Towards Optimal Filter Pruning with Balanced Performance and Pruning Speed

Dong Li[1,2](✉), Sitong Chen[2], Xudong Liu[2], Yunda Sun[2], and Li Zhang[1,2]

[1] Tsinghua University, Beijing 100084, China
[2] Nuctech AI R&D Center, Beijing 100084, China
{li.dong,chensitong,liuxudong,sunyunda,zhangli}@nuctech.com

Abstract. Filter pruning has drawn more attention since resource constrained platform requires more compact model for deployment. However, current pruning methods suffer either from the inferior performance of one-shot methods, or the expensive time cost of iterative training methods. In this paper, we propose a balanced filter pruning method for both performance and pruning speed. Based on the filter importance criteria, our method is able to prune a layer with approximate layer-wise optimal pruning rate at preset loss variation. The network is pruned in the layer-wise way without the time consuming prune-retrain iteration. If a pre-defined pruning rate for the entire network is given, we also introduce a method to find the corresponding loss variation threshold with fast converging speed. Moreover, we propose the layer group pruning and channel selection mechanism for channel alignment in network with short connections. The proposed pruning method is widely applicable to common architectures and does not involve any additional training except the final fine-tuning. Comprehensive experiments show that our method outperforms many state-of-the-art approaches.

1 Introduction

Despite the fact that neural network based approaches have achieved significant performance improvement in many computer vision tasks, the deployment of these over-parameterized model often requires high computing power and large memory footprint, which are not available on resource constrained platform such as mobile phone. To tackle this problem, researchers propose different methods for network compression and inference acceleration, including lightweight architecture designing [1,2], network pruning [3–5], weight quantization [6,7], matrix factorization [8], knowledge distillation [9], etc.

D. Li, S. Chen—Contributed equally to this work.

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-030-69538-5_16) contains supplementary material, which is available to authorized users.

Among these methods, network pruning has drawn much attention since it is able to reduce the number of model parameters and operations simultaneously. It can be categorized as structure pruning and non-structure pruning. Non-structure pruning sets unimportant weights to zero to achieve high sparsity [10,11], while sparse operation requires specialized hardware [12] or software [13] libraries to speed up the inference process, which limits the usage of the pruned network. Structure pruning is also recognized as filter pruning or channel pruning since it is implemented by removing filters in the original network. This coarse-grained filter-level pruning can be treated as modification to the network architecture, so it does not damage the usability of the model. In this paper, we propose a filter pruning method to shrink network size and accelerate its inference at the same time.

The key issue of filter pruning is selecting the unimportant filters to be pruned at a given compression ratio. To solve this combinatorial optimization problem, most methods evaluate the importance of filters then either prune them in a one-shot manner, or iteratively prune-retrain the model. On the one hand, one-shot approaches often prune filters in each layer based on some pre-defined prune rate and particular properties of the trained model [5,14–18], which are more prone to over-pruning or under-pruning at certain layers. On the other hand, iterative pruning based on greedy criteria increases the time cost and computation burden [17,19,20]. Also, some of these methods jointly optimize original objective function with compression, thus the loss function becomes more complex and difficult to converge due to the hyper-parameters introduced. Filter pruning is by far an unsolved problem, since the optimal prune rate of each layer is hard to obtain.

Our approach is a balanced method which is able to approximate the layer-wise optimal pruning rate with limited time and computing resource. Given a trained convolutional network, we observe that removing a convolution kernel in certain layer leads to different changes of loss function and accuracy, while the accuracy drop has a highly positive correlation with the absolute value of loss function change, which we denote it as the loss variation. Based on the criteria of gradient and magnitude of filters, the contribution to loss variation of different filters can be accurately estimated. As these gradients are able to compute by back propagation, we select batches of data to evaluate the importance of each filter per layer by inferences within single epoch. We propose an algorithm to obtain the maximum pruning rate in each layer, constrained by a threshold of loss variation. We use binary search to find the combination of filters which are sorted by importance, so the maximized number of filters can be pruned in one-shot. After all layers are done, the pruned model is fine-tuned only once. To verify the effectiveness of our method, we conduct a series of filter pruning experiments using CIFAR-10 [21] and ImageNet [22] dataset. Our result outperforms the state-of-the-art algorithms with many major network architectures, including VGG [23], GoogLeNet [24], ResNet [25], DenseNet [26], etc.

In summary, our main contribution is the proposed filter pruning method to approximately obtain layer-wise optimal pruning rate, which is able to prune a layer with maximum pruning rate at given loss variation without the time consuming prune-retrain iteration. For pre-defined pruning rate of entire network, our method is able to converge to the particular pruning rate without additional fine-tuning. We introduce binary search to help the layer-wise pruning and the global pruning rate converging, so that our method balances the performance of pruned network and pruning speed. The proposed method is widely applicable to common architectures of convolutional networks. Comprehensive experiments show that our method is able to achieve higher compression ratio with lower accuracy drop compared with the state-of-the-art approaches.

2 Related Work

Network pruning obtains a more compact model by removing redundant connections from the original network, thereby reducing the number of parameters and operations. Early researches on this topic are mainly addressed by removing weight-level connections for sparse pruning. Since the applicability of sparse network is limited, recent works are more focused on structure pruning methods, which can be further categorized as one-shot filter pruning and iterative filter pruning.

Sparse Pruning. Inspired by neurobiology, the optimal brain damage [3] and the optimal brain surgeon [4] removed unimportant connections according to the analysis of Hessian matrix of the loss function. Han et al. [10] determined the importance of weights in the network through the weight value, and reduce the redundancy by deleting smaller weights. Srinivas [11] proposed a data-free method to remove the redundant parameters of the fully-connected layer. Because of the sparsity of the weight tensor, these unstructured pruned model only accelerate the inference process on specialized platforms.

One-Shot Filter Pruning refers to all redundant filters in a network are pruned before fine-tuning. Some methods estimated the importance of filters based on the characteristics of the filter itself, including L1 norms [5], geometric median [17], etc. Others evaluated the filter redundancy by analyzing the information of the feature map. Hu et al. [14] used the sparsity of the output of each layer to choose the redundant filter. He et al. [16] used the least square to reconstruct the error and LASSO regression to remove filters layer by layer. Luo et al. [15] pruned filters based on the statistical information of next layer. Yu et al. [27] proposed a method based on importance score propagation, which back-propagates the score of the final response layer to each filter to determine whether the filter is redundant. However, these methods usually depend on a heuristic metric to set the pruning rate of each layer in advance. Although one-shot pruning algorithm is capable to reduce time cost, it is prone to suffer from inferior compression ratio and accuracy.

Iterative Filter Pruning selectively prune one or more filters followed by training to recover the model performance in each iteration. Liu et al. [28] performs sparse training on the scale factor of BN, and removes the filter with a smaller scale factor according to the pruning rate corresponding to each layer. Molchanov et al. [29] proposed a criterion based on Taylor expansion to evaluate the importance of filters, then applied with greedy pruning strategy. Reinforcement learning was introduced in AMC [19] for pruning, it set rewards by constraining FLOPs, accuracy and specific compression ratios in continuous space. You et al. [30] proposed the Gate Batch Normalization module, and added the FLOPs hyper-parameters to the training objective to compress the model. Huang et al. [20] proposed a data-driven method to learn the architecture of the network, introduced a new scaling factor and corresponding sparse regularization, and defined pruning as a joint sparse regularization optimization problem. Lin et al. [31] added a mask to each filter and obtained the final model by generative adversarial learning. The dynamic pruning scheme [32] globally pruned unimportant filters and adjusted the network dynamically, with a mechanism to restore the filters that were mis-pruned. Instead of pruning negligible filters, recent work [33] proposed an optimization objective to generate multiple identical filters then remove them to achieve pruning goals. A major drawback for iterative pruning is the extensive computational burden. Additionally, the pruning strategies based on training iterations often change the optimization function, and even introduce a large number of hyper-parameters, which will make the training more difficult to converge.

3 Our Method

3.1 Filter Importance Evaluation

Given a trained network, we randomly select a group of filters and set their weights to zero, then we use the pruned network to forward all the samples to calculate loss variation and accuracy drop. It is observed that the accuracy drop is almost directly proportional to loss variation. Figure 1 has shown the correlation between loss variation and accuracy drop for VGG16 trained on CIFAR-10.

For computer vision application, the majority of trained models are convolutional neural networks. We denote the dataset $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\}$ consists of N samples, $\mathcal{X} = \{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_N\}, \mathcal{Y} = \{y_0, y_1, \cdots, y_N\}$, where $\mathbf{x}_i$ and y_i are image and label of i-th sample, respectively. In a trained network with L layers, the filters can be parameterized as $\mathcal{W} = \{\mathbf{w}_1^{(1)}, \mathbf{w}_1^{(2)}, \cdots, \mathbf{w}_L^{(C_L)}\}$, i.e. for k-th filter in l-th layer, the weights are $\mathbf{w}_l^{(k)} \in \mathbb{R}^{C_{l-1} \times p \times p}$ with $l \in [1, 2, \cdots, L]$ and $k \in [1, 2, \cdots, C_l]$, where C_l represents the number of channels in l-th layer. We denote the pruned network as $\mathcal{W}'$ which sets a subset of filters of $\mathcal{W}$ to zero, i.e. $\mathbf{w}_l^{(k)} = \mathbf{0}$ represents the k-th filter of l-th layer is pruned. Since the accuracy drop is directly related to the loss variation, the filter pruning can be defined as the optimization problem:

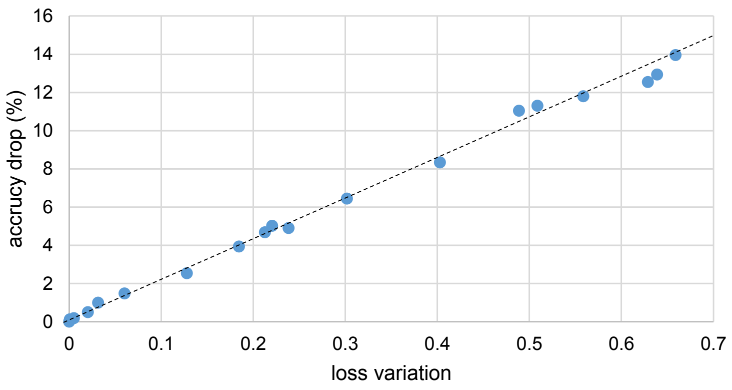


Fig. 1. Correlation between loss variation and accuracy drop. Samples are collected from pruning results of VGG16 trained on CIFAR-10.

$$\begin{aligned} &\min_{\mathcal{W}'} \left|\mathcal{L}\left(\mathcal{D};\mathcal{W}'\right)-\mathcal{L}(\mathcal{D};\mathcal{W})\right| \\ &\text{s.t.} \quad \left\|\mathcal{W}'\right\|_0 \leqslant \beta \left\|\mathcal{W}\right\|_0 \end{aligned} \tag{1}$$

where $\mathcal{L}(\cdot)$ is the loss function and $\gamma = 1-\beta$ is the specific pruning rate. Solving combinatorial optimization problem (1) is impractical for modern networks, so we evaluate the importance of filters by certain criterion, then use it as the prior knowledge for pruning. The saliency of single filter can be evaluated by calculating the loss variation on the dataset after pruning:

$$\Delta\mathcal{L}(\mathcal{D};\mathcal{W},\mathbf{w}_l^{(k)}=\mathbf{0}) = \left|\mathcal{L}(\mathcal{D};\mathcal{W},\mathbf{w}_l^{(k)}=\mathbf{0}) - \mathcal{L}(\mathcal{D};\mathcal{W})\right| \tag{2}$$

we have first-order approximation by Taylor expansion:

$$\left|\mathcal{L}(\mathcal{D};\mathcal{W},\mathbf{w}_l^{(k)}=\mathbf{0}) - \mathcal{L}(\mathcal{D};\mathcal{W})\right| \approx \left|\frac{\partial\mathcal{L}(\mathcal{D};\mathcal{W})}{\partial\mathbf{w}_l^{(k)}}\mathbf{w}_l^{(k)}\right| = \boldsymbol{G}_l^{(k)} \tag{3}$$

In this paper, $\boldsymbol{G}_l^{(k)}$ is used as the criterion to evaluate the importance of k-th filter in l-th layer. To prune a single filter from a network, the least value of (3) of all the filters is selected. It is consistent with the intuition that the filter with smaller gradient and magnitude should be pruned first.

3.2 Layer-Wise Optimal Pruning Rate Searching

The value of $\boldsymbol{G}_l^{(k)}$ is calculated once by forwarding and backwarding all samples from the dataset. In greedy-based methods, the filters are sorted by importance and the unimportant filters are pruned together by their ranks. We propose a layer-wise pruning method to improve the sub-optimal solution caused by cross-layer greedy strategies.

Algorithm 1. Optimal pruning rate searching in l-th layer

```
1: Input: original network weights $\mathcal{W}$, $\{score_k\}$ with $k \in [1, 2, \cdots, C_l]$, threshold $\theta$,
   original network loss $\varphi$
2: Output: new weights $\mathcal{W}'$ with $l$-th layer pruned
3: $rank \leftarrow C_l/2$
4: $step \leftarrow rank$
5: $\{index_k\} \leftarrow \text{sort}(\{score_k\})$    ▷ in ascending order, $score_{index_1}$ is the smallest
6: $prune_{id} \leftarrow \{\}$
7: while True do
8:     if $step < 1$ then
9:         break
10:    end if
11:    Pruning: $\mathbf{w}_l^{(i)} \leftarrow \mathbf{0}$ with $i \in [index_1, index_2, \cdots, index_{rank}]$
12:    Forward all samples to compute the loss $\varphi'$ with $l$-th layer pruned
13:    $step \leftarrow step/2$
14:    if $|\varphi' - \varphi| > \theta$ then
15:        $rank \leftarrow rank - step$
16:    else
17:        $prune_{id} \leftarrow \{index_k\}$ with $k \in [1, 2, \cdots, rank]$
18:        $rank \leftarrow rank + step$
19:    end if
20: end while
21: Pruning: $\mathbf{w}_l^{(k)} \leftarrow \mathbf{0}$ for $k \in prune_{id}$
22: Return $\mathcal{W}'$
```

Suppose there are $N = M \times P$ samples in the dataset $\mathcal{D}$, where M is the batch size and P is the number of batches. In l-th layer, we have $score_k$ to represent the importance of k-th filter evaluated on the dataset.

$$score_k = \sum_{i=1}^{P} \frac{z_{ik}}{C_l} \tag{4}$$

In (4), C_l is the number of filters in l-th layer, z_{ik} is the index of k-th filter in ascending order after i-th batch running and sorting for $\boldsymbol{G}_l^{(k)}$. As Sect. 3.1 mentioned, the decrease of model accuracy is consistent with the loss variation, so we can use the loss variation as a hyper-parameter for the layer-wise pruning. We treat it as a search problem, aiming to find maximum number of filters to be pruned per layer to achieve high compression ratio within the loss variation range. We define the original trained network loss as $\varphi = \mathcal{L}(\mathcal{D}; \mathcal{W})$ and introduce a parameter θ that represents the threshold of the loss variation. To accelerate the pruning, we use binary search to avoid the re-evaluation of filter pruning one by one. The algorithm that searches for the optimal pruning rate in one layer is described in Algorithm 1. The ablation study for Algorithm 1 indicates that the binary search speeds up the pruning process by 5–10×.

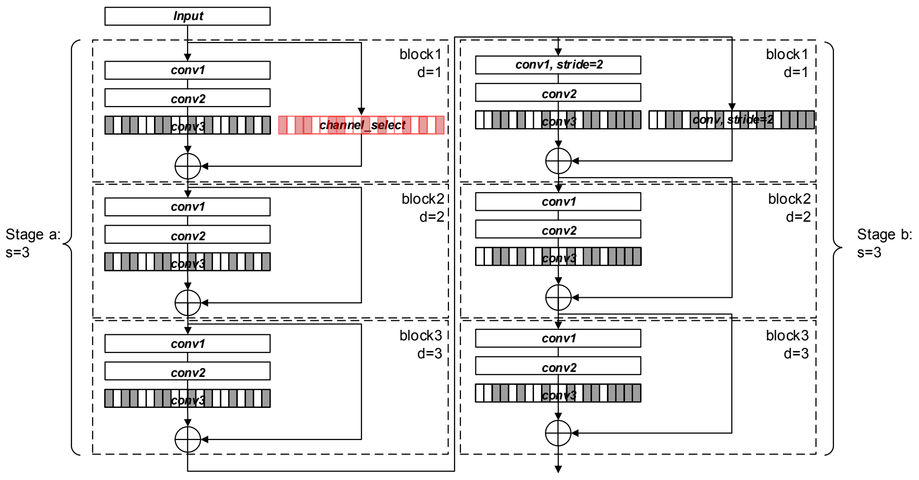


Fig. 2. Group pruning method for shortcut connections in ResNet-like networks. For one stage, the last layers of each block are pruned together in a group such that they have the same retained filters. Gray color indicates the corresponding filter is pruned. Since stage a has 3 blocks and the shortcut in each block is an identity connection, a channel_select layer (in red) is introduced to mask out the pruned channels of input; while stage b has 3 blocks with first shortcut connected by convolutional downsampling, so it can be pruned in a group. (Color figure online)

For network architectures with shortcut connections, such as ResNet, the output channels of the last convolutional layers of each block must be the same, as shown in Fig. 2. It is because that the shortcut connections require the output channels of these layers are aligned. Therefore, we put the last layers of each block in one stage together as a group for pruning. We define d as the depth of each block in one stage, s denotes the maximum value of d. Empirically, we find that the lower-level convolution kernels tend to have weaker representative capability than the high-level ones, so the scoring weight of the lower-level convolution kernels are reduced. The weighted sum for group pruning is conducted by Eq. 5:

$$score_k = \sum_{i=1}^{K} \frac{d_i}{s} \cdot score_{k,d_i} \tag{5}$$

where d_i represents depth of i-th block in one stage, with $i \in [1, 2, \cdots, K]$ and K is the number of blocks. During pruning, the filters with the same index in the group will be pruned together according to $score_k$ ranking. For some stages, the input channels are directly short connected to blocks, hence we introduce a non-parametric layer for channel selection, whose output channel layout is copied from the group pruning result. After the layer-wise optimal pruning rate searching, the layer-wise results are connected for the pruned network, then it is fine-tuned on the training dataset to obtain the final model.

Algorithm 2. Filter pruning with global pruning rate γ

```
1: **Input:** original network weights $\mathcal{W}$, $\gamma$, initial value of threshold $\theta_{init}$, pruning rate
   tolerance $\varepsilon$
2: **Output:** pruned network weights $\mathcal{W}'$ with global pruning rate $\gamma'$, s.t. $|\gamma' - \gamma| \leqslant \varepsilon$
3: $\theta_{upper} \leftarrow \theta_{init}$
4: $\theta_{lower} \leftarrow 0$
5: $\gamma' \leftarrow 0$
6: Forward/backward network $\mathcal{W}$ on $\mathcal{D}$ once, compute original loss $\varphi$ and $\{score_k\}_l$
   for $l \in [1, 2, \cdots, L]$ and $k \in [1, 2, \cdots, C_l]$          ▷ Eq. 3 and 4.
7: **while** $|\gamma' - \gamma| > \varepsilon$ **do**
8:     $\mathcal{W}' \leftarrow \mathcal{W}$
9:     **for all** $l$ in $L$ layers **do**
10:        run pruning Alg. 1 with ($\mathcal{W}$, $\{score_k\}_l$, $\theta_{upper}$, $\varphi$)
11:        update $\mathcal{W}'$ with pruned $l$-th layer
12:    **end for**
13:    $\gamma' \leftarrow 1 - \|\mathcal{W}'\|_0 \,/\, \|\mathcal{W}\|_0$
14:    $step \leftarrow \theta_{upper} - \theta_{lower}$
15:    **if** $\gamma' > \gamma$ **then**
16:        $\theta_{upper} \leftarrow (\theta_{upper} + \theta_{lower})/2$
17:    **else**
18:        $\theta_{lower} \leftarrow \theta_{upper}$
19:        $\theta_{upper} \leftarrow \theta_{upper} + 2 \times step$
20:    **end if**
21: **end while**
22: Finetune the pruned network $\mathcal{L}(\mathcal{D}; \mathcal{W}')$ once.
23: **Return** $\mathcal{W}'$, $\gamma'$, $\theta_{upper}$
```

3.3 Pruning with Global Constraints

In many cases, there are constraints for the whole network, such as global pruning rate γ defined in (1), performance loss, etc. Here we take pre-defined pruning rate γ as the global constraint. We find that there is a positive correlation between the loss variation and the layer-wise pruning rate, it implies that the loss variation also positively correlated to the global pruning rate. More details of the statistics are listed our supplementary material. We propose a similar binary search method for the loss variation threshold θ, thereby approximate the global pruning constraint γ. The algorithm is described in Algorithm 2.

Compared to iterative training based pruning methods, our approach only needs fine-tuning the entire network for once. Although pruning with global constraint requires binary search of the loss variation threshold, it only involves multiple times of forwarding, which runs much faster than multiple times of re-training. As the following experiments indicate, our method is able to achieve the balance between performance and pruning speed.

4 Experiments

We evaluate our algorithm on two commonly used datasets (CIFAR-10 and ImageNet) with popular networks implemented by PyTorch [34]. The pruning results and performance are compared with several state-of-the-art algorithms in recent years.

4.1 Experiments Settings

Experiments Setup for CIFAR-10. CIFAR-10 is a 10-class image classification dataset containing 50,000 training images and 10,000 test images. In our experiments, we apply VGG16_BN with a plain structure [23], GoogLeNet with Inception module [24], ResNet-56/110 with residual module [25] and DenseNet-40 with dense connections [26] to verify the effectiveness of our algorithm. For the ResNet-56/110 network on CIFAR-10, the first shortcut of each stage (excluding the first stage) is a downsample and data-filling layer. In order to ensure the channels are aligned, the channel selection layer is added as shown in Fig. 2. In the fine-tuning phase, we use a NVIDIA Tesla V100 GPU to train the pruned model. We solve the optimization problem by SGD with a Nesterov momentum of 0.9 and weight decay of 1e–4. The network is trained for 400 epochs. The initial learning rate is 0.01, it is decayed by a factor of 10 every 100 epochs. The batch size is 128.

Experiments Setup for ImageNet. ImageNet [22] is a large image dataset with 1000 classes, containing 1,281,167 training images and 50,000 validation images. In the experiments, we use ResNet-50 to demonstrate our pruning performance on two NVIDIA Tesla V100 GPUs. In the fine-tuning phase, the optimizer parameters are set to be the same as the parameters in CIFAR-10 experiments. The pruned network is fine-tuned for 120 epochs with batch size 256. The initial learning rate is 0.001 and divided by 10 every 30 epochs.

4.2 Results on CIFAR-10

VGG16_BN. The performance of different compression algorithms are shown in Table 1. PR denotes the pruning rate and FLOPs denotes floating point operations. Ours-0.12 indicates that the threshold of loss variation is 0.12. Compared with L1, SSS, GAL-0.1, and HRank-A, Ours-0.12 has clear advantages for both FLOPs and parameters. Ours-0.12 reduces FLOPs by 70.29% and deletes 87.72% of the parameters, while its Top-1 accuracy keeps almost the same as the baseline. For Ours-0.2, although the reductions of FLOPs and parameters are almost the same as those of HRank-B, the Top-1 accuracy is 2.36% higher than that of HRank-B.

Table 1. The pruning results on CIFAR-10. L1*, SSS* and ApoZ* are the results in GAL.

Model	Method	Top-1 (%)	FLOPs (PR)	Parameters (PR)
VGG16_BN	Baseline	93.96	313.73M (0.00%)	14.98M (0.00%)
	L1* [5]	93.40	206.00M (34.34%)	5.40M (63.95%)
	SSS* [20]	93.02	183.13M (41.63%)	3.93M (73.76%)
	GAL-0.1 [31]	93.42	171.89M (45.21%)	2.67M (82.17%)
	HRank-A [35]	93.43	145.61M (53.59%)	2.51M (83.24%)
	Ours-0.12	**93.95**	**93.22M (70.29%)**	**1.84M (87.72%)**
	HRank-B [35]	91.23	73.70M (76.51%)	1.78M (88.12%)
	Ours-0.2	**93.59**	**73.81M (76.47%)**	**1.45M (90.32%)**
GoogLeNet	Baseline	95.05	1.52B (0.00%)	6.15M (0.00%)
	L1* [5]	94.54	1.02B (32.89%)	3.51M (42.93%)
	Random	94.54	0.96B (36.84%)	3.58M (41.79%)
	GAL-0.05 [31]	94.56	0.94B (38.16%)	3.12M (49.27%)
	ApoZ* [14]	92.11	0.76B (50.00%)	2.85M (53.66%)
	HRank-A [35]	94.53	0.69B (54.60%)	2.74M (55.45%)
	Ours-0.0045	**95.19**	**0.57B (62.50%)**	**1.76M (71.38%)**
	HRank-B [35]	94.07	0.45B (70.39%)	1.86M (69.76%)
	Ours-0.01	**94.77**	**0.40B (73.68%)**	**1.14M (81.46%)**
DenseNet-40	Baseline	94.81	282.92M (0.00%)	1.04M (0.00%)
	Liu et al.-40% [28]	94.81	190.00M (32.84%)	0.66M (36.54%)
	GAL-0.01 [31]	94.61	182.92M (35.34%)	0.67M (35.58%)
	HRank-A [35]	94.24	167.41M (40.82%)	0.66M (36.54%)
	Zhao et al. [36]	93.16	156.00M (44.86%)	0.42M (59.62%)
	Ours-0.02	**94.61**	**154.34M (45.45%)**	**0.59M (43.27%)**
	HRank-B [35]	93.68	110.15M (61.07%)	0.48M (53.85%)
	Ours-0.04	**93.49**	**95.69M (66.18%)**	**0.37M (64.42%)**
ResNet-56	Baseline	93.26	125.49M (0.00%)	0.85M (0.00%)
	L1* [5]	93.06	90.90M (27.56%)	0.73M (14.12%)
	NISP [27]	93.01	81.00M (35.45%)	0.49M (42.35%)
	HRank-A [35]	93.17	62.72M (50.02%)	0.49M (42.35%)
	He et al. [16]	90.80	62.00M (50.59%)	-
	Ours-0.019	**93.64**	**59.84M (52.31%)**	**0.52M (38.82%)**
	GAL-0.8 [31]	91.58	49.99M (60.16%)	0.29M (65.88%)
	HRank-B [35]	90.72	32.52M (74.08%)	0.27M (68.24%)
	Ours-0.055	**91.54**	**25.72M (79.50%)**	**0.25M (70.59%)**
ResNet-110	Baseline	93.50	252.89M (0.00%)	1.72M (0.00%)
	L1* [5]	93.30	155.00M (38.71%)	1.16M (32.56%)
	GAL-0.5 [31]	92.55	130.20M (48.52%)	0.95M (44.77%)
	HRank [35]	93.36	105.70M (58.20%)	0.70M (59.30%)
	Ours-0.007	**93.73**	**98.04M (61.23%)**	**0.89M (48.26%)**

ResNet56/110. The results for ResNet56/110 are shown in Table 1. Firstly, we look into the result of ResNet56. Compared with L1, Ours-0.019 obtains more FLOPs and parameters reductions with higher Top-1 accuracy, and even its accuracy is 0.38% higher than the baseline. Although Ours-0.019 has a slightly lower pruning rate than NISP and HRank-A, it achieves larger reductions in FLOPs (52.31% vs. 35.45% by NISP and 52.31% vs. 50.02% by HRank-A) and better Top-1 accuracy (93.64% vs. 93.01% by NISP and 93.47% vs. 93.17% by HRank-A). Therefore, it can be verified that our method is able to greatly reduce the amount of calculation and memory footprint while achieves better model performance. From the result, Ours-0.055 can obtain a network with higher compression ratio, its FLOPs is reduced by 79.50%, parameters are pruned by 70.59%, at the cost of Top-1 accuracy drop by only 1.72%. Compared with GAL-0.8, Ours-0.055 achieves much higher pruning rate of FLOPs and parameters though their accuracies are almost the same. Meanwhile, it outperforms HRank-B in all three aspects.

Next, we analyze the result of ResNet110. Ours-0.007 leads to an improvement in Top-1 accuracy over the baseline model (93.73% vs. 93.50%) with 61.23% FLOPs and 48.26% parameters reductions. Its performance is significantly better than L1 and GAL-0.5. Compared with HRank, Ours-0.007 achieves higher reduection rate of FLOPs (61.23% by Ours-0.007 vs. 58.20% by HRank) and better accuracy (93.73% by Ours-0.007 vs. 93.36% by HRank), although the parameter pruning rate is lower.

DenseNet-40. Table 1 summarizes the result of DenseNet-40. Although Liu et al. [28] retains the same accuracy as the baseline, the compression ratio is relatively low, reducing FLOPs by only 32.84%. For Ours-0.02, 45.45% of FLOPs and 43.27% of the parameters are reduced, and the decrease of accuracy is only 0.20%. Ours-0.02 achieves a better performance compared with HRank-A and Zhao et al. [36]. Compared with GAL-0.01, Ours-0.02 has a big gain on both FLOPs and parameters pruning rate with the same Top-1 accuracy. For Ours-0.04, the Top-1 accuracy is 0.19% lower than that of HRank-B, but our method obtains more reductions of FLOPs and parameters.

GoogleNet. The results of GoogleNet are shown in Table 1. Ours-0.0045 obtains 95.19% Top-1 accuracy, which is even 0.14% higher than the baseline, and 64.47% of FLOPs and 73.17% of parameters are removed. It outperforms L1, Random, GAL-0.05, APoZ and HRank-A. Furthermore, we set the threshold to 0.01 to increase the pruning rate of the network. Ours-0.01 achieves a better performance than HRank-B (94.77% acc vs. 94.07% by HRank-B, 73.68% reduction of FLOPs vs. 70.39% by HRank-B, 81.46% reduction of parameters vs. 69.76% by HRank-B).

4.3 Results on ImageNet

Experiments are also conducted on the ImageNet dataset using ResNet50, and the results are shown in Table 2. As indicated by the results, our method has achieved a significant gain on both performance and compression ratios compared with several state-of-the-art methods. Specifically, we set the thresholds to 0.05, 0.09, 0.2 and 0.35 respectively to obtain different pruning rates. Ours-0.05 outperforms GAL-0.5, SSS-26 and HRank-A. It removes 43.03% FLOPs from baseline, while still yields 75.79% Top-1 accuracy and 92.82% Top-5 accuracy, improves the result of SSS-32 [20] and He et al. [16] by a large margin. In addition, Ours-0.09 achieves 75.04% Top-1 accuracy and 92.29% Top-5 accuracy with

53.30% and 40.16% reductions of FLOPs and parameters, respectively. Moreover, compared to GDP-0.6, GAL-0.5-joint, GAL-1, GDP-0.5 and HRank-B, Ours-0.2 has apparent advantages in all aspects, including Top-1/Top-5 accuracy as well as FLOPs and parameters reductions. For Ours-0.35, 75.80% FLOPs and 68.55% parameters are removed, its 70.58% Top-1 accuracy and 90.00% Top-5 accuracy are significantly better than those of GAL-1-joint and ThiNet-50. Compared with HRank-C, Ours-0.35 achieves higher Top-1 and Top-5 accuracy with the similar FLOPs and parameters reductions. Therefore, the ImageNet experiments indicate that our method also works well on large and complex datasets.

Table 2. Pruning results of Resnet-50 on ImageNet.

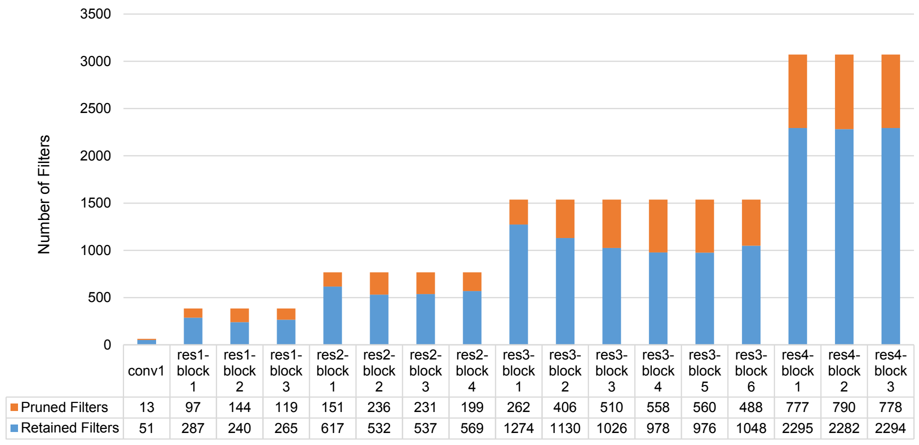
Method	Top-1 (%)	Top-5 (%)	FLOPs (PR)	Parameters (PR)
Baseline	76.15	92.87	4.09B (0.00%)	25.50M (0.00%)
SSS-32 [20]	74.18	91.91	2.82B (31.05%)	18.60M (27.06%)
He et al. [16]	72.30	90.80	2.73B (33.25%)	–
Ours-0.05	**75.79**	**92.82**	**2.33B (43.03%)**	**17.93M (29.69%)**
GAL-0.5 [31]	71.95	90.94	2.33B (43.03%)	21.20M (16.86%))
SSS-26 [20]	71.82	90.79	2.33B (43.03%)	15.60M (38.82%)
HRank-A [35]	74.98	92.33	2.30B (43.76%)	16.15M (36.67%)
Ours-0.09	**75.04**	**92.29**	**1.91B (53.30%)**	**15.26M (40.16%)**
GDP-0.6 [32]	71.19	90.71	1.88B (54.03%)	–
GAL-0.5-joint [31]	71.80	90.82	1.84B (55.01%)	19.31M (24.27%)
GAL-1 [31]	69.88	89.75	1.58B (61.37%)	14.67M (42.47%)
GDP-0.5 [32]	69.58	90.14	1.57B (61.61%)	–
HRank-B [35]	71.98	91.01	1.55B (62.10%)	13.77M (46.01%)
Ours-0.2	**73.06**	**91.30**	**1.31B (67.97%)**	**10.84M (57.49%)**
GAL-1-joint [31]	69.31	89.12	1.11B (72.86%)	10.21M (59.96%)
ThiNet-50 [15]	68.42	88.30	1.10B (73.11%)	8.66M (66.04%)
HRank-C [35]	69.10	89.58	0.98B (76.04%)	8.27M (67.59%)
Ours-0.35	**70.58**	**90.00**	**0.99B (75.80%)**	**8.02M (68.55%)**

4.4 Results on Object Detection Task

The proposed method is also applicable to other major computer vision tasks such as object detection. We take the SSD [37] (PyTorch version) as an example, its backbone network is similar to VGG16 and we prune it with VOC0712 dataset. Table 3 shows that our method is effective for object detection task, for example, it removes 30.92% FLOPs and 34,80% Parameters from baseline with improvement of 0.1% mAP. In fact, it is widely applicable to any network with common CNN structures.

Table 3. Pruning results of SSD on VOC0712.

Method	mAP (%)	FLOPs (PR)	Parameters (PR)
Baseline	77.68	31.40B (0.00%)	26.29M (0.00%)
Ours-0.05	77.78	21.69B (30.92%)	17.14M (34.80%)
Ours-0.1	77.14	18.27B (41.82%)	14.17M (46.10%)
Ours-0.3	76.45	12.22B (61.08%)	9.29M (64.66%)
Ours-0.5	75.83	9.69B (30.92%)	6.87M (16.67%)



	conv1	res1-block 1	res1-block 2	res1-block 3	res2-block 1	res2-block 2	res2-block 3	res2-block 4	res3-block 1	res3-block 2	res3-block 3	res3-block 4	res3-block 5	res3-block 6	res4-block 1	res4-block 2	res4-block 3
🟧 Pruned Filters	13	97	144	119	151	236	231	199	262	406	510	558	560	488	777	790	778
🟦 Retained Filters	51	287	240	265	617	532	537	569	1274	1130	1026	978	976	1048	2295	2282	2294

Fig. 3. Statistics of pruned (orange) and retained (blue) filters of ResNet50 on ImageNet with Ours-0.2 method. Shortcuts with convolutional layers are not included. (Color figure online)

4.5 Filter Pruning Analysis

As shown in Fig. 3, we reveal the pruning details about Ours-0.2 result on ImageNet dataset. We count the number of pruned/retained filters in all the layers except the shortcut connections with convolutional filters of ResNet50. For simplicity, the filter numbers of layers in the same block have been added together, the full result of each layer can be found in our supplementary material.

In ResNet50, there are 16 residual-blocks and one convolutional layer. Figure 3 shows that the pruned filters mainly distribute in the high-level blocks. The filters of the high-level blocks contain more semantic information in detail, some of which are redundant. Empirically, removing these filters has less impact on the performance of the network. Meanwhile, the pruning rate of each block is different as illustrated in Fig. 3, which also proves that our method attempts to search for the optimal pruning rate in each layer.

5 Conclusions

In this paper, we propose a method to search the optimal pruning rate in a layer-wise manner and only needs fine-tuning for once. Based on the filter importance criterion derived from loss variation and first-order approximation, convolutional networks can be pruned efficiently. The group pruning and channel selection mechanism are also introduced to adapt with shortcut connections in networks. For practical usage, binary search the threshold accelerates pruning at a given global pruning rate for the entire network without extra fine-tuning. Experiments demonstrate that our method outperforms previous state-of-the-art pruning methods on different datasets and networks. The code is available at https://github.com/Nuctech-AI/LBS_pruning.

References

1. Howard, A.G., et al.: Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
2. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: an extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6848–6856 (2018)
3. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in Neural Information Processing Systems, pp. 598–605 (1990)
4. Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: optimal brain surgeon. In: Advances in Neural Information Processing Systems, pp. 164–171 (1993)
5. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710 (2016)
6. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: training deep neural networks with binary weights during propagations. In: Advances in Neural Information Processing Systems, pp. 3123–3131 (2015)
7. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. The Journal of Machine Learning Research **18**, 6869–6898 (2017)
8. Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in neural information processing systems. (2014) 1269–1277
9. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
10. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, pp. 1135–1143 (2015)
11. Srinivas, S., Babu, R.V.: Data-free parameter pruning for deep neural networks. arXiv preprint arXiv:1507.06149 (2015)
12. Han, S., et al.: EIE: efficient inference engine on compressed deep neural network. ACM SIGARCH Computer Architecture News **44**, 243–254 (2016)
13. Park, J., et al.: Faster CNNs with direct sparse convolutions and guided pruning. arXiv preprint arXiv:1608.01409 (2016)

14. Hu, H., Peng, R., Tai, Y., Tang, C., Trimming, N.: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250 (2016)
15. Luo, J.H., Wu, J., Lin, W.: Thinet: a filter level pruning method for deep neural network compression. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 5058–5066 (2017)
16. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1389–1397 (2017)
17. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4340–4349 (2019)
18. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 11264–11272 (2019)
19. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: AMC: automl for model compression and acceleration on mobile devices. In: Proceedings of the European Conference on Computer Vision (ECCV). (2018) 784–800
20. Huang, Z., Wang, N.: Data-driven sparse structure selection for deep neural networks. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11220, pp. 317–334. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01270-0_19
21. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
22. Russakovsky, O., et al.: Imagenet large scale visual recognition challenge. Int. J. Comput. Vision **115**, 211–252 (2015)
23. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
24. Szegedy, C., et al.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
25. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
26. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4700–4708 (2017)
27. Yu, R., et al.: NISP: pruning networks using neuron importance score propagation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9194–9203 (2018)
28. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2736–2744 (2017)
29. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440 (2016)
30. You, Z., Yan, K., Ye, J., Ma, M., Wang, P.: Gate decorator: global filter pruning method for accelerating deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 2133–2144 (2019)
31. Lin, S., et al.: Towards optimal structured CNN pruning via generative adversarial learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2790–2799 (2019)

32. Lin, S., Ji, R., Li, Y., Wu, Y., Huang, F., Zhang, B.: Accelerating convolutional networks via global & dynamic filter pruning. IJCA **I**, 2425–2432 (2018)
33. Ding, X., Ding, G., Guo, Y., Han, J.: Centripetal SGD for pruning very deep convolutional networks with complicated structure. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4943–4953 (2019)
34. Paszke, A., et al.: Automatic differentiation in pytorch (2017)
35. Lin, M., et al.: Hrank: filter pruning using high-rank feature map. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1529–1538 (2020)
36. Zhao, C., et al.: Variational convolutional neural network pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2780–2789 (2019)
37. Liu, W., et al.: SSD: single shot MultiBox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2