# Learning Filter Pruning Criteria for Deep Convolutional Neural Networks Acceleration

Yang He[1] Yuhang Ding[2] Ping Liu[1] Linchao Zhu[1] Hanwang Zhang[3] Yi Yang[1]*
[1]ReLER, University of Technology Sydney [2]Baidu Research [3]Nanyang Technological University
yang.he-1@student.uts.edu.au, {dyh.ustc.uts,pino.pingliu,zhulinchao7}@gmail.com
hanwangzhang@ntu.edu.sg, yee.i.yang@gmail.com

## Abstract

*Filter pruning has been widely applied to neural network compression and acceleration. Existing methods usually utilize pre-defined pruning criteria, such as $\ell_p$-norm, to prune unimportant filters. There are two major limitations to these methods. First, prevailing methods fail to consider the variety of filter distribution across layers. To extract features of the coarse level to the fine level, the filters of different layers have various distributions. Therefore, it is not suitable to utilize the same pruning criteria to different functional layers. Second, prevailing layer-by-layer pruning methods process each layer independently and sequentially, failing to consider that all the layers in the network collaboratively make the final prediction.*

*In this paper, we propose Learning Filter Pruning Criteria (LFPC) to solve the above problems. Specifically, we develop a differentiable pruning criteria sampler. This sampler is learnable and optimized by the validation loss of the pruned network obtained from the sampled criteria. In this way, we could adaptively select the appropriate pruning criteria for different functional layers. Besides, when evaluating the sampled criteria, LFPC comprehensively considers the contribution of all the layers at the same time. Experiments validate our approach on three image classification benchmarks. Notably, on ILSVRC-2012, our LFPC reduces more than 60% FLOPs on ResNet-50 with only 0.83% top-5 accuracy loss.*

## 1. Introduction

Convolutional neural networks have achieved significant advancement in various computer vision research applications [43, 15, 47]. However, most of these manually designed architectures, *e.g.*, VGG [43], ResNet [15], usually come with the enormous model size and heavy computation cost. It is hard to deploy these models in scenarios demanding a real-time response. Recently, studies on model compression and acceleration are emerging. Due to its efficacy, the pruning strategy attracts attention in previous studies [14, 27, 20].

*Corresponding Author.

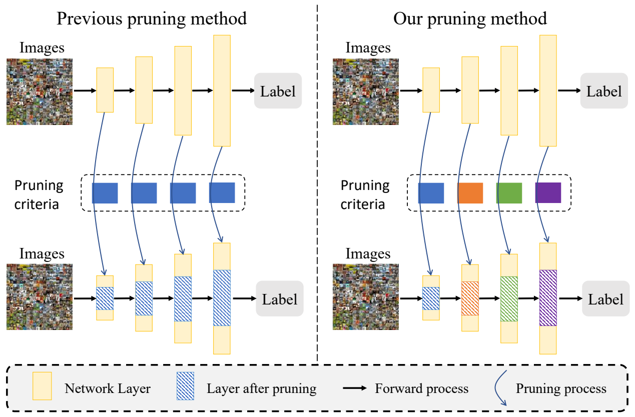

Figure 1. (a) Previous filter pruning methods manually select a criterion and apply it to all layers; (b) our pruning method learns appropriate criteria for different layers based on the filter distribution. In the blue dashed box, the solid boxes of different colors denote different pruning criteria. The yellow boxes without shadow correspond to unpruned layers of the network, while the ones with shadow are the layers pruned by a selected pruning criterion.

Recent developments on pruning can be divided into two categories, *i.e.*, weight pruning [14], and filter pruning [27]. Filter pruning is preferred compared to weight pruning because filter pruning could make the pruned model more structural and achieve practical acceleration [20]. The existing filter pruning methods follow a three-stage pipeline. (1) *Training*: training a large model on the target dataset. (2) *Pruning*: based on a particular criterion, unimportant filters from the pre-trained model are pruned. (3) *Fine-tuning (retraining)*: the pruned model is retrained to recover the original performance. During the three stages, select an appropriate pruning criterion is the key ingredient.

However, the previous works have a few drawbacks and might not be the best choice in real scenarios. First, previous works manually specify a pruning criterion and utilize the *same* pruning criterion for *different* layers. As shown in [52], different layers have different filter distributions and various functions. The lower layers tend to extract coarse level features, such as lines, dots, and curves, while the higher layers tend to extract fine level features, such as common objects and shapes. In this situation, fixing one pruning criterion for all the functional layers may not be suitable. Second, prevailing methods prune the network in a greedy layer-by-layer manner, *i.e.*, the pruning process at different layers is *independent* of each other. Considering that during training and inference, the filters of all the layers work *collaboratively* to make a final prediction, it is natural to suggest to conduct pruning in a collaborative, not an independent, manner. In other words, it is preferred that the filter importance of all layers could be evaluated concurrently.

We propose Learning Filter Pruning Criteria (LFPC) to solve the mentioned problems. The core component of LFPC is a Differentiable Criteria Sampler (DCS), which aims to sample different criteria for different layers. This sampler, since it is differentiable, can be updated efficiently to find the appropriate criteria. First, DCS initializes a learnable criteria probability for all layers. For every layer, DCS conducts *criteria forward* to get the *criteria feature map* based on the filters and criteria probability. The process of *criteria forward* is shown in Sec. 3.2.3. After *criteria forward* for all the layers, we get the *criteria loss* and utilize it as a supervision signal. The *criteria loss* can be back-propagated to update the criteria probability distribution to fit the filter distribution of the network better. Different from previous layer-by-layer pruning works, our LFPC can consider all the layers and all the pruning criteria simultaneously through the *criteria loss*. After finishing training the DCS, the optimized criteria servers as the pruning criteria for the network, as shown in Fig. 1. After pruning, we fine-tune the pruned model once to get an efficient and accurate model.

**Contributions.** Contributions are summarized as follows:

(1) We propose an effective learning framework, Learning Filter Pruning Criteria (LFPC). This framework can learn to select the most appropriate pruning criteria for each functional layer. Besides, the proposed Differentiable Criteria Sampler (DCS) can be trained end-to-end and consider all the layers concurrently during pruning. To the best of our knowledge, this is the first work in this research direction.

(2) The experiment on three benchmarks demonstrates the effectiveness of our LFPC. Notably, it accelerates ResNet-110 by two times, with even 0.31% relative accuracy improvement on CIFAR-10. Additionally, we reduce more than 60% FLOPs on ResNet-50 with only 0.83% top-5 accuracy loss.

## 2. Related Work

Previous work on pruning can be categorized into weight pruning and filter pruning. Weight pruning [14, 13, 12, 48, 2, 56, 6] focuses on pruning fine-grained weight of filters, so that leading to unstructured sparsity in models. In contrast, filter pruning [27] could achieve the structured sparsity, so the pruned model could take full advantage of high-efficiency Basic Linear Algebra Subprograms (BLAS) libraries to achieve better acceleration.

Considering how to evaluate the filter importance, we can roughly divide the filter pruning methods into two categories, *i.e.*, weight-based criteria, and activation-based criteria. Furthermore, the pruning algorithms could also be roughly grouped by the frequency of pruning, *i.e.*, greedy pruning, and one-shot pruning. We illustrate the categorization in Tab. 1.

| Algorithms | Criteria<br>W \| A | Frequency<br>O \| G |
|---|---|---|
| PFEC [27], SFP [18], FPGM [20] | W | O |
| RSA [51], PRE [39] | W | G |
| SLIM [32], PFA [45], NISP [54], CCP [41], GAL [30] | A | O |
| CP [22], SLIM [32], ThiNet [36], PRE [39], DCP [57], LPF [23], AOFP [5], GATE [53] | A | G |

Table 1. Different categories of filter pruning algorithms. "W" and "A" denote the weight-based and activation-based criteria. "O" and "G" indicate the one-shot and greedy pruning.

**Weight-based Criteria.** Some methods [27, 18, 51, 20, 17, 21, 50] utilize the weights of the filters to determine the importance of the filters. [27] prunes the filters with small $\ell_1$-norm. [18] utilizes $\ell_2$-norm criterion to select filters and prune those selected filters softly. [51] introduces sparsity on the scaling parameters of batch normalization (BN) layers to prune the network. [20] claims that the filters near the geometric median should be pruned. All the works utilize the same pruning criteria for different layers and do not take into account that different layers have various functions and different filter distributions.

**Activation-based Criteria.** Some works [32, 36, 22, 39, 10, 45, 54, 57, 23, 19, 30, 29, 38, 24, 41] utilize the training data and filter activations to determine the pruned filters. [45] adopts the Principal Component Analysis (PCA) method to specify which part of the network should be preserved. [36] proposes to use the information from the next layer to guide the filter selection. [10] minimizes the reconstruction error of training set sample activations and applies Singular Value Decomposition (SVD) to obtain a decomposition of filters. [49] explores the linear relationship in different feature maps to eliminate the redundancy in convolutional filters.

**Greedy and One-shot Pruning.** Greedy pruning [53, 5], or oracle pruning, means the pruning and retraining should be operated for multiple times. Although greedy pruning is beneficial for accuracy, it is time-consuming and requires a large number of computation resources. In contrast, one-shot pruning [20, 27] prunes the network once and retrained once to recover the accuracy. It is more efficient than the greedy pruning, but it requires careful pruning criteria selection. We focus on one-shot pruning in this paper.

**Other Pruning and Searching Methods.** Some works utilize reinforcement learning [19, 23] or meta-learning [33] for pruning. In contrast, we focus on learning the proper pruning criteria for different layers via the differential sampler. [4] proposes centripetal SGD to make several filters to converge into a single point. [54] is a global pruning method, but the importance of pruned neurons is not propagated. The idea of our learning criteria shares some similarities with the Neural Architecture Search (NAS) works [58, 31] and Autoaugment [3], the difference is that our search space is the pruning criteria instead of network architectures or augmentation policies.

## 3. Methodology

### 3.1. Preliminaries

We assume that a neural network has $L$ layers, and we represent the weight for $l_{th}$ convolutional layers as $\mathbf{W}^{(l)} \in \mathbb{R}^{K\times K\times C_I^{(l)}\times C_O^{(l)}}$, where $K$ is the kernel size , $C_I^{(l)}$ and $C_O^{(l)}$ is the number of input and output channels, respectively. In this way, $\mathbf{W}_i^{(l)} \in \mathbb{R}^{K\times K\times C_I^{(l)}}$ represents the $i_{th}$ filter of $l_{th}$ convolutional layer. We denote the $\mathbf{I}$ and $\mathbf{O}$ as the input and output feature maps, and $\mathbf{I} \in C_I^{(l)} \times H_I^{(l)} \times W_I^{(l)}$ and $\mathbf{O} \in C_O^{(l)} \times H_O^{(l)} \times W_O^{(l)}$, where $H_*^{()}$ and $W_*^{()}$ is the height and width of the feature map, respectively. The convolutional operation of the $i_{th}$ layer can be written as:

$$\mathbf{O}_i = \mathbf{W}_i^{(l)} * \mathbf{I} \ \textbf{ for } 1 \le i \le C_O^{(l)}, \tag{1}$$

Assume the filter set $\mathcal{F}$ consists all the filters in the network: $\mathcal{F} = \left\{\mathbf{W}_i^{(l)},\ \ i \in [1, C_O^{(l)}],\ \ l \in [1, L]\right\}$. We divide $\mathcal{F}$ into two disjoint subsets: the kept filter set $\mathcal{K}$ and removed filter set $\mathcal{R}$, and we have:

$$\mathcal{K} \cup \mathcal{R} = \mathcal{F}, \quad \mathcal{K} \cap \mathcal{R} = \emptyset. \tag{2}$$

Now our target becomes clear. Filter pruning aims to minimize the loss function value under sparsity constraints on filters. Given a dataset $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ where $\mathbf{x}_n$ denotes the $n_{th}$ input and $\mathbf{y}_n$ is the corresponding output, the constrained optimization problem can be formulated as:

$$\begin{aligned} \min_{\mathcal{K}} \mathcal{L}(\mathcal{K};\mathcal{D}) &= \min_{\mathcal{K}} \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}(\mathcal{K}; (\mathbf{x}_n, \mathbf{y}_n)) \\ \text{s.t. } \frac{\mathcal{C}(\mathcal{K})}{\mathcal{C}(\mathcal{F})} &\le r \end{aligned} \tag{3}$$

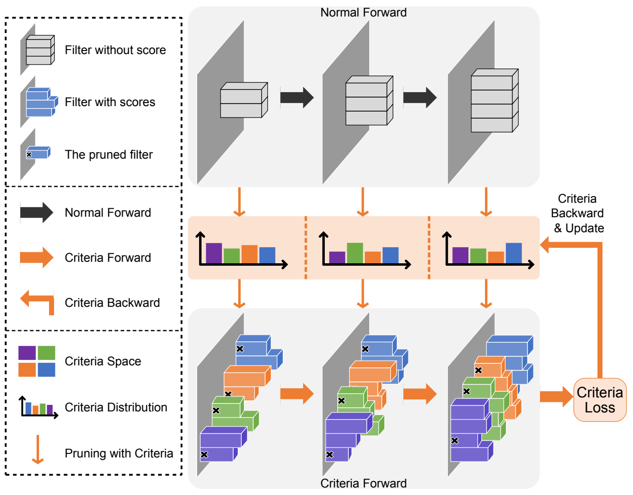

Figure 2. Criteria forward and backward in the network. Grey boxes are the normal filters. The probability distribution of criteria for three layers are initialized, as shown in the big orange shadow. After pruning with four criteria, we obtain four "pruned versions" for every layer, which are denoted as boxes in purple, green, orange, and blue color. These filters are utilized to conduct *criteria forward*. Then we get the *criteria loss* on the validation set to update the "criteria distribution".

where $\mathcal{L}(\cdot)$ is a standard loss function (*e.g.*, cross-entropy loss), $\mathcal{C}(\cdot)$ is the computation cost of the network built from the filter set, and $r$ is the ratio of the computation cost of between pruned network and the original unpruned network.

### 3.2. Learning Filter Pruning Criteria

In this section, we illustrate our proposed LFPC, which can automatically and adaptively choose an appropriate criterion for each layer based on their respective filter distribution. The overall learning process is shown in Fig. 2.

#### 3.2.1 Pruning Criteria

For simplicity, we introduce the pruning criteria based on $l_{th}$ layer. The filters in $l_{th}$ layer are denoted as a filter set $\mathcal{F}^{(l)} = \left\{\mathbf{W}_i^{(l)},\ \ i \in [1, C_O^{(l)}]\right\}$. In $l_{th}$ layer, a pruning criterion, denoted as $\text{Crit}^{(l)}(\cdot)$, is utilized to get the importance scores for the filters. Then we have

$$score^{(l)} = \text{Crit}^{(l)}(\mathcal{F}^{(l)}) \tag{4}$$

where $score^{(l)} \in \mathbb{R}^{C_O^{(l)}}$ is the importance score vector of the filters in $l_{th}$ layer. For example, $\ell_1$-norm criteria [27] could be formulated as $\text{Crit}^{(l)}(\mathcal{F}^{(l)}) = \left\{\text{Crit}^{(l)}(\mathbf{W}_i^{(l)}) = \|\mathbf{W}_i^{(l)}\|_1 \ \text{ for } \ i \in [1, C_O^{(l)}]\right\}$.

Then filter pruning is conducted based on $score^{(l)}$:

$$\begin{aligned} keepid^{(l)} &= \text{Topk}(score^{(l)}, \text{n}^{(l)}) \\ \mathcal{K}^{(l)} &= \text{Prune}(\mathcal{F}^{(l)}, keepid^{(l)}), \end{aligned} \tag{5}$$

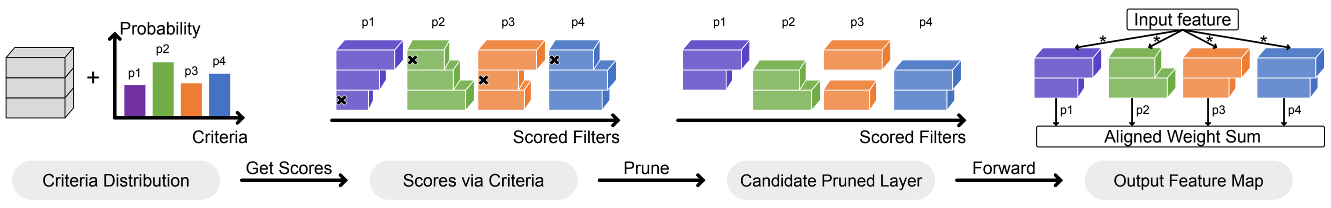

Figure 3. Criteria forward within a layer. Boxes of different colors indicate the different pruning criteria. First, we evaluate the importance of the filter based on different criteria. Second, we prune the filter with small importance scores and get four versions of pruned layers with various probabilities. After that, the output feature map is the aligned weighted sum of four feature maps of the pruned layers.

where $\mathrm{n}^{(l)}$ is the number of filters to be kept, and $\mathrm{Topk}(\cdot)$ returns the indexes of the $k$ most important filter based on their importance scores. The indexes are denoted as $keepid^{(l)}$. Given $keepid^{(l)}$, $\mathrm{Prune}(\cdot)$ keeps the critical filters with the indexes specified in $keepid^{(l)}$, and prunes the other filters. The filter set after pruning is denoted as $\mathcal{K}^{(l)}$.

#### 3.2.2 Criteria Space Complexity

If we want to keep $\mathrm{n}^{(l)}$ filters in $l_{th}$ layer, which has totally $C_O^{(l)}$ filters, then the number of selection could be $\binom{C_O^{(l)}}{\mathrm{n}^{(l)}} = \frac{C_O^{(l)}!}{\mathrm{n}^{(l)}!\ (C_O^{(l)}-\mathrm{n}^{(l)})!}$, where $\binom{}{}$ denotes the combination [1]. For those frequently used CNN architectures, the number of selections might be surprisingly big. For example, pruning 10 filters from a 64-filter-layer has $\binom{64}{10} = 151,473,214,816$ selections. This number would increase dramatically if the more layers are considered. Therefore, it is impossible to learn the pruning criteria from scratch. Fortunately, with the help of the proposed criteria of previous works [27, 20], we could reduce the criteria space complexity from $\binom{C_O^{(l)}}{\mathrm{n}^{(l)}}$ to $S$, which is the number of criteria that we adopted.

#### 3.2.3 Differentiable Criteria Sampler

Assuming there are $S$ candidate criteria in the criteria space, we could use $\alpha^{(l)} \in \mathbb{R}^S$ to indicate the distribution of the possible criteria for $l_{th}$ layer. The probability of choosing the $i_{th}$ criterion can be formulated as:

$$p_i = \frac{\exp\left(\alpha_i^{(l)}\right)}{\sum_{j=1}^{S} \exp\left(\alpha_j^{(l)}\right)} \quad \text{where} \quad 1 \le i \le S \tag{6}$$

However, since Eq. 6 needs to sample from a discrete probability distribution, we cannot back-propagate gradients through $p_i$ to $\alpha_i^{(l)}$. To allow back-propagation, inspired from [9], we apply Gumbel-Softmax [25, 37] to reformulate Eq. 6 as Eq. 7:

**Algorithm 1:** Algorithm Description of LFPC

```
Input  : training data $\mathbf{X}$, validation data $\mathbf{Y}$ ;
    the pre-trained model with parameters
    $\mathbf{W} = \{\mathbf{W}^{(l)}, 1 \le l \le L\}$;
    $S$ candidate criteria and criteria parameters $\alpha$ ;
    expected ratio of computational cost $r$ ;
Output: The compact model and its parameters $\mathbf{W}^*$
1 for $epoch \leftarrow 1$ to $epoch_{max}$ do
2     for $l \leftarrow 1$ to $L$ do
3         Sample criteria based on Eq. 7 ;
4         Calculate criteria feature with Eq. 9 and $r$;
5     end
6     Update $\alpha$ based on $\mathbf{Y}$ using Eq. 11 ;
7 end
8 Get final criteria set $\mathcal{T}$ and conduct pruning ;
9 Re-training pruned model with $\mathbf{X}$ and obtain $\mathbf{W}^*$.
```

$$\hat{p}_i = \frac{\exp\left(\left(\log\left(p_i\right) + \boldsymbol{o}_i\right)/\tau\right)}{\sum_{j=1}^{S} \exp\left(\left(\log\left(p_j\right) + \boldsymbol{o}_j\right)/\tau\right)} \tag{7}$$
$$\text{s.t. } \boldsymbol{o}_i = -\log(-\log(u)) \ \ \& \ \ u \sim \mathcal{U}(0,1)$$

where $\mathcal{U}(0,1)$ is the uniform distribution between 0 and 1, $u$ is a sample from the distributio $\mathcal{U}(0,1)$, and $\tau$ is the softmax temperature. We denote $\hat{p} = [\hat{p}_1, \ldots, \hat{p}_j, \ldots]$ as the Gumbel-softmax distribution. Change the parameter $\tau$ would lead to different $\hat{p}$. When $\tau \to \infty$, $\hat{p}$ becomes a uniform distribution. When $\tau \to 0$, samples from $\hat{p}$ become one-shot, and they are identical to the samples from the categorical distribution [25].

**Criteria Forward.** The illustration of criteria forward for $l_{th}$ layer is shown in Fig 3. For simplicity, we rewrite the Eq. 4 and Eq. 5 as $\mathcal{K}^{(l)} = g(\mathcal{F}^{(l)}, \mathrm{Crit}^{(l)}, \mathrm{n}^{(l)})$. For $l_{th}$ layer, it has $S$ sampled "pruned version" which can be formulated as:

$$\mathcal{K}_s^{(l)} = g(\mathcal{F}^{(l)}, \mathrm{Crit}_s^{(l)}, \mathrm{n}^{(l)}) \ \text{ for } \ s \in [1, S] \tag{8}$$

where $\mathrm{Crit}_s^{(l)}$ denotes the process of utilizing $s_{th}$ pruning criterion to get the importance scores for the filters in $l_{th}$ layer, and $\mathcal{K}_s^{(l)}$ is the kept filter set under $s_{th}$ criterion. To comprehensively consider the contribution of every criterion during training, the output feature map is defined as the Aligned Weighted Sum (AWS) of the feature maps from different $\mathcal{K}_s^{(l)}$, which can be formulated as:

$$\begin{aligned} \mathbf{O}^{AWS} &= \sum_{s=1}^{S} \text{Align}(\hat{p}_s \times \hat{\mathbf{O}}_s), \\ \text{Align}(\hat{\mathbf{O}}_{s,i}) &= \hat{\mathbf{O}}'_{s,keepid_s^{(l)}[i]} \qquad i \in [1, \mathrm{n}^{(l)}]. \end{aligned} \tag{9}$$

where $\mathbf{O}^{AWS}$ is the *criteria feature map* of the layer, $\hat{p}_s$ is the probability for $s_{th}$ criteria, $\times$ denotes the scalar multiplication, $\hat{\mathbf{O}}_s$ is the output feature map of $\mathcal{K}_s^{(l)}$, and $\hat{\mathbf{O}}'_s$ is the aligned feature. For the second formulation, $keepid_s^{(l)}[i]$ is the $i_{th}$ element of the $keepid$ (Eq. 5) under $s_{th}$ criteria in $l_{th}$ layer. To explain the $\text{Align}(\cdot)$ function, we take the third figure of Fig. 3 for example. The first channel of *purple* network could only be added with the first channel of *orange* network, not the green and blue one. This operation can avoid the interference of the information from different channels. Further we have:

$$\begin{aligned} \hat{\mathbf{O}}_s &= \left[\hat{\mathbf{O}}_{s,1}, \hat{\mathbf{O}}_{s,2}, ..., \hat{\mathbf{O}}_{s,\mathrm{n}^{(l)}}\right], \\ \hat{\mathbf{O}}_{s,i} &= \mathcal{K}_{s,i}^{(l)} * \mathbf{I}, \qquad s \in [1, S], \;\; i \in [1, \mathrm{n}^{(l)}], \end{aligned} \tag{10}$$

where $\hat{\mathbf{O}}_{s,i}$ is the $i_{th}$ output feature of $\mathcal{K}_s^{(l)}$, and $*$ is the convolution operation. After *criteria forward* for all the layers, we could get the *criteria loss*, as shown in Fig. 2.

**Training Objectives.** For a $L$-layer network, the criteria parameter $\alpha = \left\{\alpha^{(1)}, \alpha^{(2)}, ..., \alpha^{(L)}\right\}$. We aim to find a proper $\alpha$ to give us guidance about which criterion is suitable for different layers. Specifically, $\alpha$ is found by minimizing the validation loss $\mathcal{L}_{val}$ after trained the *criteria network* $\theta_\alpha$ by minimizing the training loss $\mathcal{L}_{train}$:

$$\begin{aligned} &\min_{\alpha} \mathcal{L}_{val}\left(\theta_\alpha^*, \alpha\right) \\ \text{s.t.} \quad &\theta_\alpha^* = \arg\min_{\theta_\alpha} \mathcal{L}_{train}(\theta_\alpha, \alpha), \end{aligned} \tag{11}$$

where $\theta_\alpha^*$ is the optimized *criteria network* under the optimized criteria set $\alpha$. The training loss is the cross-entropy classification loss of the networks. To further consider the computation cost of the pruned network, the penalty for the computation cost is also included in the validation loss:

$$\mathcal{L}_{val} = \mathcal{L}_{crit} + \lambda_{comp}\mathcal{L}_{comp}, \tag{12}$$

where $\mathcal{L}_{crit}$ is the standard classification loss of the criteria network, namely the *criteria loss*, and $\mathcal{L}_{comp}$ is the computation loss of the pruned network. $\lambda_{comp}$ is a balance of these two losses, whose details can be found in the supplementary material. In this way, we could get the optimized criteria parameters $\alpha$ for the network under different computation constraints.

**Criteria Backward.** We backward $\mathcal{L}_{val}$ in Eq. 12 to $\alpha$ to update these parameters collaboratively at the same time. The illustration of this process is shown in Fig. 2.

**After DCS Training.** By choosing the criterion with the maximum probability, we get the final criteria set $\mathcal{T}$ for all the layers. Then we conduct a conventional pruning operation based on the optimized criteria $\mathcal{T}$ to get the pruned network. The pruned network is then retrained to get the final accurate pruned model. The whole process is shown in the Alg. 1.

## 4. Experiments

### 4.1. Experimental Setting

**Datasets.** In this section, we validate the effectiveness of our acceleration method on three benchmark datasets, CIFAR-10, CIFAR-100 [26], and ILSVRC-2012 [42]. The CIFAR-10 dataset contains $50,000$ training images and $10,000$ testing images, in total $60,000$ $32 \times 32$ color images in 10 different classes. CIFAR-100 has 100 classes, and the number of images is the same as CIFAR-10. ILSVRC-2012 [42] contains 1.28 million training images and 50k validation images of $1,000$ classes.

**Architecture Setting.** As ResNet has the shortcut structure, existing works [7, 36, 22] claim that ResNet has less redundancy than VGGNet [43] and accelerating ResNet is more difficult than accelerating VGGNet. Therefore, we follow [8] to focus on pruning the challenging ResNet.

**Normal Training Setting.** For ResNet on CIFAR-10 and CIFAR-100, we utilize the same training schedule as [55]. In the CIFAR experiments, we run each setting three times and report the "mean $\pm$ std". In the ILSVRC-2012 experiments, we use the default parameter settings, which are the same as [15, 16], and the same data argumentation strategies as the official PyTorch [40] examples.

**DCS training Setting.** The weight-based criteria are selected as our candidate criteria for their efficiency. Specifically, $\ell_1$-norm [27], $\ell_2$-norm [18] and geometric median based [20] criteria. The criteria could be formulated as $\text{Crit}^{(l)}(\mathbf{W}_i^{(l)}) = \|\mathbf{W}_i^{(l)}\|_p$ and $\text{Crit}^{(l)}(\mathbf{W}_i^{(l)}) = \sqrt[2]{\sum_{j=1}^{C_O^{(l)}} \left|\mathbf{W}_i^{(l)} - \mathbf{W}_j^{(l)}\right|^2}$ for $i \in [1, C_O^{(l)}]$. Note that our framework is able to extend to more criteria.

We set desired FLOPs according to compared pruning algorithms and set $\lambda_{comp}$ of Eq. 12 as 2. We randomly split half of the training set as the validation set for Eq. 11. We optimize the criteria parameters via Adam, and we use the constant learning rate of 0.001 and a weight decay of 0.001. On CIFAR, we train the DCS for 600 epochs with a batch size of 256. On ILSVRC-2012, we train the DCS for 35 epochs with a batch size of 256. The $\tau$ in Eq. 7 is linearly decayed from 5 to 0.1. During training DCS, we fix the pre-trained weights [8] to reduce overfitting.

**Pruning Setting.** After training DCS, we prune the network with the optimized criteria and fine-tune the network with the full training set. We analyze the difference be-

| Depth | Method | Init pretrain | Baseline acc. (%) | Pruned acc. (%) | Acc. ↓ (%) | FLOPs | FLOPs ↓(%) |
|---|---|---|---|---|---|---|---|
| 32 | MIL [7] | ✗ | 92.33 | 90.74 | 1.59 | 4.70E7 | 31.2 |
| | SFP [18] | ✗ | **92.63** (±0.70) | 92.08 (±0.08) | 0.55 | 4.03E7 | 41.5 |
| | FPGM [20] | ✗ | **92.63** (±0.70) | **92.31** (±0.30) | **0.32** | 4.03E7 | 41.5 |
| | Ours | ✗ | **92.63** (±0.70) | 92.12 (±0.32) | 0.51 | **3.27E7** | **52.6** |
| 56 | PFEC [27] | ✗ | 93.04 | 91.31 | 1.75 | 9.09E7 | 27.6 |
| | Ours | ✗ | **93.59** (±0.58) | **93.56** (±0.29) | **0.03** | 6.64E7 | 47.1 |
| | CP [22] | ✗ | 92.80 | 90.90 | 1.90 | – | 50.0 |
| | SFP [18] | ✗ | **93.59** (±0.58) | 92.26 (±0.31) | 1.33 | 5.94E7 | 52.6 |
| | FPGM [20] | ✗ | **93.59** (±0.58) | 92.89 (±0.32) | 0.70 | 5.94E7 | 52.6 |
| | Ours | ✗ | **93.59** (±0.58) | 93.34 (±0.08) | 0.25 | **5.91E7** | **52.9** |
| | PFEC [27] | ✓ | 93.04 | 93.06 | -0.02 | 9.09E7 | 27.6 |
| | NISP [54] | ✓ | – | – | 0.03 | – | 42.6 |
| | Ours | ✓ | **93.59** (±0.58) | **93.72** (±0.29) | **-0.13** | 6.64E7 | 47.1 |
| | CP [22] | ✓ | 92.80 | 91.80 | 1.00 | – | 50.0 |
| | AMC [19] | ✓ | 92.80 | 91.90 | 0.90 | – | 50.0 |
| | FPGM [20] | ✓ | **93.59** (±0.58) | 93.26 (±0.03) | 0.33 | 5.94E7 | 52.6 |
| | Ours | ✓ | **93.59** (±0.58) | 93.24 (±0.17) | **0.35** | **5.91E7** | **52.9** |
| 110 | PFEC [27] | ✗ | 93.53 | 92.94 | 0.61 | 1.55E8 | 38.6 |
| | MIL [7] | ✗ | 93.63 | 93.44 | 0.19 | - | 34.2 |
| | SFP [18] | ✗ | 93.68 (±0.32) | 93.38 (±0.30) | 0.30 | 1.50E8 | 40.8 |
| | Rethink [34] | ✗ | **93.77** (±0.23) | 93.70 (±0.16) | 0.07 | 1.50E8 | 40.8 |
| | FPGM [20] | ✗ | 93.68 (±0.32) | 93.73 (±0.23) | -0.05 | 1.21E8 | 52.3 |
| | Ours | ✗ | 93.68 (±0.32) | **93.79** (±0.38) | **-0.11** | **1.01E8** | **60.3** |
| | PFEC [27] | ✓ | 93.53 | 93.30 | 0.20 | 1.55E8 | 38.6 |
| | NISP [54] | ✓ | – | – | 0.18 | – | 43.8 |
| | GAL [30] | ✓ | 93.26 | 92.74 | 0.81 | – | 48.5 |
| | FPGM [20] | ✓ | **93.68** (±0.32) | **93.74** (±0.10) | **-0.16** | 1.21E8 | 52.3 |
| | Ours | ✓ | **93.68** (±0.32) | 93.07 (±0.15) | 0.61 | **1.01E8** | **60.3** |

Table 2. Comparison of the pruned ResNet on CIFAR-10. In "Init pretrain" column, "✓" and "✗" indicate whether to use the pre-trained model as initialization or not, respectively. The "Acc. ↓" is the accuracy drop between pruned model and the baseline model, the smaller, the better. A negative value in "Acc. ↓" indicates an improved model accuracy.

tween pruning a scratch model and the pre-trained model. For pruning the scratch model, we utilize the regular training schedule without additional fine-tuning. For pruning the pre-trained model, we reduce the learning rate to one-tenth of the original learning rate. To conduct a fair comparison, we use the same baseline model as [20] for pruning. During retraining, we use the cosine scheduler [35, 8] for a stable result. The pruning rate of every layer is sampled in the same way as DCS[1], so we could search the ratio automatically and adaptively [8].

We compare our method with existing state-of-the-art acceleration algorithms, *e.g.*, MIL [7], PFEC [27], CP [22], ThiNet [36], SFP [18], NISP [54], FPGM [20], LFC [44], ELR [49], GAL [30], IMP [38], DDS [24]. Experiments show that our LFPC achieves a comparable performance with those works. Our experiments are based on the PyTorch [40] framework. No significant performance difference has been observed with the PaddlePaddle framework.

[1]See supplementary material for details.

## 4.2. ResNet on CIFAR-10

For the CIFAR-10 dataset, we test our LFPC on ResNet with depth 32, 56, and 110. As shown in Tab. 2, the experiment results validate the effectiveness of our method. For example, MIL [7] accelerates the random initialized ResNet-32 by 31.2% speedup ratio with 1.59% accuracy drop, but our LFPC achieves 52.6% speedup ratio with only 0.51% accuracy drop. When we achieve similar accuracy with FPGM [20] on ResNet-32, our acceleration ratio is much larger than FPGM [20]. Comparing to SFP [18], when we prune similar FLOPs of the random initialized ResNet-56, our LFPC has 1.07% accuracy improvement over SFP [18]. For pruning the pre-trained ResNet-56, our method achieves a higher acceleration ratio than CP [22] with a 0.65% accuracy increase over CP [22]. Comparing to PFEC [27], our method accelerates the random initialized ResNet-110 by 60.3% speedup ratio with even 0.11% accu-

| Depth | Method | Init Pretrain | Baseline top-1 acc.(%) | Pruned top-1 acc.(%) | Baseline top-5 acc.(%) | Pruned top-5 acc.(%) | Top-1 acc. ↓(%) | Top-5 acc. ↓(%) | FLOPs↓ (%) |
|---|---|---|---|---|---|---|---|---|---|
| 50 | SFP [18] | ✗ | **76.15** | **74.61** | **92.87** | **92.06** | **1.54** | **0.81** | 41.8 |
| | FPGM [20] | ✗ | **76.15** | 74.13 | **92.87** | 91.94 | 2.02 | 0.93 | 53.5 |
| | Ours | ✗ | **76.15** | 74.18 | **92.87** | 91.92 | 1.97 | 0.95 | **60.8** |
| | DDS [24] | ✓ | 76.12 | 74.18 | 92.86 | 91.91 | 1.94 | 0.95 | 31.1 |
| | ThiNet [36] | ✓ | 72.88 | 72.04 | 91.14 | 90.67 | **0.84** | **0.47** | 36.7 |
| | SFP [18] | ✓ | 76.15 | 62.14 | **92.87** | 84.60 | 14.01 | 8.27 | 41.8 |
| | NISP [54] | ✓ | – | – | – | – | 0.89 | – | 44.0 |
| | IMP [38] | ✓ | **76.18** | 74.50 | | | 1.68 | | 45.0 |
| | CP [22] | ✓ | – | – | 92.20 | 90.80 | – | 1.40 | 50.0 |
| | LFC [44] | ✓ | 75.30 | 73.40 | 92.20 | 91.40 | 1.90 | 0.80 | 50.0 |
| | ELR [49] | ✓ | – | – | 92.20 | 91.20 | – | 1.00 | 50.0 |
| | FPGM [20] | ✓ | **76.15** | **74.83** | **92.87** | **92.32** | 1.32 | 0.55 | **53.5** |
| | Ours | ✓ | 76.15 | 74.46 | **92.87** | 92.04 | 1.69 | 0.83 | **60.8** |

Table 3. Comparison of the pruned ResNet on ImageNet. "Init Pretrain" and "acc. ↓" have the same meaning with Table 2.

racy improvement, while PFEC [27] achieves 21.7% less acceleration ratio with 0.61% accuracy drop.

The reason for our superior result is that our proposed method adaptively selects suitable criteria for each functional layer based on their respective filter distribution. On the contrary, none of previous works [18, 22, 27] did this. We notice that pruning from a scratch model sometimes achieves a slightly better performance than pruning a pretrained model, which is consistent with [34]. Note that we achieve a higher acceleration ratio than [34] on ResNet-110 with similar accuracy. We conjecture that the optimized criteria might change the random initialization to "biased" random initialization, which is beneficial to the final performance. This result is consistent with the conclusion of [11] that a proper initialization is critical for the network.

**Criteria Visualization.** The learned pruning criteria for ResNet-56 on CIFAR-10 is shown in Figure 4. The blue, orange and green denote pruning this layer with $\ell_1$-norm, $\ell_2$-norm and geometric median, respectively. The pruned network achieve 93.54($\pm$0.14)% accuracy with pruning 53.0% FLOPs. In this figure, we find that the GM-based criterion is adopted more at higher layers, while the $\ell_p$-norm-based criteria are preferred at lower layers. An explanation is that filters of higher layers tend to extract semantic information, and their activations are semantically related to each other [46]. Therefore, our LFPC chooses the relation-based criteria instead of magnitude-based criteria when pruning higher layers. [2]

## 4.3. ResNet on CIFAR-100

The results of pruning ResNet-56 on CIFAR-100 is shown in Tab. 4. We only list a few methods as other methods have no experiment results on CIFAR-100. When achieving a similar ratio of acceleration, our LFPC could obtain much higher accuracies than the candidate algorithms [18] and [20]. This result again validates the effectiveness of our method.

[2]GM is a relation-based criterion, while $\ell_p$-norm is a magnitude-based criterion. See supplementary material for different filter distribution.

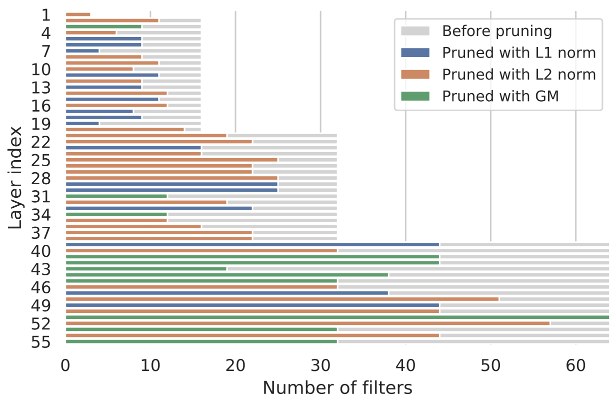

Figure 4. Visualization of the learned criteria and kept filters for ResNet-56 on CIFAR-10. The grey strip indicates the layers before pruning. The blue, orange and green color denote $\ell_1$-norm, $\ell_2$-norm and geometric median criteria, respectively. For example, the bottom green strip means that for all the 64 filters in $55_{th}$ layer, GM criterion is automatically selected to prune half of those filters, base on the filter distribution on that layer.

| Depth | Method | Pruned Acc.(%) | Acc. ↓(%) | FLOPs | FLOPs ↓(%) |
|---|---|---|---|---|---|
| 56 | MIL [7] | 68.37 | 2.96 | 7.63E7 | 39.3% |
| | SFP [18] | 68.79 | 2.61 | 5.94E7 | 52.6% |
| | FPGM [20] | 69.66 | 1.75 | 5.94E7 | 52.6% |
| | Ours | 70.83 | 0.58 | 6.08E7 | 51.6% |

Table 4. Comparison of the pruned ResNet-56 on CIFAR-100.

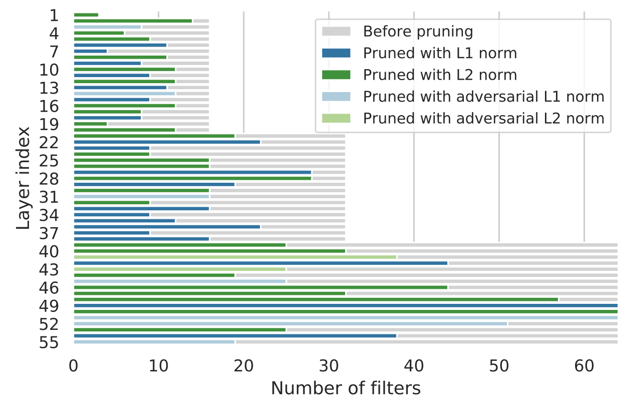

Figure 5. Visualization of the conventional and adversarial criteria for ResNet-56 on CIFAR-10. The grey strip indicates the layers before pruning. Different blue and green colors represent different pruning criteria.

## 4.4. ResNet on ILSVRC-2012

For the ILSVRC-2012 dataset, we test our method on ResNet-50. Same as [20], we do not prune the projection shortcuts. Tab. 3 shows that our LPFC outperforms existing methods on ILSVRC-2012. For the random initialized ResNet-50, when our LFPC prunes 7.3% more FLOPs than FPGM [20], the accuracy is even higher than FPGM [20]. For pruning the pre-trained ResNet-50, we achieve 92.04% top-5 accuracy when we prune 60.8% FLOPs. While the previous methods (CP [22], LFC [44], ELR [49]) have lower top-5 accuracy when pruning less FLOPs (50%). ThiNet [36] also has a lower accuracy than our LFPC when its acceleration ratio is lower than ours. The superior performance comes from that our method considers the different filter distribution of different layers.

## 4.5. More Explorations

**Adversarial Criteria.** To further validate the effeteness of our LFPC, we add the adversarial criteria, which is the adversarial version of the current pruning criteria, to our system. For example, conventional norm-based criteria keep the filters with **large** $\ell_p$-norm. In contrast, adversarial norm-based criteria keep the filters with **small** $\ell_p$-norm, which could be formulate as $\mathrm{Crit}^{(l)}(\mathbf{W}_i^{(l)}) = \frac{1}{\|\mathbf{W}_i^{(l)}\|_p}$ for $i \in [1, C_O^{(l)}]$.

The learned criteria for ResNet-56 on CIFAR-10 are shown in Fig. 5. In this experiment, we utilize four criteria, including $\ell_1$-norm, $\ell_2$-norm, adversarial $\ell_1$-norm, adversarial $\ell_2$-norm. As shown in Tab. 5, for all the 55 criteria for ResNet-56, the adversarial criteria only account for a small proportion (16.4%). This means that our LFPC successfully selects conventional criteria and circumvents the adversarial criteria, which would be another evidence of the effectiveness of our LFPC.

**Criteria During Training** The learned criteria during training DCS is shown in Fig. 6. A small strip of a specific color means the layer of the network utilizes a corresponding pruning criterion at the current epoch. We find that the sampler gradually converges to a regular pattern of criteria, which provides stable guidance for the next pruning step.

| Setting | Adversarial criteria (%) | Conventional criteria(%) | FLOPs ↓(%) | Accuracy (%) |
|---|---|---|---|---|
| w Adv | 16.4% | 83.6% | 58.0 | 93.09 (±0.09) |
| w/o Adv | 0 | 100% | 53.0 | 93.45 (±0.13) |

Table 5. Analysis of adversarial criteria. "w Adv" and "w/o Adv" denote containing the adversarial criteria or not, respectively.

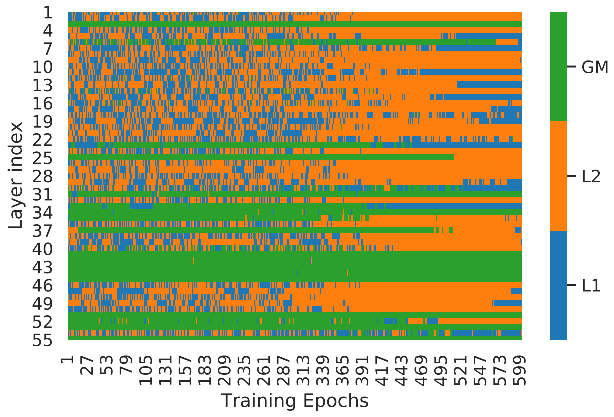

Figure 6. The learned criteria during training the criteria sampler. The L1, L2, and GM denote conventional $\ell_1$-norm, $\ell_2$-norm, and geometric median criteria, respectively.

**Retraining Scheduler.** We compare the cosine scheduler [35] and step scheduler [20] during retraining. When pruning 47.6% FLOPs of the ResNet-56, cosine scheduler can achieve 93.56(±0.15)% accuracy, while step scheduler can obtain 93.54(±0.16)% accuracy. It shows that LFPC can achieve a slightly stable result with a cosine scheduler.

## 5. Conclusion and Future Work

In this paper, we propose a new learning filter pruning criteria (LFPC) framework for deep CNNs acceleration. Different from the existing methods, LFPC explicitly considers the difference between layers and adaptively selects a set of suitable criteria for different layers. To learn the criteria effectively, we utilize Gumbel-softmax to make the criteria sampler process differentiable. LFPC achieves comparable performance with state-of-the-art methods in several benchmarks. In the future, we could consider utilizing more kinds of criteria into LFPC and combine it with other acceleration algorithms, *e.g.*, matrix decomposition [28], to improve the performance further. Moreover, it is meaningful to adopt the proposed method to recent compact ConvNets such as MobileNets.

## References

[1] A. T. Benjamin and J. J. Quinn. *Proofs that really count: the art of combinatorial proof*. Number 27. MAA, 2003. 4

[2] M. A. Carreira-Perpinán and Y. Idelbayev. "learning-compression" algorithms for neural net pruning. In *CVPR*, 2018. 2

[3] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, 2019. 3

[4] X. Ding, G. Ding, Y. Guo, and J. Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *CVPR*, 2019. 3

[5] X. Ding, G. Ding, Y. Guo, J. Han, and C. Yan. Approximated oracle filter pruning for destructive cnn width optimization. In *ICML*, 2019. 2, 3

[6] X. Dong, S. Chen, and S. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *NeurIPS*, 2017. 2

[7] X. Dong, J. Huang, Y. Yang, and S. Yan. More is less: A more complicated network with less inference complexity. In *CVPR*, 2017. 5, 6, 7

[8] X. Dong and Y. Yang. Network pruning via transformable architecture search. In *NeurIPS*, 2019. 5, 6

[9] X. Dong and Y. Yang. Searching for a robust neural architecture in four gpu hours. In *CVPR*, 2019. 4

[10] A. Dubey, M. Chatterjee, and N. Ahuja. Coreset-based neural network compression. *arXiv preprint arXiv:1807.09810*, 2018. 2

[11] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019. 7

[12] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient DNNs. In *NeurIPS*, 2016. 2

[13] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2015. 2

[14] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015. 1, 2

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 5

[16] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 5

[17] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, and Y. Yang. Asymptotic soft filter pruning for deep convolutional neural networks. *IEEE Transactions on Cybernetics*, pages 1–11, 2019. 2

[18] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*, 2018. 2, 5, 6, 7

[19] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, 2018. 2, 3, 6

[20] Y. He, P. Liu, Z. Wang, and Y. Yang. Pruning filter via geometric median for deep convolutional neural networks acceleration. In *CVPR*, 2019. 1, 2, 3, 4, 5, 6, 7, 8

[21] Y. He, P. Liu, L. Zhu, and Y. Yang. Meta filter pruning to accelerate deep convolutional neural networks. *arXiv preprint arXiv:1904.03961*, 2019. 2

[22] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017. 2, 5, 6, 7, 8

[23] Q. Huang, K. Zhou, S. You, and U. Neumann. Learning to prune filters in convolutional neural networks. In *WACV*, 2018. 2, 3

[24] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. In *ECCV*, 2018. 2, 6, 7

[25] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017. 4

[26] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009. 5

[27] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient ConvNets. In *ICLR*, 2017. 1, 2, 3, 4, 5, 6, 7

[28] S. Lin, R. Ji, C. Chen, D. Tao, and J. Luo. Holistic cnn compression via low-rank decomposition with knowledge transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(12):2889–2905, 2019. 8

[29] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*, 2018. 2

[30] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *CVPR*, 2019. 2, 6

[31] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. In *ICLR*, 2019. 3

[32] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017. 2

[33] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, T. K.-T. Cheng, and J. Sun. Metapruning: Meta learning for automatic neural network channel pruning. *arXiv preprint arXiv:1903.10258*, 2019. 3

[34] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018. 6, 7

[35] I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. In *ICLR*, 2017. 6, 8

[36] J.-H. Luo, J. Wu, and W. Lin. ThiNet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017. 2, 5, 6, 7, 8

[37] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017. 4

[38] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz. Importance estimation for neural network pruning. In *CVPR*, 2019. 2, 6, 7

[39] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. In *ICLR*, 2017. 2

[40] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 5, 6

[41] H. Peng, J. Wu, S. Chen, and J. Huang. Collaborative channel pruning for deep networks. In *ICML*, 2019. 2

[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet large scale visual recognition challenge. *IJCV*, 2015. 5

[43] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 5

[44] P. Singh, V. K. Verma, P. Rai, and V. P. Namboodiri. Leveraging filter correlations for deep model compression. *arXiv preprint arXiv:1811.10559*, 2018. 6, 7, 8

[45] X. Suau, L. Zappella, V. Palakkode, and N. Apostoloff. Principal filter analysis for guided network compression. *arXiv preprint arXiv:1807.10585*, 2018. 2

[46] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 7

[47] K. Tang, Y. Niu, J. Huang, J. Shi, and H. Zhang. Unbiased scene graph generation from biased training. In *CVPR*, 2020. 1

[48] F. Tung and G. Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *CVPR*, 2018. 2

[49] D. Wang, L. Zhou, X. Zhang, X. Bai, and J. Zhou. Exploring linear relationship in feature map subspace for convnets compression. *arXiv preprint arXiv:1803.05729*, 2018. 2, 6, 7, 8

[50] X. Wang, Z. Zheng, Y. He, F. Yan, Z. Zeng, and Y. Yang. Progressive local filter pruning for image retrieval acceleration. *arXiv preprint arXiv:2001.08878*, 2020. 2

[51] J. Ye, X. Lu, Z. Lin, and J. Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *ICLR*, 2018. 2

[52] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *ICML Deep Learning Workshop*, 2015. 2

[53] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1909.08174*, 2019. 2, 3

[54] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. NISP: Pruning networks using neuron importance score propagation. In *CVPR*, 2018. 2, 3, 6, 7

[55] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016. 5

[56] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. *ECCV*, 2018. 2

[57] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu. Discrimination-aware channel pruning for deep neural networks. In *NeurIPS*, 2018. 2

[58] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 3