.[31] proposes to evaluate the importance of channels based on the L1-norm criterion. They believe that the filters with a smaller values have smaller contributions, which can be safely removed without reduces the performance of the model. Liu et al.[35] introduces sparsity regularization on the scaling factors of the Batch Normalization (BN)[23] layers, using these scaling factors to evaluate the importance of the corresponding channels. In addition to Liu et al.[35]. Zhao et al.[56] also considers the parameters of the BN layer, they believe that not only the scaling factor needs to be considered, but the shift factor should also be considered. We call these methods " channel pruning based on single structure". When these methods only use convolutional layers or BN layers to evaluate the importance of a channels, they sometimes mistakenly delete important output feature maps. They did not comprehensively consider the relationship between the two consecutive structures before the activation function. That is, when the channel in the convolutional layer is judged to be unimportant, its corresponding scaling factor may be important, vice versa. For example, Li et al.[31] evaluates the importance of channels based on the L1-norm criterion, but they did not considered whether the BN layer will change the channels contribution ability. Consequently, the channel will be misjudged as an important channel.

In this paper, we propose a new method, namely Collaborative Channel Pruning, to evaluate the importance of channels, which combines the convolution layer weights and the BN layer scaling factors. Different from the previous methods, the relationship between the convolutional layer and the BN layer is considered for channel pruning. We first introduce the L1 regularization on the convolution layer weights and the BN layer scaling factors, respectively. And then combine the convolution layer weights and the BN layer scaling factors to judge the importance of channels, rather than relying on the statistical data of a single layer to evaluate the channel importance. Finally, we can delete the unimportant channels without reducing the performance of the model. We use a greedy algorithm to iteratively prune all channels at the global level. Rather than performing sensitivity analysis on each layer as in [31], setting a different pruning ratio for each layer. In order to maintain the accuracy of the network, after each pruning, we will retrain the pruned network to restore the accuracy of the network. For VGG-19 on CIFAR-10 [26], our method can reduce the FLOP of VGG-19 by 85.50% while only slightly reducing the accuracy of the model.

Our contribution can be summarized as follows:

1. We analyze the works that only consider the convolutional layer or the BN layer to evaluate the importance of channels. They did not comprehensively considered the relationship between the two consecutive structures before the activation function. To solve the above issues, we propose a novel pruning method, namely Collaborative Channel Pruning, which combines the convolution layer weights and the BN layer scaling factors.

2. We propose a new method to pruning unimportance channels, which combines the convolution layer weights and the BN layer scaling factors. We first introduce the regularization on the convolution layer weights and the BN layer scale factors respectively. Then combine the convolution layer weights and the BN layer scale factors to evaluate the importance of channels. Finally, we can delete the unimportant channels.

3. The experimental results show that our method is effective and efficient. For VGG-19 on CIFAR-10 [26], our method can reduce the FLOPs of VGG-19 by 85.50% while only slightly reducing the accuracy of the model. To the best of our knowledge, and as far as we know, our work is the first to consider both the convolutional layer and the BN layer.

2. Related Work

Pruning is a very effective compression method, which can be divided into two categories: unstructured pruning[16] and structured pruning[12, 18, 19, 22, 31, 35, 37, 53].

Unstructured pruning is mainly to delete individual weights. Han et al.[16] focuses on deleting unimportant connections in the network to reduce network parameters and reduce storage space, this method deletes connections below the threshold based on the magnitude. Ding et al.[13] proposed to divide all parameters into two categories in each iteration, then use different update rules to update them, and finally zero out the unimportant parameters. However, unstructured pruning will produce a sparse network structure, resulting in discontinuous storage space.

In comparison, structured pruning is the more popular method. C-SDG[11] to train some filters to gather on a single point, similar filters at the same point can be deleted without reducing the accuracy of the model. Ding et al.[10] believe that choosing a suitable pruning layer is more important, to this end, they use long short-term memory (LSTM) to learn the hierarchical characteristics of the network, and propose a data-related soft pruning method-Squeeze-Excitation Pruning (SEP). Li et al.[31] uses the sum of the absolute value of the filters to evaluate the importance of the filters. Unimportant filters have a small contribution to the output feature maps and can be safely deleted. Liu et al.[35] analyzes the operation of the BN layer to consider the importance of channels, which uses the value of the scaling factors to evaluate the importance of channels, and the channels corresponding to the unimportant scaling factors should be deleted. They did not notice that the conversion of the BN layer will affect the channel's contribution ability. Zhao et al.[56] also considers the parameters of the BN layer, they believe that not only the scaling factor needs to be considered, but the shift factor should also be considered. A data-driven sparse structure selection is proposed[22], which introduces scaling factors (a new parameter) to the specified structure, then add regularization to this parameter. The corresponding structure of the scaling factors that tends to 0 will be deleted. He et al.[18] uses the soft pruning method, the pruned channel can be restored in the subsequent training process. Yu et al.[53] uses feedback propagation to calculate the importance score of channels, and channels with low score will be deleted. He et al.[19] proposes geometric median to determine redundant channels. Luo et al.[37] determines the importance of channels based on the statistical information of the next layer. However, many of the above methods only consider the statistical information of one of the weights and the scaling factors to determine the channel to be deleted, while ignore the influence of the other.

In addition to pruning, weight quantization [4, 6, 15, 24], low-rank decomposition [9, 15, 29], and knowledge distillation [21, 44, 52] are also very effective compression methods. Moreover, these methods are not mutually opposed, such as Mishra et al.[38] combines knowledge distillation and weight quantization to achieve further compression. Our method can also be combined with these methods to achieve further compression.

3. Proposed Method

Most previous methods only consider the statistical information of a single element between the weights and the scaling factors to evaluate the importance of channels. For example, Li et al.[31] uses the sum of the absolute value of the filters to evaluate the importance of filters. It believes that the filters with a smaller value will produce feature maps with weaker activity than a filters with a larger value. In this part, we will analyze the operation of the convolutional layers and the BN layers, and then introduce our method in detail.

3.1. Motivation

We start by analyzing the operation of convolutional layers and BN layers. The operation of convolutional layers can be expressed by the following Equation(1):

$$Z_l = W_l X_l + b_l \tag{1}$$

where X_l is the input of the l-th convolutional layer, $W_l \in \mathbb{R}^{n_{l+1} \times n_l \times H_l \times G_l} (1 \leq l \leq L)$ and b_l denote the weights and bias of the l-th convolutional layer, respectively. n_{l+1} is the number of output channels, n_l is the number of input channels, H_l and G_l are the height and width of the convolution kernel, Z_l is the output feature map, and L is the number of layers. Many pruning methods[18, 31] focus on the redundancy of the channels of the convolutional layers. These methods believe that redundant channels have no positive effect on the performance of the model, and even delete these channels will not reduce the performance of the model. These methods are very effective without the BN layer, but for the modern network with BN layer, the relationship between the convolutional layer and the BN layer should be considered.

The modern CNNs usually add a BN layer after the convolutional layers. After add the BN layer, the bias of the convolutional layers is generally discarded, that is, the operation of the l-th convolutional layer will become the following Equation(2):

$$Z_l = W_l X_l \tag{2}$$

The Batch Normalization (BN) layers use mini-batch statistics to standardize the output of the convolutional layers, which can improves the training speed and accelerates convergence process. The transformation of the BN layers is as

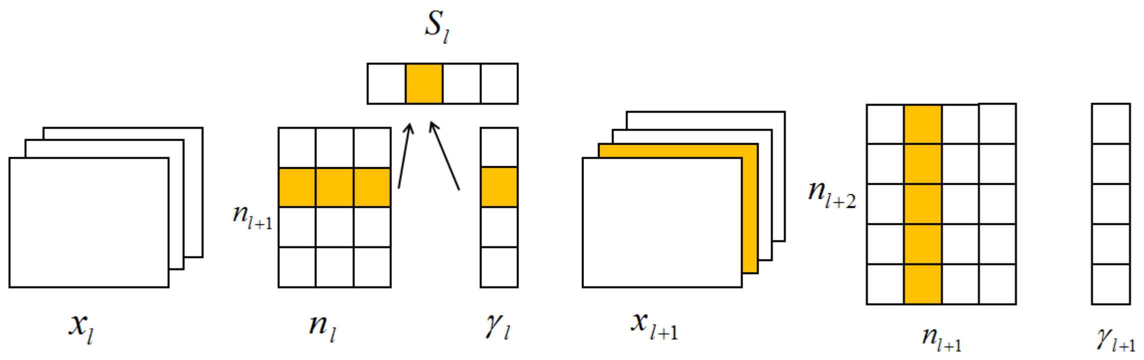


Figure 1: We can get the l-th layer and global channel importance vector by Equation(8) and Equation(9). By comparing with all channels, unimportant channels in each layer will be selected. As shown in the figure, the channel corresponding to the position of the yellow grid in S_l has a small contribution and can be safely deleted.

follows:

$$X_{l+1} = \gamma_l \frac{Z_l - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} + \beta_l \tag{3}$$

X_{l+1} represents the output of the l-th BN layer, Z_l is the input of the BN l-th layer, and $\boldsymbol{B}$ denotes the current mini-batch. $\mu_{\boldsymbol{B}}$ and $\sigma_{\boldsymbol{B}}$ are the mean and standard deviation over $\boldsymbol{B}$. γ_l (scaling factor) and β_l (shift) are learnable parameters. Because the scaling factors of the BN layers corresponds to the output channels one to one, some methods[35, 56] are inspired by this relationship and consider the importance of the channels through the scaling factors. However, these methods face the same problems as those that only consider channels redundancy on the convolutional layers, and do not consider the relationship between the two layers.

3.2. Sparsity Regularization Training

To solve the above issues, we propose a novel method to evaluate the importance of channels, which combines the convolution layer weights and the BN layer scaling factors. We first introduce sparsity regularization on weights and scaling factors so that we can better select important channels. Based on this, our optimization objective during training is as follows:

$$\text{LOSS} = G(f(X, W), Y) + \alpha_1 R(W) + \alpha_2 R(\gamma) \tag{4}$$

where,

$$R(W) = \sum_{l=1}^{L} \sum_{j=1}^{n_{l+1}} \sum_{i=1}^{n_l} \sum_{h=1}^{H_l} \sum_{g=1}^{G_l} \|W_l\| \tag{5}$$

$$R(\gamma) = \sum_{l=1}^{L} \sum_{j=1}^{n_{l+1}} \|\gamma_l\| \tag{6}$$

X is the input dataset, Y is the corresponding label, W is the weights set of all convolutional layers, γ is the scaling factors set of all BN layers. γ_l denotes the scaling factors of the l-th BN layer, which is an n_{l+1} dimension vector. $G(f(X, W), Y)$ is the normal training loss on dataset X. $R(\cdot)$ denotes sparsity regularization term. In this paper, we choose the widely used L1 regularization to get a sparse network. α_1 and α_2 are hyperparameters, which are used to balance the normal loss function and the sparsity regularization term.

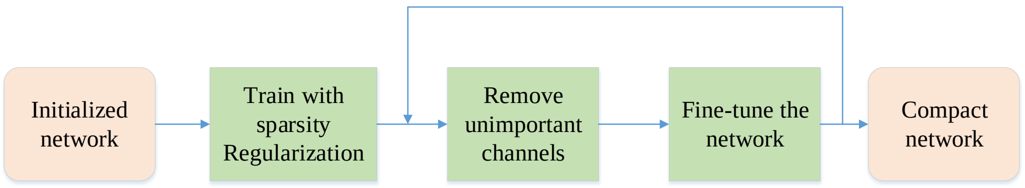

Initialized network
Train with sparsity Regularization
Remove unimportant channels
Fine-tune the network
Compact network

Figure 2: Pruning flow-chart. This figure corresponds to Algorithm 1. More details are introduced in Algorithm 1.

3.3. Channel Selection

Based on the above analysis, by combining Equation(2) and Equation(3), we can get the following Equation(7):

$$X_{l+1} = \gamma_l \frac{W_l X_l - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} + \beta_l \tag{7}$$

By the Equation(7), we can find that there is a product relationship between scaling factors and weights. Many previous methods did not comprehensively considered the impact of this relationship. Therefore, these methods are more likely to mistakenly delete important channels.

In order to more accurately evaluate the importance of the channels, and delete channels with smaller contribution. We combine the weights of the convolutional layers and the scaling factors of the BN layers to jointly evaluate the importance of channels, rather than just use one of them to determine which channel should be deleted. For computational efficiency, we will use the channel importance vector S to denotes the importance of channels. The channel importance vector can be obtained uses the following Equation(8):

$$S_l = \|\gamma_l\| \sum_{i=1}^{n_l} \sum_{h=1}^{H_l} \sum_{g=1}^{G_l} \|W_l\| \tag{8}$$

$$S = U_{l=1}^{L} s_l \tag{9}$$

Through Equation(8), we will get an n_{l+1} dimension vector S_l, each value of which represents the importance of its corresponding channel. We can get a global channel importance vector S by Equation(9), which allows us to compare the importance of channels at the global level.

As shown in Fig.1, we use a two-dimensional matrix with n_{l+1} rows and n_l columns to denote the weight matrix of n_{l+1} output channels and n_l input channels. In each layer, we can use Equation(8) to get the channel importance vector S_l of this layer, then the channel importance vector S for all channels can be obtained by uses Equation(9). The unimportant channel will be removed at the pruning stage without affecting the performance of the model. In Fig.1, the smaller value in S is represented by a yellow grid, which means that the channel is unimportant. During pruning, the corresponding channels and scaling factors will be discarded. And the corresponding input channels of the next layer will also be removed. That is, the yellow grids in Fig.1 will be removed.

3.4. Pruning Frame

Our pruning flow chart is shown in Fig.2. For an initialized network model, we firstly train the network model from scratch with sparsity regularization, then use Equation(8) to evaluate the importance of channels, finally prune unimportant channels. As shown in Algorithm 1, in order to maintain the accuracy of the network model, we iteratively use a greedy algorithm to prune the network globally, and fine-tune the new network model to restore the lost accuracy.

Compared with one-shot pruning strategy, we use an iterative method to prune the network. We fine-tune the network to restore the accuracy of the network after each pruning, which will be used as the next pruned network. Because one-shot pruning is easier to delete the more important channels in the network model, which seriously damage the accuracy of the model, making fine-tuning unable to restore its accuracy. Relatively speaking, the iterative pruning method will get a smooth model without harm the performance of the network, and even the accuracy will be improved after fine-tuning. We believe that this is because the unimportant channels in the network will learn wrong information, which can lead to wrong judgment, and affect the judgment ability of the whole network.

Another disadvantage of one-shot pruning is that it is easy to delete the whole layer, which will destroy the structure of the network. To solve aforementioned issues, many methods are to set the maximum pruning ratio for each layer for retain at least a certain number of channels. In this paper, we use a simple and effective way to deal with this problem. During the pruning process, our only restriction is that the channels in each convolutional layer cannot be pruned to 0 in this iteration. In the iterative process, if all channels in this layer are judged as unimportant channels, the pruning operation of this layer will be cancelled. The extreme case is that this layer will only retain one channel.

Algorithm 1 CCPrune algorithm

```
Input: Dataset $X$, label $Y$
1: Initialize model parameters $W$
2: Training model $W^0$ with sparsity regularization
3: Set the pruning iterations $K$ and the pruning ratio $P$
4: for $k = 1$ to $K$ do
5:     for $l = 1$ to $L$ do
6:         $S_l \leftarrow$ Equation(8)
7:     end for
8:     $S \leftarrow$Equation(9), $S^{sort}$=sort($S$)
9:     $C \leftarrow$Calculate the number of channels
10:    Delete $C \cdot P$ channels, Obtain new model
11:    Initialize the new model weight $W^k \leftarrow W^{k-1}$
12:    Fine-tune the new model
13:end for
Output: The Compact model
```

4. Experiment

In this section, we evaluate our method on the CIFAR-10[26], CIFAR-100[26], MNIST, SVHN[41], CINIC-10[8] and ImageNet[45] datasets, using two popular networks, VGG[47] and Resnet[17]. In VGG, we choose VGG-11, VGG-16 and VGG-19 for experiment. In Resnet family, we choose Resnet-20, Resnet-32, Resnet-56, Resent-110 and Resnet-50 for experiment. Because VGG and Resnet50 are designed for the ImageNet dataset, in this paper, we choose their variants[28, 54] for experiment.

4.1. Training and Pruning Setting

We use SGD to train all networks from scratch as baseline. The weight delay is set to 10^{-4}, and the Nesteroy momentum[48] is set to 0.9. The initial learning rate is set to 0.1. We initialize all weights and scaling factors with the setting of Liu et al.[35]. On the CIFAR dataset, for VGG-16, VGG-19 and Resnet-50, we use mini-batch size 64 to train for 160 epochs, at 80 epochs and 120 epochs, and divide the learning rate by 10. And for Resnet20, 32, 56, 110, we train 200 epochs, divide the learning rate by 10 at 40%, 60% and 80% of the number of training epochs. On the SVHN and MNIST datasets, we train 30 epochs with mini-batch size 64, at 10 epochs and 20 epochs, divide the learning rate by 10. On the CINIC-10 dataset, we train 100 epochs with mini-batch size 256, at the 50 epochs and 75 epochs, divide the learning rate by 10. On the ImageNet dataset, we train the VGG-11 network for 60 epochs, with a batch size 256, the initial learning rate is set to 0.1, which is divided by 10 at 20 and 40 epochs.

When training with sparsity regularization, α_1 and α_2 are hyperparameters, which control the balance between the normal loss function and the sparsity regularization terms. The choice of these two hyperparameters is very important. When choosing hyperparameters, we refer to previous work Liu et al.[35] and OICSR[32]. In Liu et al.[35], it adds regularization to the BN layer scaling factor, and search for the value of the hyperparameter on 10^{-3}, 10^{-4}, 10^{-5}. Finally, they chose 10^{-4} for VGG and 10^{-5} for ResNet. In OICSR[32], they combine one out-channel in current layer and the corresponding in-channel in next layer as a regularization group. During training, they empirically set the value

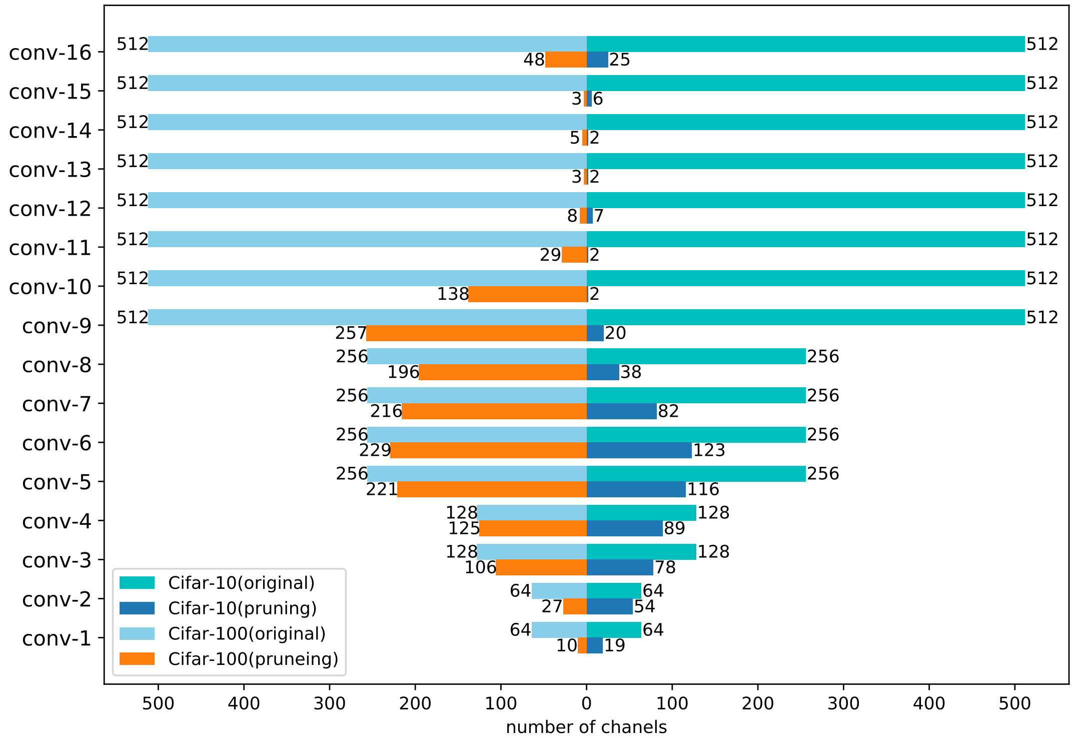


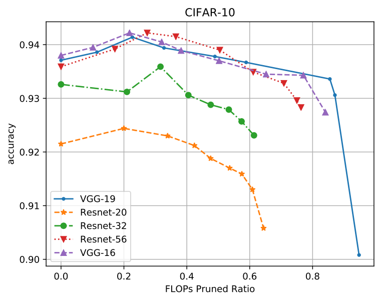
Figure 3: VGG-19(85.50%) on CIFAR-10 and VGG-19(62.10%) on CIFAR-100. The statistics of the remaining channels in the convolutional layer.

of the hyperparameter to 10^{-4} (except for DenseNet-89 with 10^{-5}), and they also suggest that more complex networks should set smaller values for the regularization parameters. These two papers provide a lot of references for our work. Another important principle in selecting these values is that the influence of the value of these hyperparameters should be less than the influence of the learning rate, so we set the value of the hyperparameter to be less than the value of the learning rate. At the same time, because the number of convolutional layer parameters is more than that of the BN layer scaling factor, we consider that α_1 should be much smaller than α_2, so as not to affect the accuracy of the model. We have also verified the above description in the 9 combinations of $\alpha_1 = 10^{-5}, 10^{-6}, 10^{-7}$ and $\alpha_2 = 10^{-3}, 10^{-4}, 10^{-5}$. The experimental results are shown in Table 1. It can be seen that when α_2 is set to 10^{-3}, the accuracy is worse. Other combinations have better performance, according to the two works mentioned above and considering that the network after pruning is more sensitive, we conservatively set the values of α_1 and α_2. Based on the above considerations, we have chosen our own hyperparameter values. Finally, on VGG, we set $\alpha_1 = 10^{-6}$, $\alpha_2 = 10^{-4}$, and on Resnet, we set $\alpha_1 = 10^{-7}$,$\alpha_1 = 10^{-5}$.

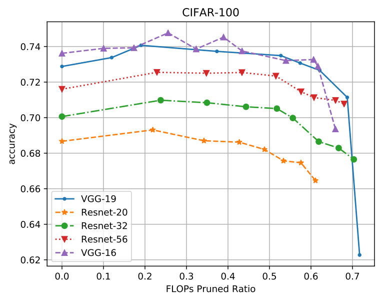
In each iteration, we prune the corresponding number of channels according to the pruning ratio. The accuracy of the pruned network will decrease, we fine-tune them to restore accuracy, and the fine-tuning uses the same optimization settings as sparsity regularization training. In the pruning process, sometimes all channels of the whole layer are evaluated as unimportant channels. We propose a simple and effective method to deal with this problem, that is, if this iteration will cause the number of output channels to be pruned to 0, we will cancel the pruning operation for this layer, and keep all the remaining channels in the last iteration. Because we are iteratively pruning on a global level, instead of setting the pruning rate separately for each layer. Therefore, when we set a larger pruning rate in one iteration, we sometimes judge the remaining channels in a certain layer as unimportant channels, and then prune them. That is, the channels of this layer are not only compared with the channels of this layer, but also compared with

Table 1
Hyperparameter search(VGG-19 on the Cifar-10 dataset).

Hyperparameter	$\alpha_1 = 10^{-5}$	$\alpha_1 = 10^{-6}$	$\alpha_1 = 10^{-7}$
$\alpha_2 = 10^{-3}$	88.33%	88.47%	88.92%
$\alpha_2 = 10^{-4}$	93.64%	93.66%	93.56%
$\alpha_2 = 10^{-5}$	93.66%	93.64%	93.71%



(a)



(b)

Figure 4: The detail of the pruning process about several benchmark networks. In the early stage of pruning, the proposed method can improve the accuracy of the model.

the channels of other layers. When comparing with other layers, it is possible that all channels of this layer are in a weak position. We think this phenomenon can be predicted, and as the pruning progresses, the number of channels in each layer becomes less and less, and this phenomenon will occur more and more frequently. Under this strategy, the most extreme phenomenon is that the number of channels reserved by the convolutional layer is 1. In other words, the maximum pruning rate we set is to reserve at least one channel per layer. It can ensure that the pruning rate is maximized, and we believe that this strategy is simple and effective, and it can save time and energy.

4.2. Results

4.2.1. Results on CIFAR

The CIFAR-10 and CIFAR-100 datasets contain 50,000 training images and 10,000 test images respectively. The difference is that CIFAR-10 has 10 classes, and CIFAR-100 has 100 classes. For CIFAR-10 and CIFAR-100 datasets. During training, we choose the model with the highest test accuracy as the model to be pruned.

The results on the CIFAR dataset are shown in Table 2 and Table 3. Our method greatly reduces the amount of FLOPs and parameters, and even improves the accuracy of the model within a certain degree of pruning ratio. Fig.3 shows the statistics of the remaining channels in the VGG-19 convolutional layer. In Fig.3, we can clearly understand the network structure after pruning. Fig.4 shows the details of the pruning process of several benchmark networks. It can be seen that when the pruning rate is between 0% and 40%, the performance of almost all models has been improved. We think this is because our criteria for judging the importance of channels are more effective. As shown in Table 2, on CIFAR10, VGG-19 and Resnet-50 reduce FLOPs by 48.92% and 49.96%, but their accuracy increased by 0.07% and 0.89%, respectively. When we reduce FLOPs by 85.50% and 78.31% respectively, their accuracy has only slightly decreased. Especially in the VGG-19 model, only 0.46M parameters were retained. On the CIFAR-100 dataset, because there are 100 classes, the pruning ratio is slightly lower than CIFAR-10. As shown in Table 3, for VGG-19 and Resnet-50, we still reduced the FLOPs by more than 60%, but only slightly reduced accuracy, even on Resnet-50, we improved the accuracy by 0.02%. On the CIFAR-10 and CIFAR-100 datasets, we can reduce FLOPs by more than 50% from Resnet-20 to Resnet-110.

Table 2
Results on CIFAR-10.The first and third columns denote the pruning ratio of FLOPs and the pruned ratio of the parameters. The second and fourth columns denote the amount of parameters and FLOPs. Baseline denote normal training without sparsity regularization.

FLOPs Pruned Ratio	FLOPs	Params pruned	Params	Accuracy
VGG-19(Baseline)	7.97×10^8	-	20.03M	93.71%
VGG-19(48.92%)	4.07×10^8	86.82%	2.64M	93.78%
VGG-19(85.50%)	1.15×10^8	97.70%	0.46M	93.36%
VGG-16(Baseline)	6.27×10^8	-	14.72M	93.80%
VGG-16(73.68%)	1.65×10^8	94.90%	0.75M	93.39%
Resnet-50(Baseline)	26.09×10^8	-	23.52M	94.87%
Resnet-50(49.96%)	13.05×10^8	53.95%	10.83M	95.76%
Resnet-50(78.31%)	5.65×10^8	88.84%	5.29M	94.84%
Resnet-20(Baseline)	0.82×10^8	-	0.27M	92.15%
Resnet-20(57.31%)	0.35×10^8	37.03%	0.17M	91.59%
Resnet-32(Baseline)	1.39×10^8	-	0.46M	93.26%
Resnet-32(57.55%)	0.59×10^8	34.78%	0.30M	92.57%
Resnet-56(Baseline)	2.53×10^8	-	0.85M	93.59%
Resnet-56(73.51%)	0.67×10^8	58.82%	0.35M	93.04%
Resnet-110(Baseline)	5.10×10^8	-	1.73M	94.11%
Resnet-110(68.03%)	1.63×10^8	64.16%	0.62M	93.36%

4.2.2. Results on MNIST and SVHN

Both MNIST and SVHN (The Street View House Numbers) are numeric datasets. The MNIST dataset is a handwritten dataset, including a training set of 60,000 samples and a test set of 10,000 samples. Each images is composed of 28×28 pixel gray value. The SVHN (The Street View House Numbers) dataset[41] includes 73257 training set images and 26,032 test set images, and 531131 relatively simple extended training set.

When pruning the network, we choose the model saved in the last epoch during training as the pruned model. The results are shown in Table 4 and Table 5. On both datasets, for VGG-19 and Resnet-50, we are able to reduce FLOPs by more than 90% and 85%, respectively. Only 0.39M and 0.14M parameters are kept on VGG-19, saving more than 98% of the parameters. Only the parameters of 5.45M and 4.00M are retained on Resnet-50, saving more than 76% of the parameters.

4.2.3. Results on CINIC-10

The CINIC-10[8] dataset is a dataset obtained by combining the images selected from CIFAR-10 and the images down-sampled from the Imagenet dataset. There are 270000 images, which are divided into three subsets, the training set, validation set and test set contain 90,000 images respectively. We combine the training set and the validation set as the training set of our experiment. We randomly select 30,000 images from the test set as the test set of our experiment. The results are shown in Table 6. Because the images of the CINIC-10 training set are more different (combined from the CIFAR-10 and Imagemet datasets), the training accuracy is lower than that of CIFAR-10. We reduce FLOPs by 62.49% and 71.46% on VGG-19 and Resnet-50, respectively. And retained only one-tenth of the parameters on VGG-19 without reducing the performance of the model.

4.2.4. Results on ImageNet

The imageNet dataset is a very large dataset, with 1000 classes, including 1.2 million training images and 50,000 validation images. The results on the ImageNet dataset are shown in Table 7, we summarized the accuracy of normal training, training with regularization and pruned network. When the regularization term is introduced, the accuracy of

Table 3
Results on CIFAR-100

FLOPs Pruned Ratio	FLOPs	Params pruned	Params	Accuracy
VGG-19(Baseline)	7.97×10^8	-	20.08M	72.88%
VGG-19(42.48%)	4.58×10^8	77.78%	4.46M	73.73%
VGG-19(62.10%)	3.02×10^8	87.60%	2.49M	72.66%
VGG-16(Baseline)	6.27×10^8	-	14.77M	73.62%
VGG-16(54.06%)	2.88×10^8	72.98%	3.99M	73.21%
Resnet-50(Baseline)	26.09×10^8	-	23.52M	77.72%
Resnet-50(41.09%)	15.37×10^8	15.48%	20.03M	79.32%
Resnet-50(65.25%)	9.06×10^8	33.16%	15.84M	77.74%
Resnet-20(Baseline)	0.82×10^8	-	0.27M	68.67%
Resnet-20(53.65%)	0.38×10^8	25.92%	0.20M	67.57%
Resnet-32(Baseline)	1.39×10^8	-	0.47M	70.06%
Resnet-32(58.27%)	0.58×10^8	29.78%	0.33M	69.98%
Resnet-56(Baseline)	2.53×10^8	-	0.86M	71.60%
Resnet-56(66.01%)	0.86×10^8	40.69%	0.51M	70.97%
Resnet-110(Baseline)	5.10×10^8	-	1.73M	72.49%
Resnet-110(61.76%)	1.95×10^8	41.61%	1.01M	72.73%

Table 4
Results on SVHN

FLOPs Pruned Ratio	FLOPs	Params pruned	Params	Accuracy
VGG-19(Baseline)	7.97×10^8	-	20.03M	95.54%
VGG-19(73.49%)	2.11×10^8	90.61%	1.88M	95.77%
VGG-19(92.35%)	0.61×10^8	98.05%	0.39M	95.48%
Resnet-50(Baseline)	26.09×10^8	-	23.52M	95.83%
Resnet-50(76.03%)	6.25×10^8	62.71%	8.77M	96.11%
Resnet-50(85.73%)	3.72×10^8	76.82%	5.45M	95.41%

the model is reduced by 0.41%. Because the regularization term may affect our pruning effect, we also look forward to finding suitable fine-tuning strategies with regularization terms. Even so, our network has a good performance on the ImageNet dataset. As shown in Table 7, when we prune nearly 30% (31.95%) of FLOPs, it still maintains 65.72% accuracy, when pruning about 40% (39.89%) of FLOPs, it only reduces accuracy by 2.67%. We also pay attention to another important resource usage - GPU memory usage. This problem is often overlooked in previous work, we believe that GPU memory usage is equally important. When the model has a large number of FLOPs, its deployment on resource-constrained equipment may be computationally slow, but it can still be executed. However, if the memory footprint of the model is much greater than the maximum value the device can provide, comparing to the slower computation speed, completely unable to run may be even worse. As can be seen in Table 7, we reduce the GPU memory footprint by 42.47% while only reducing the accuracy by 2.67%. Hence, our method is effective and efficient.

Table 5
Results on MNIST

FLOPs Pruned Ratio	FLOPs	Params pruned	Params	Accuracy
VGG-19(Baseline)	7.95×10^8	-	20.03M	99.55%
VGG-19(69.8%)	2.40×10^8	76.9%	4.61M	99.55%
VGG-19(96.4%)	0.28×10^8	99.3%	0.14M	99.54%
Resnet-50(Baseline)	26.06×10^8	-	23.51M	99.54%
Resnet-50(54.92%)	11.75×10^8	42.74%	13.46M	99.60%
Resnet-50(86.92%)	3.40×10^8	82.98%	4.00M	9.59%

Table 6
Results on CINIC-10

FLOPs Pruned Ratio	FLOPs	Params pruned	Params	Accuracy
VGG-19(Baseline)	7.97×10^8	-	20.03M	86.79%
VGG-19(62.49%)	2.99×10^8	90.51%	1.90M	86.99%
Resnet-50(Baseline)	26.09×10^8	-	23.52M	88.30%
Resnet-50(71.46%)	7.44×10^8	45.06%	12.92M	88.53%

Table 7
Results on ImageNet. Because of the hugeness of the ImageNet dataset, in addition to FLOPs and the amount of parameters, we also pay attention to the GPU memory usage during training

FLOPs Pruned Ratio	FLOPs	Params pruned	Params	Accuracy	GPU
VGG-11(Normal training)	150.16×10^8	-	9.74M	67.80%	26102M
VGG-11(Training with regularization)	150.16×10^8	-	9.74M	67.39%	-
VGG-11(31.95%)	102.18×10^8	11.57%	8.61M	65.72%	15856M
VGG-11(39.89%)	90.25×10^8	20.38%	7.75M	65.13%	15016M

Table 8
Comparison of VGG19 on CIFAR-100

Model	Accuracy	Params pruned
Baselline	72.88%	-
Pruned[31]	71.64%	76.00%
Pruned[35]	73.48%	75.10%
Pruned(ours)	73.73%	77.78%

4.3. Comparison with Other Methods

In this subsection, we compare our method with other methods. As shown in Table 8, on CIFAR-100, when the parameter pruning ratio is almost the same, our method has better performance than [35] (Liu et al.) and [31] (L1). When about 75% of the parameters are pruned, our method improves the top-1 accuracy by 2.09% and 0.25%, respectively.

Table 9
Normal training VS sparse training on VGG-19. The sparse regularization training method will slightly affect the accuracy of the model

Model	CIFAR-10	CIFAR-100
Normal train	93.71%	72.88%
Train with sparsity regularization	93.66%	72.44%

Table 10
Comparison on CIFAR

Dataset	FLOPs Pruned Ratio	Accuracy
CIFAR-10	VGG-16 (Baseline)	93.80%
	Zhao et al.(39.10%)[56]	93.18%
	SSS(41.60%)[22]	93.02%
	GAL-0.05(39.60%)[34]	92.03%
	HRank(53.50%)[33]	93.43%
	VGG-16 Ours(73.68%)	93.39%
CIFAR-10	Resnet-56 (Baseline)	93.59%
	Li et al.(28.14%)[31]	93.06%
	NISP(35.96%)[53]	93.01%
	GAL-0.6(38.10%)[34]	92.98%
	HRank(50.41%)[33]	93.17%
	Resnet-56 Ours(73.51%)	93.04%
CIFAR-10	Resnet-110 (Baseline)	94.11%
	Li et al.(39.21%)[31]	93.30%
	GAL-0.5(48.94%)[34]	92.55%
	HRank(58.54%)[33]	93.36%
	Resnet-110 Ours(68.03%)	93.36%
CIFAR-100	VGG-16 Ours(54.06%)	73.21%
	VGG-16(18.05%)[56]	73.33%

We also compare our method and [31, 35] on the CIFAR-10 dataset. In the experiment, the pruning ratio is set to 10% for each iteration of pruning. The result is shown in Fig.5. It can be seen that when the FLOPs compression ratio of FLOPs is within 40%. Our method shows better performance than [35]. And with further compression, our test accuracy is slightly lower than [35]. We think this is caused by more sparsity regularization terms, although regularization can zero output some channels, so that we can distinguish unimportant channels. But it will also slightly affect the accuracy of models, as shown in Table 7 and Table 9. This can also explain why Li et al.[31] have higher accuracy when the pruning ratio is between 20% and 40%, because Li et al.[31] does not introduces sparsity regularization terms, regularization will slightly affect the performance of the model, this is also the reason why the hyperparameter of the regularization term must be less than the learning rate. And in the early stage of pruning, even if important channels are deleted by mistake, the remaining channels have enough ability to express the performance of the whole model. In further compression, Li et al.[31] ignore the subsequent transformation of the BN layers and deletes important channels, which will result in drastic decrease in accuracy.

Fig.6 shows the number of channels in convolutional layer when about 70% of FLOPs are deleted. It can be seen

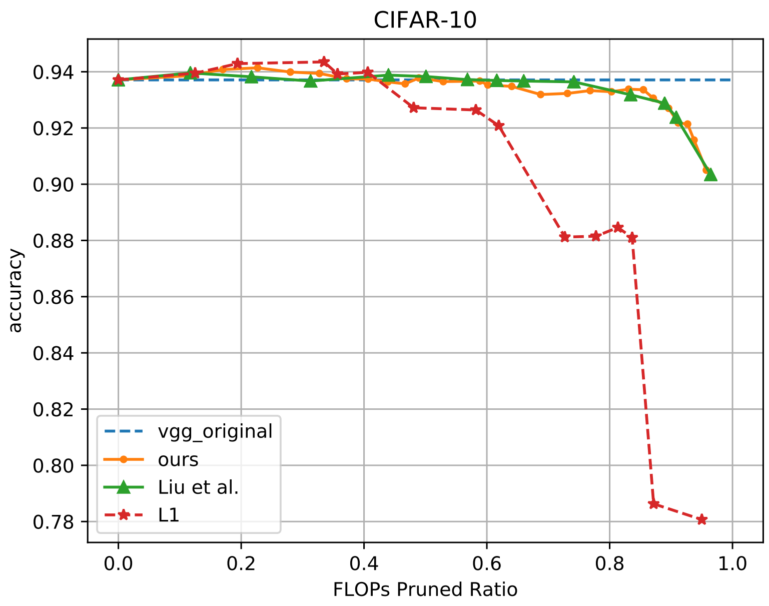


Figure 5: The accuracy of VGG19 under different pruning ratio. We compare our method and Liu et al.[35], L1[31] on the CIFAR-10 dataset.

Table 11
Comparison on ImageNet dataset

FLOPs Pruned Ratio	FLOPs	Params pruned	Params	Accuracy
VGG-11(Baseline)	150.16×10^8	-	9.74M	67.80%
HRank(6.67%)[33]	140.14×10^8	26.10%	7.19M	65.94%
VGG-11 Ours(31.95%)	102.18×10^8	11.57%	8.61M	65.72%
Li et al.(32.41%)[31]	101.48×10^8	5.79%	9.17M	65.02%
VGG-11 Ours(39.89%)	90.25×10^8	20.38%	7.75M	65.13%

that the structure retained by the three methods (the number of channels retained by the convolutional layer) is different, The difference in the number of channels reserved by Liu et al.[35] in each layer is small, while our method and Li et al.[31] tend to retain more channels in the middle. Our structure is similar to L1[31], but as shown in Fig.5, we have higher accuracy with greater pruning rate.

We use sparsity regularization training method to fine-tune the network. Compared with [35], we also introduce regularization on the weights, so there will be a slight decrease when the FLOPs compression ratio is between 60% and 80%. But we think this can be compensated by non-sparse fine-tuning, which will be explored in future work.

In [31, 35], they add more restrictions on the pruning ratio in order to maintain the accuracy of models. Especially in [31], it is necessary to perform sensitivity analysis on each layer and set a separate pruning ratio for each layer, which will take a lot of time and effort. In future work, we explore the effect of combining [31] with our method.

More experimental comparisons are shown in Table 10. For VGG-16, on Cifar-10 dataset, compared with Zhao et al.[56], SSS[22] and GAL-0.05[34], our method provides better parameters and FLOPs reduction. Compared with HRank[33], we can reduce the FLOPs by 20.18% (73.68% VS 50.50%) while only reducing the accuracy of 0.04%. In particular, on the Cifar-100 dataset, our pruning ability is also far more than Zhao et al.[56]. For Resnet-56 on Cifar-10, while the pruning rate is higher than NISP[53] and GAL-0.6[34], our model also shows higher accuracy. Compared with Li et al.[31] and HRank[33], our model reduces the accuracy of 0.02% and 0.13% respectively, but the reduction

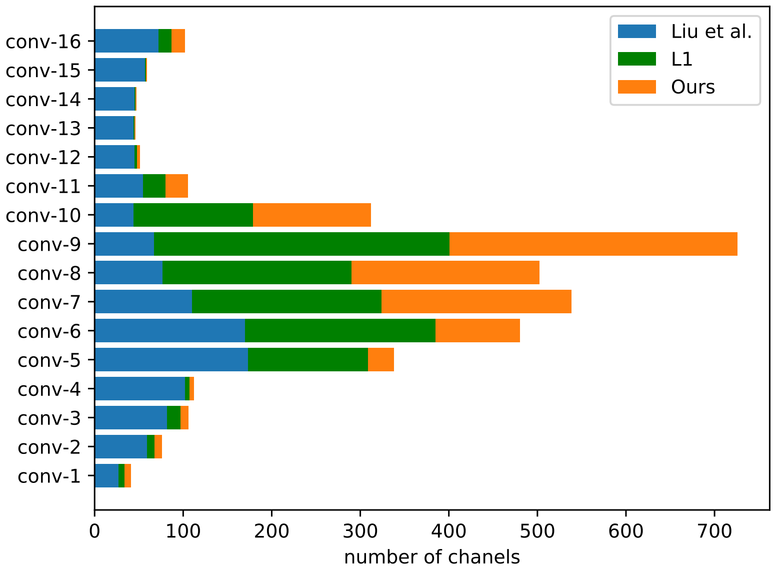


Figure 6: VGG-19 on CIFAR-10. Statistics of channels in convolutional layer (Liu et al.[35], L1[31], Ours). Our structure is similar to L1[31].

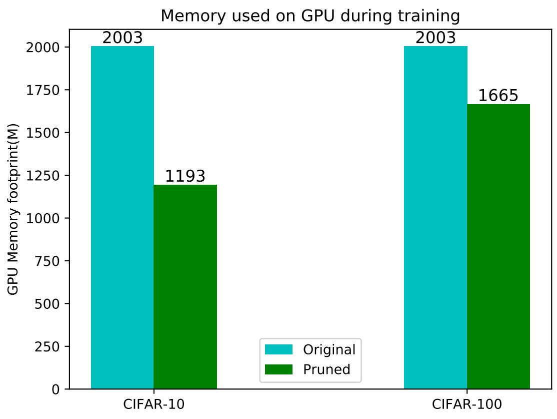


Figure 7: Memory usage on GPU during training (M). The pruned networks occupy less GPU resources.

of FLOPs is much higher than them (73.51% VS 28.14%, 73.51% VS 50.41%). For Resnet-110 on Cifar-10, we reduced the FLOPs by 28.82% (68.03% VS 39.21%) and 19.09% (68.03% VS 48.94%) more than Li et al.[31] and GAL-0.5[34], and improved the accuracy by 0.06% and 0.81%, respectively. HRank[33] can achieve the same accuracy as ours, but we can reduce the FLOPs of 68.03%, and HRank[33] can only reduce FLOPs by 58.54%. Based on the above, it can be seen that our method can better accelerate the neural network model.

4.4. Realistic Acceleration

We test the realistic acceleration of VGG-19 on the CIFAR dataset. We test the memory usage on the GPU and the inference time of an epoch. Fig.7 shows the memory usage during training on Tesla P100 GPU before and after pruning. It can be seen that the models occupy less memory resources after pruning, which is very important for some

Table 12
VGG-19 training time for one epoch on CIFAR

Dataset	FLOPs Pruned Ratio	Training time
CIFAR-10	VGG-19(Baseline)	28.09s
	VGG-19(85.50%)	16.51s
CIFAR-100	VGG-19(Baseline)	28.39s
	VGG-19(62.10%)	17.12s

devices with limited resources. As shown in Table 12, the pruned networks can greatly reduce the inference time. But the realistic acceleration result is less than the theoretical acceleration. This is completely predictable and acceptable. As analyzed in [18, 19], non-tensors such as BN layer and pooling layer also need to be inferred on GPU. And IO delay, data loading, buffer switch and the efficiency of the BLAS library will also take up a lot of time. In future work, we will look forward to combining these methods to truly achieve theoretical acceleration.

5. Conclusion and Future Work

Modern neural network models are very effective and redundant. In this paper, we propose a collaborative channel pruning method based on weights and scaling factors to compress the neural network model. We first introduce sparsity regularization on the weights and scaling factors respectively, and then jointly judge the importance of the channel with the weights and scaling factors. Finally the unimportant channels will be deleted without affecting the accuracy of the model. The experimental results demonstrate the effectiveness of proposed method. In the future, we look forward to combining other compression methods to achieve further compression.

Declaration of Competing Interest

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

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China under Grand (No.61802001).

References

[1] Acharya, U.R., Fujita, H., Oh, S.L., Hagiwara, Y., Tan, J.H., Adam, M., 2017. Application of deep convolutional neural network for automated detection of myocardial infarction using ecg signals. Information Sciences 415, 190–198. https://doi.org/10.1016/j.ins.2017.06.027.

[2] Carreira-Perpiñán, M.Á., Idelbayev, Y., 2018. "learning-compression" algorithms for neural net pruning, in: CVPR, IEEE Computer Society. pp. 8532–8541. https://ieeexplore.ieee.org/xpl/conhome/8576498/proceeding.

[3] Chen, L., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L., 2015a. Semantic image segmentation with deep convolutional nets and fully connected crfs, in: ICLR (Poster).

[4] Chen, W., Wilson, J.T., Tyree, S., Weinberger, K.Q., Chen, Y., 2015b. Compressing neural networks with the hashing trick, in: ICML, JMLR.org. pp. 2285–2294.

[5] Chen, Y., Li, C., Gong, L., Wen, X., Zhang, Y., Shi, W., 2020. A deep neural network compression algorithm based on knowledge transfer for edge devices. Computer Communications 163, 186 – 194. URL: http://www.sciencedirect.com/science/article/pii/S0140366420319368, doi:https://doi.org/10.1016/j.comcom.2020.09.016.

[6] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y., 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830 .

[7] Dai, J., Li, Y., He, K., Sun, J., 2016. R-FCN: object detection via region-based fully convolutional networks, in: NIPS, pp. 379–387.

[8] Darlow, L.N., Crowley, E.J., Antoniou, A., Storkey, A.J., 2018. Cinic-10 is not imagenet or cifar-10. arXiv preprint arXiv:1810.03505 .

[9] Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R., 2014. Exploiting linear structure within convolutional networks for efficient evaluation, in: NIPS, pp. 1269–1277.

[10] Ding, G., Zhang, S., Jia, Z., Zhong, J., Han, J., 2021. Where to prune: Using LSTM to guide data-dependent soft pruning. IEEE Trans. Image Process. 30, 293–304.

[11] Ding, X., Ding, G., Guo, Y., Han, J., 2019a. Centripetal SGD for pruning very deep convolutional networks with complicated structure, in: CVPR, Computer Vision Foundation / IEEE. pp. 4943–4953.

[12] Ding, X., Ding, G., Guo, Y., Han, J., Yan, C., 2019b. Approximated oracle filter pruning for destructive CNN width optimization, in: ICML, PMLR. pp. 1607–1616.

[13] Ding, X., Ding, G., Zhou, X., Guo, Y., Han, J., Liu, J., 2019c. Global sparse momentum SGD for pruning very deep neural networks, in: NeurIPS, pp. 6379–6391.

[14] Girshick, R.B., Donahue, J., Darrell, T., Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation, in: CVPR, IEEE Computer Society. pp. 580–587. https://doi.org/10.1109/CVPR.2014.81.

[15] Gong, Y., Liu, L., Yang, M., Bourdev, L.D., 2014. Compressing deep convolutional networks using vector quantization. CoRR abs/1412.6115.

[16] Han, S., Pool, J., Tran, J., Dally, W.J., 2015. Learning both weights and connections for efficient neural network, in: NIPS, pp. 1135–1143.

[17] He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: CVPR, IEEE Computer Society. pp. 770–778. https://doi.org/10.1109/CVPR.2016.90.

[18] He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y., 2018. Soft filter pruning for accelerating deep convolutional neural networks, in: IJCAI, ijcai.org. pp. 2234–2240. https://doi.org/10.24963/ijcai.2018/309.

[19] He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y., 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration, in: CVPR, Computer Vision Foundation / IEEE. pp. 4340–4349.

[20] He, Y., Zhang, X., Sun, J., 2017. Channel pruning for accelerating very deep neural networks, in: ICCV, IEEE Computer Society. pp. 1398–1406. https://doi.org/10.1109/ICCV.2017.155.

[21] Hinton, G.E., Vinyals, O., Dean, J., 2015. Distilling the knowledge in a neural network. CoRR abs/1503.02531. http://arxiv.org/abs/1503.02531.

[22] Huang, Z., Wang, N., 2018. Data-driven sparse structure selection for deep neural networks, in: ECCV (16), Springer. pp. 317–334. https://doi.org/10.1109/ICCV.2017.298.

[23] Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: ICML, JMLR.org. pp. 448–456.

[24] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A.G., Adam, H., Kalenichenko, D., 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: CVPR, IEEE Computer Society. pp. 2704–2713.

[25] Jaderberg, M., Vedaldi, A., Zisserman, A., 2014. Speeding up convolutional neural networks with low rank expansions, in: BMVC, BMVA Press.

[26] Krizhevsky, A., Hinton, G., et al., 2009. Learning multiple layers of features from tiny images .

[27] Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: NIPS, pp. 1106–1114.

[28] Kuang, L., 2018. Train cifar10 with pytorch.

[29] Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I.V., Lempitsky, V.S., 2015. Speeding-up convolutional neural networks using fine-tuned cp-decomposition, in: ICLR (Poster).

[30] Lemaire, C., Achkar, A., Jodoin, P., 2019. Structured pruning of neural networks with budget-aware regularization, in: CVPR, Computer Vision Foundation / IEEE. pp. 9108–9116.

[31] Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P., 2017. Pruning filters for efficient convnets, in: ICLR (Poster), OpenReview.net.

[32] Li, J., Qi, Q., Wang, J., Ge, C., Li, Y., Yue, Z., Sun, H., 2019. OICSR: out-in-channel sparsity regularization for compact deep neural networks, in: CVPR, Computer Vision Foundation / IEEE. pp. 7046–7055.

[33] Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., Shao, L., 2020. Hrank: Filter pruning using high-rank feature map, in: CVPR, IEEE. pp. 1526–1535.

[34] Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F., Doermann, D.S., 2019. Towards optimal structured CNN pruning via generative adversarial learning, in: CVPR, Computer Vision Foundation / IEEE. pp. 2790–2799.

[35] Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C., 2017. Learning efficient convolutional networks through network slimming, in: ICCV, IEEE Computer Society. pp. 2755–2763. https://doi.org/10.1109/ICCV.2017.298.

[36] Long, J., Shelhamer, E., Darrell, T., 2015. Fully convolutional networks for semantic segmentation, in: CVPR, IEEE Computer Society. pp. 3431–3440. https://doi.org/10.1109/CVPR.2015.7298965.

[37] Luo, J., Wu, J., Lin, W., 2017. Thinet: A filter level pruning method for deep neural network compression, in: ICCV, IEEE Computer Society. pp. 5068–5076. https://doi.org/10.1109/ICCV.2017.541.

[38] Mishra, A.K., Marr, D., 2018. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy, in: ICLR (Poster), OpenReview.net.

[39] Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J., 2019. Importance estimation for neural network pruning, in: CVPR, Computer Vision Foundation / IEEE. pp. 11264–11272. http://openaccess.thecvf.com/CVPR2019.py.

[40] Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J., 2017. Pruning convolutional neural networks for resource efficient inference, in: ICLR (Poster), OpenReview.net. https://openreview.net/forum?id=SJGCiw5gl.

[41] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y., 2011. Reading digits in natural images with unsupervised feature learning .

[42] Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A., 2016. Xnor-net: Imagenet classification using binary convolutional neural networks, in: ECCV (4), Springer. pp. 525–542. https://doi.org/10.1007/978-3-319-46493-0_32.

[43] Ren, S., He, K., Girshick, R.B., Sun, J., 2015. Faster R-CNN: towards real-time object detection with region proposal networks, in: NIPS, pp. 91–99.

[44] Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y., 2015. Fitnets: Hints for thin deep nets, in: ICLR (Poster). http://arxiv.org/abs/1412.6550.

[45] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.S., Berg, A.C., Li, F., 2015. Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. 115, 211–252.

[46] Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L., 2016. Edge computing: Vision and challenges. IEEE Internet Things J. 3, 637–646.

[47] Simonyan, K., Zisserman, A., 2015. Very deep convolutional networks for large-scale image recognition, in: ICLR. http://arxiv.org/abs/1409.1556.

[48] Sutskever, I., Martens, J., Dahl, G.E., Hinton, G.E., 2013. On the importance of initialization and momentum in deep learning, in: ICML (3), JMLR.org. pp. 1139–1147.

[49] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S.E., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2015. Going deeper with convolutions, in: CVPR, IEEE Computer Society. pp. 1–9. https://doi.org/10.1109/CVPR.2015.7298594.

[50] Wang, Y., Han, C., Yao, G., Zhou, W., 2020. Mapd:an improved multi-attribute pedestrian detection in a crowd. Neurocomputing URL: http://www.sciencedirect.com/science/article/pii/S0925231220318865, doi:https://doi.org/10.1016/j.neucom.2020.12.005.

[51] Wu, J., Leng, C., Wang, Y., Hu, Q., Cheng, J., 2016. Quantized convolutional neural networks for mobile devices, in: CVPR, IEEE Computer Society. pp. 4820–4828. https://doi.org/10.1109/CVPR.2016.521.

[52] Yim, J., Joo, D., Bae, J., Kim, J., 2017. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning, in: CVPR, IEEE Computer Society. pp. 7130–7138. https://doi.org/10.1109/CVPR.2017.754.

[53] Yu, R., Li, A., Chen, C., Lai, J., Morariu, V.I., Han, X., Gao, M., Lin, C., Davis, L.S., 2018. NISP: pruning networks using neuron importance score propagation, in: CVPR, IEEE Computer Society. pp. 9194–9203.

[54] Zagoruyko, 2016. 92.5% on cifar-10 in torch. https://github.com/szagoruyko/cifar.torch.

[55] Zhang, X., Zou, J., Ming, X., He, K., Sun, J., 2015. Efficient and accurate approximations of nonlinear convolutional networks, in: CVPR, IEEE Computer Society. pp. 1984–1992. https://doi.org/10.1109/CVPR.2015.7298809.

[56] Zhao, C., Ni, B., Zhang, J., Zhao, Q., Zhang, W., Tian, Q., 2019. Variational convolutional neural network pruning, in: CVPR, Computer Vision Foundation / IEEE. pp. 2780–2789.