UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Accelerating Deep Convolutional Neural Networks via Filter Pruning

by Yang He

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of Yi Yang

University of Technology Sydney
Faculty of Engineering and Information Technology
July 2021

Certificate of Authorship/Originality

I, Yang He, declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctor of Philosophy in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature: Yang He Production Note: Signature removed prior to publication.

Date: 1/July/2021

Acknowledgements

First, I would like to express my tremendous gratitude to my supervisor, Professor Yi Yang, I really appreciate his mentoring and cultivating in my research and my life. Although I changed my major for my PhD study and had little knowledge in the area computer science when I entered the group, Yi has always been patient with me and given me quite a lot guidance. Every time I encountered difficulties, Yi is always here to help me. It is really my fortune to have Yi as my supervisor.

I am grateful for all my colleagues and teammates in Yi's group. I convey my gratitude to my co-supervisor, Prof. Liang Zheng, for his help and advises during this research. I feel so lucky to met Prof. Yanwei Fu, Guoliang Kang and Xuanyi Dong that they teached me quite a lot about doing research and writing code when I am a novice of the research field. It is so important to express my gratitude to Ping Liu, who is reliable and heart-warming, and gives me many kinds of help and advice. I am happy to collaborate with many creative teammates in our team and I really appreciate the kind and useful suggestions given by them. Moreover, I am thankful for Prof. Hanwang Zhang for his guidance and advises during my visiting to NTU. I have met quite a lot friends at NTU and thanks for your support. I also thank Qi Yao for helping me with my PhD study.

Finally, I want to express my special gratitude to my parents, who give me endless love, help, and encouragement in my life. How lucky to be your child.

Thanks for all the people that ever helped me and encouraged me.

Yang He

Sydney, Australia, 2021.

List of Publications

Journal Papers

J-1. **Yang He**, Xuanyi Dong, Guoliang Kang, Yanwei Fu, Chenggang Yan, and Yi Yang, "Asymptotic Soft Filter Pruning for Deep Convolutional Neural Networks," IEEE Transactions on Cybernetics, vol. 50, no. 8, pp. 3594–3604, 2019.

Conference Papers

C-1. **Yang He**, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang, "Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks," in International Joint Conference on Artificial Intelligence (IJCAI), pp. 2234–2240, 2018.

C-2. **Yang He**, Ping Liu, Ziwei Wang, Zhilan Hu, Yi Yang,"Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4340–4349, 2019.

C-3. **Yang He**, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, Yi Yang, "Learning Filter Pruning Criteria for Deep Convolutional Neural Networks Acceleration," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2009–2018, 2020.

Contents

List of Figures

List of Tables

ABSTRACT

Accelerating Deep Convolutional Neural Networks via Filter Pruning

by

Yang He

The superior performance of deep Convolutional Neural Networks (CNNs) usually comes from the deeper and wider architectures, which cause the prohibitively expensive computation cost. To reduce the computational cost, works on model compression and acceleration have recently emerged. Among all the directions for this goal, filter pruning has attracted attention in recent studies due to its efficacy. For a better understanding of filter pruning, this thesis explores different aspects of filter pruning, including pruning mechanism, pruning ratio, pruning criteria, and automatic pruning. First, we improve the pruning mechanism with soft filter pruning so that the mistaken pruned filters can have a chance to be recovered. Second, we consider the asymptotic pruning rate to reduce the sudden information loss in the pruning process. Then we explore the pruning criteria to better measure the importance of filters. Finally, we propose the automatic pruning method to save human labor. Our methods lead to superior convolutional neural network acceleration results.

Dissertation directed by Professor Yi Yang

ReLER, Australian Artificial Intelligence Institute, School of Software

Chapter 1

Introduction

1.1 Background

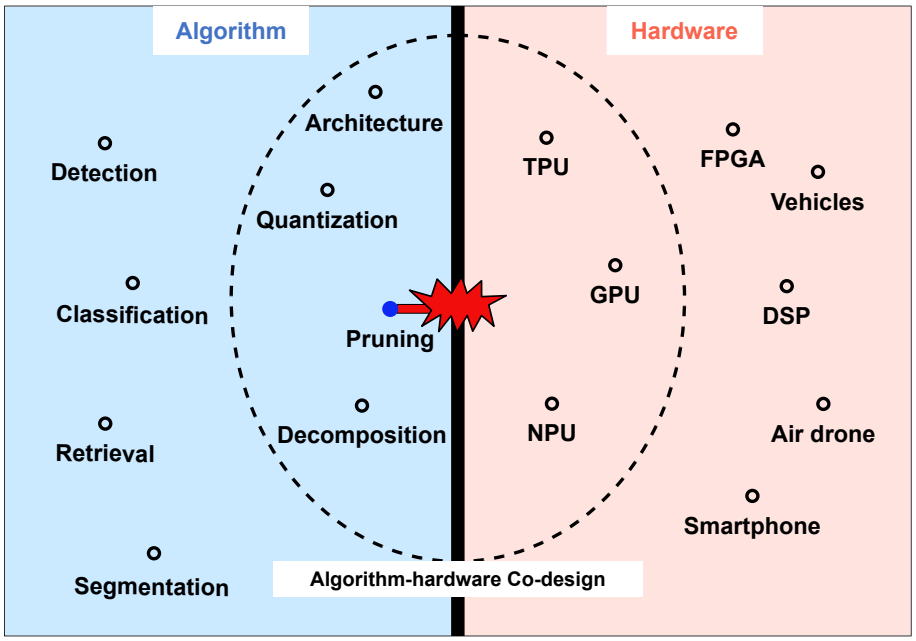


Figure 1.1 : Neural network acceleration is about breaking the existing "wall" between algorithm and hardware to fit machine learning algorithms into the resource-constrained hardware platforms.

The superior performance of deep neural networks usually comes from the deeper and wider architectures, which cause the prohibitively expensive computation cost. The storage, memory, and computation costs of these cumbersome models significantly exceed the computing limitation of resource constraint platforms such as self-driving cars, unmanned aerial vehicle, and neural processing unit (NPU) in chips. Therefore, it is essential to maintain the deep neural network models to have

a relatively low computational cost but ensure high accuracy in real-world applications.

As shown in in Fig. 1.1, the left side are different kinds of algorithms, including detection, classification, image retrieval, segmentation. The right side are some hardware platforms such as FPGA, vehicles, Digital Signal Processors (DSP), air drones, and smart photos. Inside the black dashed circle, it is about Algorithm-hardware Co-design. For example, some novel neural architectures, quantization methods, pruning methods, and network tensor decomposition are proposed to fit the hardware. Similarly, Tensor Processing Unit (TPU), Graphics Processing Unit (GPU), and Neural Processing Unit (NPU) are proposed to fit the algorithms. Neural network acceleration is about breaking the existing "wall" between algorithm and hardware, to fit machine learning algorithms into the resource-constrained hardware platforms.

Deep neural network compression and accelerating methods can be roughly divided into four categories, namely, *novel architecture*, *matrix decomposition*, *quantization*, and *pruning*. *novel architecture* such as depth-wise separable convolution of MobileNet and pointwise group convolution and channel shuffle of ShuffleNet can lead to lightweight deep neural networks. *Matrix decomposition* proposes to representing the weight matrix of the convolutional network as a low-rank product of two smaller matrices to reduce the calculation. *Quantization* aims to store and use only low-precision weights during the inference procedure so that the storage and computation cost can be reduced. *Pruning*-based approaches aim to remove the unnecessary connections of the neural network [2], and have recently attracted attention.

Recent developments on pruning can be divided into two categories, *i.e.*, weight pruning [2–5] and filter pruning [1, 6–54]. Weight pruning directly removes weight

values and causes unstructured sparsities. On the contrary, filter pruning removes the whole filter to obtain the models with structured sparsity. Therefore, filter pruning is preferred compared to weight pruning because filter pruning could make the pruned model more structural and achieve practical acceleration. The conventional 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.

Research Objectives. As depicted in Fig. 1.1, neural network acceleration is about bridging the hardware and machine learning algorithm. The research objectives of this thesis is to build new paradigms of pruning methods to obtain neural networks that fit into resource-constrained hardware platforms.

1.2 Thesis Organization

An introduction to the battery storage system, including the lithium-ion battery and the BMS are given in the previous sections. According to the overview, there is space to improve the efficiency and reliability of the battery storage system from the measurement, modelling, and states estimation perspectives. This thesis is organised as follows:

- *Chapter 2*: This chapter proposed a Soft Filter Pruning (SFP) method to accelerate the inference procedure of deep Convolutional Neural Networks (CNNs). Specifically, the proposed SFP enables the pruned filters to be updated when training the model after pruning. SFP has two advantages over previous works: (1) Larger model capacity; (2) Less dependence on the pre-trained model.

- *Chapter 3*: This chapter proposed a Soft Filter Pruning (SFP) method to accelerate the inference procedure of deep Convolutional Neural Networks (CNNs).

Specifically, the proposed SFP enables the pruned filters to be updated when training the model after pruning. SFP has two advantages over previous works: (1) Larger model capacity; (2) Less dependence on the pre-trained model.

- *Chapter 4*: In this chapter, we analyze previous norm-based criterion and point out that its effectiveness depends on two requirements that are not always met: (1) the norm deviation of the filters should be large; (2) the minimum norm of the filters should be small. To solve this problem, we propose a novel filter pruning method, namely Filter Pruning via Geometric Median (FPGM), to compress the model regardless of those two requirements.

- *Chapter 5*: In this chapter, we propose an Asymptotic Soft Filter Pruning (ASFP) method to accelerate the inference procedure of the deep neural networks. Specifically, we prune few filters at first and asymptotically prune more filters during the training procedure. With asymptotic pruning, the information of the training set would be gradually concentrated in the remaining filters, so the subsequent training and pruning process would be stable.

- *Chapter 6*: In this chapter, we propose Learning Filter Pruning Criteria (LFPC) to adaptively select the appropriate pruning criteria for different functional layers. 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. Besides, when evaluating the sampled criteria, LFPC comprehensively considers the contribution of all the layers at the same time.

- *Chapter 7*: A brief summary of the thesis contents and its contributions are given in the final chapter. Recommendations for future research are given in addition.

Chapter 2

Related Works

2.1 Matrix Decomposition

To reduce the computation costs of the convolutional layers, previous works propose to represent the weight matrix of the convolutional network as a low-rank product of two smaller matrices [55–60]. Then the calculation of production of one large matrix turns to the production of two smaller matrices. For example, [55] exploits cross-channel or filter redundancy to construct a low-rank basis of filters that are rank-1 in the spatial domain. [56, 57] develops an effective solution to the resulting nonlinear optimization problem without the need for stochastic gradient descent (SGD). [58] proposes a new algorithm for computing the low-rank tensor decomposition for removing the redundancy in the convolution kernel. [59] presents an efficient general sparse-with-dense matrix multiplication implementation that is applicable to the convolution of feature maps with kernels of arbitrary sparsity patterns. [60] presents a new tensor-factorized NN (TFNN), which tightly integrates TF and NN for multiway feature extraction and classification under a unified discriminative objective.

However, the computational cost of tensor decomposition operation is expensive, which is not friendly to train deep CNNs. Besides, there exists an increasing usage of 1×1 convolution kernel in some recent neural networks, such as the bottleneck block structure of ResNet [61], cases where it is difficult to apply matrix decomposition.

2.2 Low Precision

Some other researchers focus on low-precision implementation to compress and accelerate CNN models [5, 62–65]. Zhou *et al.* [62] propose trained ternary quantization to reduce the precision of weights in neural networks to ternary values. The authors of [63] present incremental network quantization, targeting to convert pre-trained full-precision CNN model into a low-precision version efficiently. [64] introduces a method to train Binarized Neural Networks (BNNs) - neural networks with binary weights and activations at run-time. [65] proposes two efficient approximations to standard convolutional neural networks: Binary-Weight-Networks and XNOR-Networks.

In this situation, only low-precision weights are stored and used during the inference procedure, with the storage and computation cost being dramatically reduced. This direction can work cooperatively with pruning methods [5] to achieve better performance.

2.3 Weight Pruning

Recent work [4, 5, 66] prunes weights of neural networks. For example, [4] proposed an iterative weight pruning method by discarding the small weights whose values are below the threshold. [66] proposed the dynamic network surgery to reduce the training iteration while maintaining good prediction accuracy. [3, 67] leveraged the sparsity property of feature maps or weight parameters to accelerate the CNN models. To this end, [3] proposed the Structured Sparsity Learning (SSL) method to regularize filter, channel, filter shape and depth structures. [67] applied the group-sparsity regularization on the loss function to shrink some entire groups of weights towards zeros. However, weight pruning always leads to unstructured models, so the model cannot leverage the existing efficient BLAS libraries in practice. Therefore,

it is difficult for weight pruning to achieve realistic speedup. Meanwhile, Bayesian methods [68] are also applied to network pruning. [68] extends the soft weight sharing to obtain a sparse and compressed network. [**?**] uses hierarchical priors to prune nodes and utilizes the posterior uncertainties to encode the weights. [**?**] uses variational inference to learn the dropout rate, which can then be used to prune the network. However, these methods are evaluated on rather small datasets such as MNIST [69] and CIFAR-10 [70].

2.4 Filter Pruning

Pruning the filters [1,14,15,19] leads to the removal of the corresponding feature maps, thus not only reducing the storage usage on devices but also decreasing the memory footprint consumption. Considering whether to utilize the training data to determine the pruned filters, the filter pruning methods are roughly divided into two categories, data dependent and data independent filter pruning. The latter method is more efficient than the former since training data may not be available during the pruning process.

Data Dependent Filter Pruning. Some approaches [14–17, 19, 21, 22, 71, 72] utilize the training data to determine the pruned filters. The authors of [71] minimize the reconstruction error of activation maps to obtain a decomposition of convolutional layers. Luo *et al.* [19] adopt the statistics information from the next layer to guide the importance evaluation of filters. [8] proposes an inherently data-driven that which uses Principal Component Analysis (PCA) to specify the proportion of the energy that should be preserved. [73] applies subspace clustering to feature maps to eliminate the redundancy in convolutional filters. [15] imposes sparsity regularization on the scaling factors of the network. [14] utilizes the LASSO regression to select channels. [17] proposes to minimize the reconstruction error of important responses in the "final response layer", and derives a closed-form solution

to it for pruning neurons in earlier layers.

Data Independent Filter Pruning. Concurrently with our work, some data independent filter pruning strategies [1, 6, 20, 74] have been explored. Li *et al.* [1] explore the sensitivity of layers for filter pruning and utilize a ℓ_1-norm criterion to prune unimportant filters. Ye *et al.* [20] prune models by enforcing sparsity on the scaling parameters of batch normalization layers. [6] proposes to select filters with a ℓ_2-norm criterion and prune those selected filters in a soft manner. [9] uses spectral clustering on filters to select unimportant ones. However, for all these filter pruning methods, the representative capacity of the neural network after pruning is seriously affected by a smaller optimization space. Besides, the information loss at the beginning is significant and unrecoverable.

2.5 Neural Architecture Search

DARTS [75] proposes to search the architecture efficiently using gradient descent. ENAS [76] proposes to share parameters between child models, so the search process is fast. Note that the sub-architectures in DARTS and child models in ENAS need to be trained during searching, while our sub-networks do not. Our method is also different from one-shot NAS [77] which generates the weights for sampled architecture using a HyperNet. [78] presents the MobileNet-V3 based on a combination of complementary search techniques as well as a novel architecture design. [79] proposes an automated mobile neural architecture search (MNAS) approach, which explicitly incorporates model latency into the main objective so that the search can identify a model that achieves a good trade-off between accuracy and latency. [80] addresses the high memory consumption issue of differentiable NAS and reduces the computational cost (GPU hours and GPU memory) to the same level of regular training. [81] proposes a differentiable neural architecture search (DNAS) framework that uses gradient-based methods to optimize ConvNet architectures, avoiding enu-

merating and training individual architectures separately as in previous methods. Our LFPC method shares some basic ideas with Neural Architecture Search.

2.6 Vision Transformer Compression

Vision transformer is firstly proposed in ViT [82], which utilizes attention not on pixels, but instead on small patches of the images. Becasuse Vision Transformers require lots of computational costs, some methods [83–91] are proposed for vision transformer compression and acceleration. Reduce the **input image tokens** is a direction. For example, DynamicViT [85] prunes redundant tokens progressively. EViT [86] reorganizes the token to reduce the computational cost of multi-head self-attention. Another direction is to reduce the parameters of the **network itself**. In this direction, pruning methods can be utilized. NViT [91] uses structural pruning with latency-aware regularization on all parameters of the vision transformer. UVC [90] assembles three effective techniques, pruning, layer skipping, and knowledge distillation, for efficient ViT. VTP [87] prunes the unimportant features of the ViT with sparsity regularization. AutoFormer [88] uses architecture search framework for vision transformer search. As-ViT [89] automatically scales up ViTs.

2.7 Applications

Pruning methods have lots of industrial applications, including robotics, computer vision, and the medical industry. Specifically, for inspection robots which are typically limited in computing and memory resources, [92] introduces a solution based on network pruning using Taylor expansion to utilize pre-trained deep convolutional neural networks for efficient edge computing. [93] finds that, with some forms of pruning, a large portion of the connections can be pruned without strongly affecting robot capabilities. [94] finds that incorporating some forms of pruning in neuro-evolution leads to almost equally effective controllers for a locomotion task,

and for centralized as well as distributed controllers.

Pruning methods are also utilized in [95] for real-time fruit detection and localization in orchards. [96] focuses on real-time and accurate detection of apple flowers using pruning. [97] uses pruning methods for synthetic aperture radar (SAR) ship real-time detection. [98] proposes pruning methods for tracking social distances or recognizing face masks. Pruning methods can also be applied to hardware. In [99], the authors use pruning on three hardware architectures, namely CPU, GPU, and Intel Movidius Neural Computer Stick (NCS). Medical image analysis also uses pruning methods. [100] utilizes pruning for biomedical image segmentation. [101] introduces pruning to skin lesion image segmentation. [102] improves the defense of the medical imaging system against adversarial examples with pruning.

Chapter 3

Soft Filter Pruning

3.1 Introduction

The superior performance of deep CNNs usually comes from the deeper and wider architectures, which cause the prohibitively expensive computation cost. Even if we use more efficient architectures, such as residual connection [61] or inception module [103], it is still difficult in deploying the state-of-the-art CNN models on mobile devices. For example, ResNet-152 has 60.2 million parameters with 231MB storage spaces; besides, it also needs more than 380MB memory footprint and six seconds (11.3 billion float point operations, FLOPs) to process a single image on CPU. The storage, memory, and computation of this cumbersome model significantly exceed the computing limitation of current mobile devices. Therefore, it is essential to maintain the small size of the deep CNN models which has relatively low computational cost but high accuracy in real-world applications.

Recent efforts have been made either on directly deleting weight values of filters [4] (i.e., weight pruning) or totally discarding some filters (i.e., filter pruning) [1, 14, 19]. However, the weight pruning may result in the unstructured sparsity of filters, which may still be less efficient in saving the memory usage and computational cost, since the unstructured model cannot leverage the existing high-efficiency BLAS libraries. In contrast, the filter pruning enables the model with structured sparsity and more efficient memory usage than weight pruning, and thus takes full advantage of BLAS libraries to achieve a more realistic acceleration. Therefore, the filter pruning is more advocated in accelerating the networks.

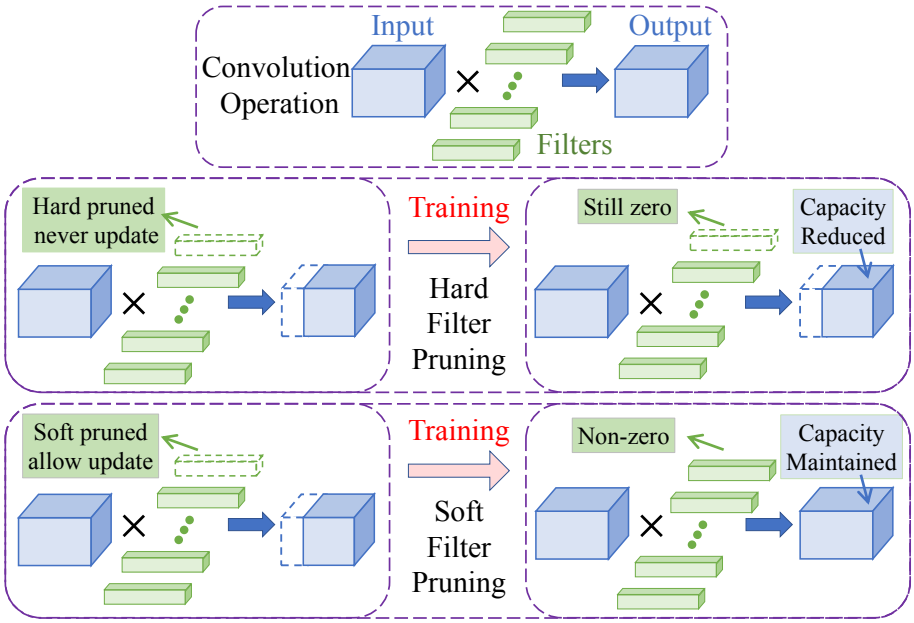


Figure 3.1 : **Hard Filter Pruning** *v.s.* **Soft Filter Pruning**. We mark the pruned filter as the green dashed box. For the hard filter pruning, the pruned filters are always ***fixed*** during the whole training procedure. Therefore, the model capacity is reduced and thus harms the performance because the dashed blue box is useless during training. On the contrary, our SFP ***allows*** the pruned filters to be updated during the training procedure. In this way, the model capacity is recovered from the pruned model, and thus leads a better accuracy.

Nevertheless, most of the previous works on filter pruning still suffer from the problems of (1) *the model capacity reduction* and (2) *the dependence on pre-trained model*. Specifically, as shown in Fig. 3.1, most previous works conduct the "hard filter pruning", which directly delete the pruned filters. The discarded filters will reduce the model capacity of original models, and thus inevitably harm the performance. Moreover, to maintain a reasonable performance with respect to the full models, previous works [1, 14, 19] always fine-tuned the hard pruned model after pruning the filters of a pre-trained model, which however has low training efficiency and often requires much more training time than the traditional training schema.

To address the above mentioned two problems, we propose a novel Soft Filter Pruning (SFP) approach. The SFP dynamically prunes the filters in a soft manner. Particularly, before first training epoch, the filters of almost all layers with small ℓ_2-norm are selected and set to zero. Then the training data is used to update the pruned model. Before the next training epoch, our SFP will prune a new set of filters of small ℓ_2-norm. These training process is continued until converged. Finally, some filters will be selected and pruned without further updating. The SFP algorithm enables the compressed network to have a larger model capacity, and thus achieve a higher accuracy than others.

Contributions. We highlight three contributions: (1) We propose SFP to allow the pruned filters to be updated during the training procedure. This soft manner can dramatically maintain the model capacity and thus achieves the superior performance. (2) Our acceleration approach can train a model from scratch and achieve better performance compared to the state-of-the-art. In this way, the fine-tuning procedure and the overall training time is saved. Moreover, using the pre-trained model can further enhance the performance of our approach to advance the state-of-the-art in model acceleration. (3) The extensive experiment on two benchmark datasets demonstrates the effectiveness and efficiency of our SFP. We accelerate ResNet-110 by two times with about 4% relative accuracy improvement on CIFAR-10, and also achieve state-of-the-art results on ILSVRC-2012.

3.2 Related Works

Most previous works on accelerating CNNs can be roughly divided into three categories, namely, *matrix decomposition*, *low-precision weights*, and *pruning*. In particular, the *matrix decomposition* of deep CNN tensors is approximated by the product of two low-rank matrices [55, 56, 58]. This can save the computational cost. Some works [62, 63] focus on compressing the CNNs by using *low-precision weights*.

Pruning-based approaches aim to remove the unnecessary connections of the neural network [1, 4]. Essentially, the work of this paper is based on the idea of pruning techniques; and the approaches of matrix decomposition and low-precision weights are orthogonal but potentially useful here – it may be still worth simplifying the weight matrix after pruning filters, which would be taken as future work.

Weight Pruning. Many recent works [4,5,66] pruning weights of neural network resulting in small models. For example, [4] proposed an iterative weight pruning method by discarding the small weights whose values are below the threshold. [66] proposed the dynamic network surgery to reduce the training iteration while maintaining a good prediction accuracy. [3, 67] leveraged the sparsity property of feature maps or weight parameters to accelerate the CNN models. A special case of weight pruning is neuron pruning. However, pruning weights always leads to unstructured models, so the model cannot leverage the existing efficient BLAS libraries in practice. Therefore, it is difficult for weight pruning to achieve realistic speedup.

Filter Pruning. Concurrently with our work, some filter pruning strategies [1, 14, 15, 19] have been explored. Pruning the filters leads to the removal of the corresponding feature maps. This not only reduces the storage usage on devices but also decreases the memory footprint consumption to accelerate the inference. [1] uses ℓ_1-norm to select unimportant filters and explores the sensitivity of layers for filter pruning. [15] introduces ℓ_1 regularization on the scaling factors in batch normalization (BN) layers as a penalty term, and prune channel with small scaling factors in BN layers. [16] proposes a Taylor expansion based pruning criterion to approximate the change in the cost function induced by pruning. [19] adopts the statistics information from next layer to guide the importance evaluation of filters. [14] proposes a LASSO-based channel selection strategy, and a least square reconstruction algorithm to prune filers. However, for all these filter pruning methods, the representative capacity of neural network after pruning is seriously affected by smaller

optimization space.

Discussion. To the best of our knowledge, there is only one approach that uses the soft manner to prune weights [66]. We would like to highlight our advantages compared to this approach as below: (1) Our SPF focuses on the filter pruning, but they focus on the weight pruning. As discussed above, weight pruning approaches lack the practical implementations to achieve the realistic acceleration. (2) [66] paid more attention to the model compression, whereas our approach can achieve both compression and acceleration of the model. (3) Extensive experiments have been conducted to validate the effectiveness of our proposed approach both on large-scale datasets and the state-of-the-art CNN models. In contrast, [66] only had the experiments on Alexnet which is more redundant the advanced models, such as ResNet.

3.3 Methodology

3.3.1 Preliminaries

We will formally introduce the symbol and annotations in this section. The deep CNN network can be parameterized by $\{\mathbf{W}^{(i)} \in \mathbb{R}^{N_{i+1}\times N_i\times K\times K}, 1 \leq i \leq L\}$ $\mathbf{W}^{(i)}$ denotes a matrix of connection weights in the i-th layer. N_i denotes the number of input channels for the i-th convolution layer. L denotes the number of layers. The shapes of input tensor $\mathbf{U}$ and output tensor $\mathbf{V}$ are $N_i \times H_i \times W_i$ and $N_{i+1} \times H_{i+1} \times W_{i+1}$, respectively. The convolutional operation of the i-th layer can be written as:

$$\mathbf{V}_{i,j} = \mathcal{F}_{i,j} * \mathbf{U} \quad \textbf{for } 1 \leq j \leq N_{i+1}, \tag{3.1}$$

where $\mathcal{F}_{i,j} \in \mathbb{R}^{N_i\times K\times K}$ represents the j-th filter of the i-th layer. $\mathbf{W}^{(i)}$ consists of

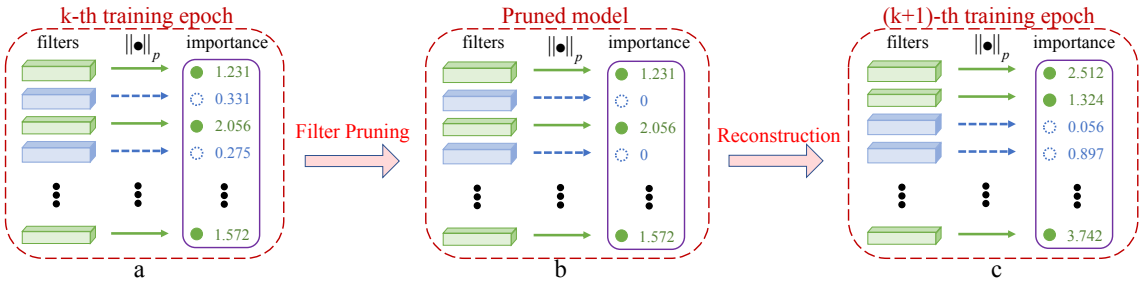


Figure 3.2 : Overview of SFP. At the end of each training epoch, we prune the filters based on their importance evaluations. The filters are ranked by their ℓ_p-norms (purple rectangles) and the small ones (blue circles) are selected to be pruned. After filter pruning, the model undergoes a reconstruction process where pruned filters are capable of being reconstructed (i.e., updated from zeros) by the forward-backward process. **(a)**: filter instantiations before pruning. **(b)**: filter instantiations after pruning. **(c)**: filter instantiations after reconstruction.

$\{\mathcal{F}_{i,j}, 1 \leq j \leq N_{i+1}\}$. The $\mathbf{V}_{i,j}$ represents the j-th output feature map of the i-th layer.

Pruning filters can remove the output feature maps. In this way, the computational cost of the neural network will reduce remarkably. Let us assume the pruning rate of SFP is P_i for the i-th layer. The number of filters of this layer will be reduced from N_{i+1} to $N_{i+1}(1 - P_i)$, thereby the size of the output tensor $\mathbf{V}_{i,j}$ can be reduced to $N_{i+1}(1 - P_i) \times H_{i+1} \times W_{i+1}$. As the output tensor of i-th layer is the input tensor of $i + 1$-th layer, we can reduce the input size of i-th layer to achieve a higher acceleration ratio.

3.3.2 Soft Filter Pruning (SFP)

Most of previous filter pruning works [1, 14, 15, 19] compressed the deep CNNs in a hard manner. We call them as the hard filter pruning. Typically, these algorithms firstly prune filters of a single layer of a pre-trained model and fine-tune the pruned

model to complement the degrade of the performance. Then they prune the next layer and fine-tune the model again until the last layer of the model is pruned. However, once filters are pruned, these approaches will not update these filters again. Therefore, the model capacity is drastically reduced due to the removed filters; and such a hard pruning manner affects the performance of the compressed models negatively.

As summarized in Alg. 1, the proposed SFP algorithm can dynamically remove the filters in a soft manner. Specifically, the key is to keep updating the pruned filters in the training stage. Such an updating manner brings several benefits. It not only keeps the model capacity of the compressed deep CNN models as the original models, but also avoids the greedy layer by layer pruning procedure and enable pruning almost *all layers* at the same time. More specifically, our approach can prune a model either in the process of training from scratch, or a pre-trained model. In each training epoch, the full model is optimized and trained on the training data. After each epoch, the ℓ_2-norm of all filters are computed for each weighted layer and used as the criterion of our filter selection strategy. Then we will prune the selected filters by setting the corresponding filter weights as zero, which is followed by next training epoch. Finally, the original deep CNNs are pruned into a compact and efficient model. The details of SFP is illustratively explained in Alg. 1, which can be divided into the following four steps.

Filter selection. We use the ℓ_p-norm to evaluate the importance of each filter as Eq. equation 3.2. In general, the convolutional results of the filter with the smaller ℓ_p-norm lead to relatively lower activation values; and thus have a less numerical impact on the final prediction of deep CNN models. In term of this understanding, such filters of small ℓ_p-norm will be given high priority of being pruned than those of higher ℓ_p-norm. Particularly, we use a pruning rate P_i to select $N_{i+1}P_i$ unimportant filters for the i-th weighted layer. In other words, the lowest $N_{i+1}P_i$ filters are

Algorithm 1 Algorithm Description of SFP

Input: training data: $\mathbf{X}$, pruning rate: P_i

the model with parameters $\mathbf{W} = \{\mathbf{W}^{(i)}, 0 \leq i \leq L\}$.

Initialize the model parameter $\mathbf{W}$

for $epoch = 1$; $epoch \leq epoch_{max}$; $epoch++$ **do**

Update the model parameter $\mathbf{W}$ based on $\mathbf{X}$

for $i = 1$; $i \leq L$; $i++$ **do**

Calculate the ℓ_2-norm for each filter $\|\mathcal{F}_{i,j}\|_2, 1 \leq j \leq N_{i+1}$

Zeroize $N_{i+1}P_i$ filters by ℓ_2-norm filter selection

end for

end for

Obtain the compact model with parameters $\mathbf{W}^*$ from $\mathbf{W}$

Output: The compact model and its parameters $\mathbf{W}^*$

selected, e.g., the blue filters in Fig. 3.2. In practice, ℓ_2-norm is used based on the empirical analysis.

$$\|\mathcal{F}_{i,j}\|_p = \sqrt[p]{\sum_{n=1}^{N_i}\sum_{k_1=1}^{K}\sum_{k_2=1}^{K}|\mathcal{F}_{i,j}(n,k_1,k_2)|^p}, \tag{3.2}$$

Filter Pruning. We set the value of selected $N_{i+1}P_i$ filters to zero (see the filter pruning step in Fig. 3.2). This can temporarily eliminate their contribution to the network output. Nevertheless, in the following training stage, we still allow these selected filters to be updated, in order to keep the representative capacity and the high performance of the model.

In the filter pruning step, we simply prune *all* the weighted layers at the same time. In this way, we can prune each filter in parallel, which would cost negligible computation time. In contrast, the previous filter pruning methods always conduct

layer by layer greedy pruning. After pruning filters of one single layer, existing methods always require training to converge the network [14, 19]. This procedure cost much extra computation time, especially when the depth increases. Moreover, we use the *same* pruning rate for *all* weighted layers. Therefore, we need only one hyper-parameter $P_i = P$ to balance the acceleration and accuracy. This can avoid the inconvenient hyper-parameter search or the complicated sensitivity analysis [1]. As we allow the pruned filters to be updated, the model has a large model capacity and becomes more flexible and thus can well balance the contribution of each filter to the final prediction.

Reconstruction. After the pruning step, we train the network for one epoch to reconstruct the pruned filters. As shown in Fig. 3.2, the pruned filters are updated to non-zero by back-propagation. In this way, SFP allows the pruned model to have the same capacity as the original model during training. In contrast, hard filter pruning decreases the number of feature maps. The reduction of feature maps would dramatically reduce the model capacity, and further harm the performance. Previous pruning methods usually require a pre-trained model and then fine-tune it. However, as we integrate the pruning step into the normal training schema, our approach can train the model from scratch. Therefore, the fine-tuning stage is no longer necessary for SFP. As we will show in experiments, the network trained from scratch by SFP can obtain the competitive results with the one trained from a well-trained model by others. By leveraging the pre-trained model, SFP obtains a much higher performance and advances the state-of-the-art.

Obtaining Compact Model. SFP iterates over the filter selection, filter pruning and reconstruction steps. After the model gets converged, we can obtain a sparse model containing many "zero filters". One "zero filter" corresponds to one feature map. The features maps, corresponding to those "zero filters", will always be zero during the inference procedure. There will be no influence to remove these filters

as well as the corresponding feature maps. Specifically, for the pruning rate P_i in the i-th layer, only $N_{i+1}(1-P_i)$ filters are non-zero and have an effect on the final prediction. Consider pruning the previous layer, the input channel of i-th layer is changed from N_i to $N_i(1-P_{i-1})$. We can thus re-build the i-th layer into a smaller one. Finally, a compact model $\{\mathbf{W}^{*(i)} \in \mathbb{R}^{N_{i+1}(1-P_i)\times N_i(1-P_{i-1})\times K\times K}\}$ is obtained.

3.3.3 Computation Complexity Analysis

Theoretical speedup analysis. Suppose the filter pruning rate of the ith layer is P_i, which means the $N_{i+1} \times P_i$ filters are set to zero and pruned from the layer, and the other $N_{i+1} \times (1-P_i)$ filters remain unchanged, and suppose the size of the input and output feature map of ith layer is $H_i \times W_i$ and $H_{i+1} \times W_{i+1}$. Then after filter pruning, the dimension of useful output feature map of the ith layer decreases from $N_{i+1} \times H_{i+1} \times W_{i+1}$ to $N_{i+1}(1-P_i) \times H_{i+1} \times W_{i+1}$. Note that the output of ith layer is the input of $(i+1)$ th layer. And we further prunes the $(i+1)$th layer with a filter pruning rate P_{i+1}, then the calculation of $(i+1)$th layer is decrease from $N_{i+2} \times N_{i+1} \times k^2 \times H_{i+2} \times W_{i+2}$ to $N_{i+2}(1-P_{i+1}) \times N_{i+1}(1-P_i) \times k^2 \times H_{i+2} \times W_{i+2}$. In other words, a proportion of $1-(1-P_{i+1}) \times (1-P_i)$ of the original calculation is reduced, which will make the neural network inference much faster.

Realistic speedup analysis. In theoretical speedup analysis, other operations such as batch normalization (BN) and pooling are negligible comparing to convolution operations. Therefore, we consider the FLOPs of convolution operations for computation complexity comparison, which is commonly used in previous work [1, 19]. However, reduced FLOPs cannot bring the same level of realistic speedup because non-tensor layers (e.g., BN and pooling layers) also need the inference time on GPU [19]. In addition, the limitation of IO delay, buffer switch and efficiency of BLAS libraries also lead to the wide gap between theoretical and realistic speedup ratio. We compare the theoretical and realistic speedup in Section 3.4.3.

Depth	Method	Fine-tune?	Baseline Accu. (%)	Accelerated Accu. (%)	Accu. Drop (%)	FLOPs	Pruned FLOPs(%)
20	[104]	N	91.53	91.43	0.10	3.20E7	20.3
	Ours(10%)	N	**92.20 ± 0.18**	**92.24 ± 0.33**	**-0.04**	3.44E7	15.2
	Ours(20%)	N	**92.20 ± 0.18**	91.20 ± 0.30	1.00	2.87E7	29.3
	Ours(30%)	N	**92.20 ± 0.18**	90.83 ± 0.31	1.37	**2.43E7**	**42.2**
32	[104]	N	92.33	90.74	1.59	4.70E7	31.2
	Ours(10%)	N	**92.63 ± 0.70**	**93.22 ± 0.09**	**-0.59**	5.86E7	14.9
	Ours(20%)	N	**92.63 ± 0.70**	90.63 ± 0.37	0.00	4.90E7	28.8
	Ours(30%)	N	**92.63 ± 0.70**	90.08 ± 0.08	0.55	**4.03E7**	**41.5**
56	[1]	N	93.04	91.31	1.75	9.09E7	27.6
	[1]	Y	93.04	93.06	-0.02	9.09E7	27.6
	[14]	N	92.80	90.90	1.90	-	50.0
	[14]	Y	92.80	91.80	1.00	-	50.0
	Ours(10%)	N	**93.59 ± 0.58**	**93.89 ± 0.19**	**-0.30**	1.070E8	14.7
	Ours(20%)	N	**93.59 ± 0.58**	93.47 ± 0.24	0.12	8.98E7	28.4
	Ours(30%)	N	**93.59 ± 0.58**	93.10 ± 0.20	0.49	7.40E7	41.1
	Ours(30%)	Y	**93.59 ± 0.58**	93.78 ± 0.22	-0.19	7.40E7	41.1
	Ours(40%)	N	**93.59 ± 0.58**	92.26 ± 0.31	1.33	**5.94E7**	**52.6**
	Ours(40%)	Y	**93.59 ± 0.58**	93.35 ± 0.31	0.24	**5.94E7**	**52.6**
110	[1]	N	93.53	92.94	0.61	1.55E8	38.6
	[1]	Y	93.53	93.30	0.20	1.55E8	38.6
	[104]	N	93.63	93.44	0.19	-	34.2
	Ours(10%)	N	**93.68 ± 0.32**	93.83 ± 0.19	-0.15	2.16E8	14.6
	Ours(20%)	N	**93.68 ± 0.32**	**93.93 ± 0.41**	**-0.25**	1.82E8	28.2
	Ours(30%)	N	**93.68 ± 0.32**	93.38 ± 0.30	0.30	**1.50E8**	**40.8**
	Ours(30%)	Y	**93.68 ± 0.32**	93.86 ± 0.21	-0.18	**1.50E8**	**40.8**

Table 3.1 : Comparison of pruning ResNet on CIFAR-10. In “Fine-tune?” column, “Y” and “N” indicate whether to use the pre-trained model as initialization or not, respectively. The “Accu. Drop” is the accuracy of the pruned model minus that of the baseline model, so negative number means the accelerated model has a higher accuracy than the baseline model. A smaller number of ”Accu. Drop” is better.

3.4 Evaluation and Results

3.4.1 Benchmark Datasets and Experimental Setting

Our method is evaluated on two benchmarks: CIFAR-10 [70] and ILSVRC-2012 [105]. The CIFAR-10 dataset contains 50,000 training images and 10,000 testing images, which are categorized into 10 classes. ILSVRC-2012 is a large-scale dataset containing 1.28 million training images and 50k validation images of 1,000 classes. Following the common setting in [14, 19, 104], we focus on pruning the challenging ResNet model in this paper. SFP should also be effective on different computer vision tasks, such as [106–112], and we will explore this in future.

In the CIFAR-10 experiments, we use the default parameter setting as [113] and follow the training schedule in [114]. On ILSVRC-2012, we follow the same parameter settings as [61,113]. We use the same data argumentation strategies with PyTorch official examples [115].

We conduct our SFP operation at the end of every training epoch. For pruning a scratch model, we use the normal training schedule. For pruning a pre-trained model, we reduce the learning rate by 10 compared to the schedule for the scratch model. We run each experiment three times and report the "mean ± std". We compare the performance with other state-of-the-art acceleration algorithms, e.g., [1, 14, 19, 104].

3.4.2 ResNet on CIFAR-10

Settings. For CIFAR-10 dataset, we test our SFP on ResNet-20, 32, 56 and 110. We use several different pruning rates, and also analyze the difference between using the pre-trained model and from scratch.

Results. Tab. 3.1 shows the results. Our SFP could achieve a better performance than the other state-of-the-art hard filter pruning methods. For example, [1] use the hard pruning method to accelerate ResNet-110 by 38.6% speedup ratio with

0.61% accuracy drop when without fine-tuning. When using pre-trained model and fine-tuning, the accuracy drop becomes 0.20%. However, we can accelerate the inference of ResNet-110 to 40.8% speed-up with only 0.30% accuracy drop without fine-tuning. When using the pre-trained model, we can even outperform the original model by 0.18% with about more than 40% FLOPs reduced.

These results validate the effectiveness of SFP, which can produce a more compressed model with comparable performance to the original model.

3.4.3 ResNet on ILSVRC-2012

Table 3.2 : Comparison of pruning ResNet on ImageNet. "Fine-tune?" and "Accu. Drop" have the same meaning with Tab. 3.1.

Depth	Method	Fine-tune?	Top-1 Accu. Baseline(%)	Top-1 Accu. Accelerated(%)	Top-5 Accu. Baseline(%)	Top-5 Accu. Accelerated(%)	Top-1 Accu. Drop(%)	Top-5 Accu. Drop(%)	Pruned FLOPs(%)
18	[104]	N	69.98	66.33	89.24	86.94	3.65	2.30	34.6
	Ours(30%)	N	**70.28**	**67.10**	**89.63**	**87.78**	**3.18**	**1.85**	**41.8**
34	[104]	N	73.42	**72.99**	91.36	**91.19**	**0.43**	**0.17**	24.8
	[1]	Y	73.23	72.17	-	-	1.06	-	24.2
	Ours(30%)	N	**73.92**	71.83	**91.62**	90.33	2.09	1.29	**41.1**
50	[14]	Y	-	-	92.20	90.80	-	1.40	**50.0**
	[19]	Y	72.88	72.04	91.14	90.67	**0.84**	**0.47**	36.7
	Ours(30%)	N	**76.15**	**74.61**	**92.87**	**92.06**	1.54	0.81	41.8
	Ours(30%)	Y	**76.15**	62.14	**92.87**	84.60	14.01	8.27	41.8
101	Ours(30%)	N	**77.37**	77.03	**93.56**	93.46	0.34	0.10	**42.2**
	Ours(30%)	Y	**77.37**	**77.51**	**93.56**	**93.71**	**-0.14**	**-0.20**	**42.2**

Settings. For ILSVRC-2012 dataset, we test our SFP on ResNet-18, 34, 50 and 101; and we use the same pruning rate 30% for all the models. All the convolutional layer of ResNet are pruned with the same pruning rate at the same time. (We do not prune the projection shortcuts for simplification, which only need negligible time and do not affect the overall cost.)

Model	Baseline time (ms)	Pruned time (ms)	Realistic Speed-up(%)	Theoretical Speed-up(%)
ResNet-18	37.10	26.97	27.4	41.8
ResNet-34	63.97	45.14	29.4	41.1
ResNet-50	135.01	94.66	29.8	41.8
ResNet-101	219.71	148.64	32.3	42.2

Table 3.3 : Comparison on the theoretical and realistic speedup. We only count the time consumption of the forward procedure.

Results. Tab. 3.2 shows that SFP outperforms other state-of-the-art methods. For ResNet-34, SFP without fine-tuning achieves more inference speedup to the hard pruning method [19], but the accuracy of our pruned model exceeds their model by 2.57%. Moreover, for pruning a pre-trained ResNet-101, SFP reduces more than 40% FLOPs of the model with even 0.2% top-5 accuracy increase, which is the state-of-the-art result. In contrast, the performance degradation is inevitable for hard filter pruning method. Maintained model capacity of SFP is the main reason for the superior performance. In addition, the non-greedy all-layer pruning method may have a better performance than the locally optimal solution obtained from previous greedy pruning method, which seems to be another reason. Occasionally, large performance degradation happens for the pre-trained model (e.g., 14.01% top-1 accuracy drop for ResNet-50). This will be explored in our future work.

To test the realistic speedup ratio, we measure the forward time of the pruned models on one GTX1080 GPU with a batch size of 64 (shown in Tab. 3.3). The gap between theoretical and realistic model may come from and the limitation of IO delay, buffer switch and efficiency of BLAS libraries.

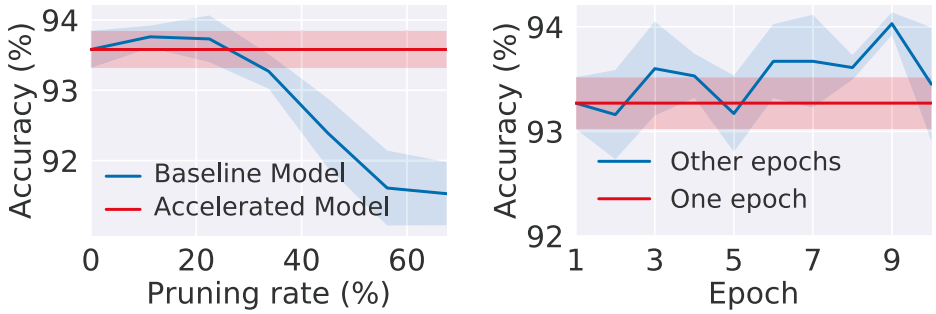
3.4.4 Ablation Study

We conducted extensive ablation studies to further analyze each component of SFP.

Filter Selection Criteria. The magnitude based criteria such as ℓ_p-norm are widely used to filter selection because computational resources cost is small [1]. We compare the ℓ_2-norm and ℓ_1-norm. For ℓ_1-norm criteria, the accuracy of the model under pruning rate 10%, 20%, 30% are 93.68±0.60%, 93.68±0.76% and 93.34±0.12%, respectively. While for ℓ_2-norm criteria, the accuracy are 93.89±0.19%, 93.93±0.41% and 93.38±0.30%, respectively. The performance of ℓ_2-norm criteria is slightly better than that of ℓ_1-norm criteria. The result of ℓ_2-norm is dominated by the largest element, while the result of ℓ_1-norm is also largely affected by other small elements. Therefore, filters with some large weights would be preserved by the ℓ_2-norm criteria. So the corresponding discriminative features are kept so the performance of the pruned model is better.

Varying pruning rates. To comprehensively understand SFP, we test the accuracy of different pruning rates for ResNet-110, shown in Fig. 3.3(a). As the pruning rate increases, the accuracy of the pruned model first rises above the baseline model and then drops approximately linearly. For the pruning rate between 0% and about 23%, the accuracy of the accelerated model is higher than the baseline model. This shows that our SFP has a regularization effect on the neural network because SFP reduces the over-fitting of the model.

Sensitivity of SFP interval. By default, we conduct our SFP operation at the end of every training epoch. However, different SFP intervals may lead to different performance; so we explore the sensitivity of SFP interval. We use the ResNet-110 under pruning rate 30% as a baseline, and change the SFP interval from one epoch to ten epochs, as shown in Fig. 3.3(b). It is shown that the model accuracy has no large



(a) Different Pruning Rates (b) Different SFP Intervals

Figure 3.3 : Accuracy of ResNet-110 on CIFAR-10 regarding different hyper-parameters. (Solid line and shadow denotes the mean and standard deviation of three experiment, respectively.)

fluctuation along with the different SFP intervals. Moreover, the model accuracy of most (80%) intervals surpasses the accuracy of one epoch interval. Therefore, we can even achieve a better performance if we fine-tune this parameter.

Selection of pruned layers. Previous works always prune a portion of the layers of the network. Besides, different layers always have different pruning rates. For example, [1] only prunes insensitive layers, [19] skips the last layer of every block of the ResNet, and [19] prunes more aggressive for shallower layers and prune less for deep layers. Similarly, we compare the performance of pruning first and second layer of all basic blocks of ResNet-110. We set the pruning rate as 30%. The model with all the first layers of blocks pruned has an accuracy of $93.96 \pm 0.13\%$, while that with the second layers of blocks pruned has an accuracy of $93.38 \pm 0.44\%$. Therefore, different layers have different sensitivity for SFP, and careful selection of pruned layers would potentially lead to performance improvement, although more hyper-parameters are needed.

3.5 Conclusion

In this paper, we propose a soft filter pruning (SFP) approach to accelerate the deep CNNs. During the training procedure, SFP allows the pruned filters to be updated. This soft manner can maintain the model capacity and thus achieve the superior performance. Remarkably, SFP can achieve the competitive performance compared to the state-of-the-art without the pre-trained model. Moreover, by leveraging the pre-trained model, SFP achieves a better result and advances the state-of-the-art.

Chapter 4

Asymptotic Soft Filter Pruning

4.1 Introduction

Convolutional Neural Networks (CNNs) have demonstrated state-of-the-art performance in computer vision tasks [107, 116, 117]. The superior performance of deep CNNs usually comes from the deeper and wider architectures [61, 103, 116, 118, 119], which cause the prohibitively expensive computation cost. The storage, memory, and computation of these cumbersome models significantly exceed the computing limitation of current mobile devices or drones. [4] shows that running a 1 billion connection neural network at 20Hz would require 12.8W just for DRAM access. Besides, the authors of [120, 121] claim that VGGNet [118] requires 321.1 MBytes memory and 236 mW power for a batch of three frames and processes 0.7 frame per second even when it is deployed on the optimized energy-efficient chip. Therefore, it is essential to maintain the deep CNN models to have a relatively low computational cost but ensure high accuracy in real-world applications.

Pruning deep CNNs [122–124] is an important direction for accelerating the networks. Recent efforts have been made either on directly deleting some weight values of filters [4] (*i.e.*, weight pruning) or totally discarding some filters (*i.e.*, filter pruning) [1, 14, 19]. However, weight pruning results in the *unstructured sparsity* of filters. Since the unstructured model cannot leverage the existing high-efficiency BLAS (Basic Linear Algebra Subprograms) libraries, weight pruning is not efficient in saving the memory usage and computational cost. In contrast, filter pruning enables the model with *structured sparsity*, taking full advantage of BLAS libraries

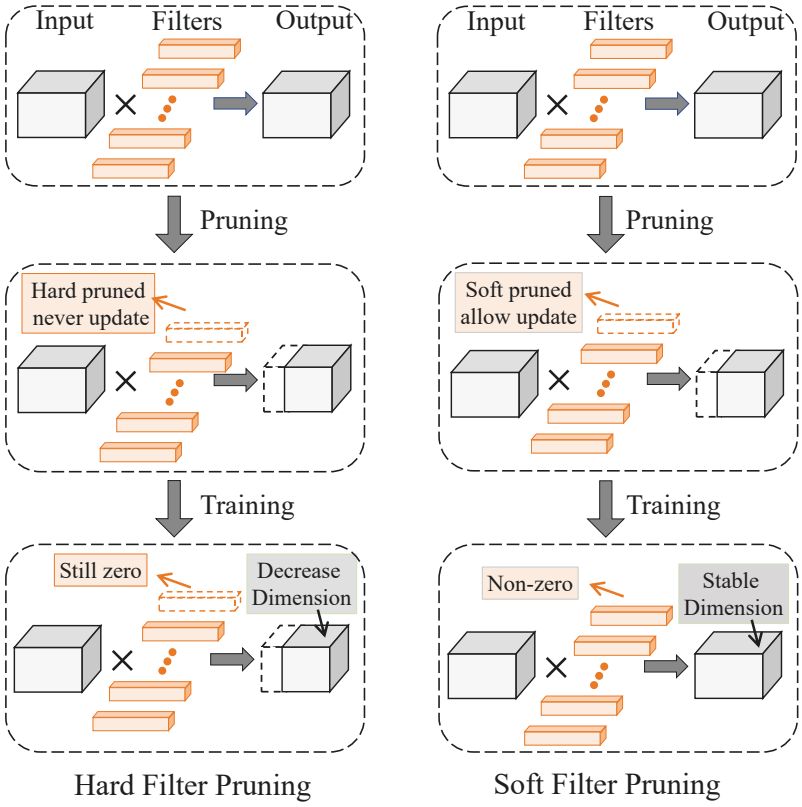


Figure 4.1 : Hard filter pruning *v.s.* soft filter pruning. We mark the pruned filter as the orange dashed box. For the hard filter pruning, the pruned filters are always ***fixed*** during the whole training procedure. Therefore, the model capacity is reduced and thus harms the performance because the dashed blue box is useless during training. On the contrary, our soft pruning method allows the pruned filters to be ***updated*** during the training procedure. In this way, the model capacity is recovered from the pruned model and thus leads a better accuracy.

to achieve more efficient memory usage and more realistic acceleration. Therefore, filter pruning is more favored in accelerating the networks.

Nevertheless, most filter pruning algorithms suffer from two problems: (1) *the model capacity reduction* and (2) *the unrecoverable filter information loss*. Specifically, as shown in Figure 4.1, most researchers conduct the "hard filter pruning (HFP)" [1, 14, 19], then the pruned filters are directly deleted and have no possi-

bility to be recovered. The discarded filters will reduce the optimization space and model capacity, and thus it is unfavorable for the pruned network to learn enough knowledge. To alleviate this problem, they use pre-training to maintain a good performance, which in turn induces much more training time. Furthermore, existing methods directly prune a large number of filters, which contain information of training set, at first. This process leads to severe and unrecoverable information loss and thus inevitably degrades the performance.

To solve the above two problems, we propose ASFP, which prunes the convolutional filters dynamically. Particularly, before the first training epoch, the filters with small ℓ_2-norm are selected and set to zero. Then we retrain the model, and the previously pruned filters could be updated. Before the next training epoch, we will prune a *new set* of filters with small ℓ_2-norm. These training processes are continued until converged. Lastly, some filters with smallest ℓ_2-norm will be selected and pruned without further updating. This soft manner enables the compressed network to have a larger optimization space and model capacity. Hence it is easier for the model to learn from the training data, and achieve higher accuracy even without the pre-training process.

In addition, we prune the network asymptotically —— pruning few filters at first, and more filters at later training epochs. If few filters are pruned at first, little pre-trained information would be lost, so it is easy for the model to recover from pruning. With the following iterative training and soft pruning, the training set information would be gradually concentrated in some important filters. At the same time, the training and pruning process would be stable, as the information is lost gradually instead of suddenly.

We highlight the following three contributions of ASFP:

(1) We propose a soft manner to allow the previous pruned filters to be reconstructed

during training. This soft manner could significantly maintain the model capacity, which enables the network to be trained and pruned simultaneously from scratch.

(2) To avoid severe information loss, we propose to asymptotically prune the filters, which makes the subsequent training and pruning process more stable.

(3) Experiments on CIFAR-10 and ImageNet demonstrate the effectiveness and efficiency of the proposed ASFP.

4.2 Related Work

CNN accelerating methods can be roughly divided into four categories, namely, *matrix decomposition*, *low-precision weights*, *weight pruning*, and *filter pruning*.

4.2.1 Matrix Decomposition

To reduce the computation costs of the convolutional layers, previous work propose to representing the weight matrix of the convolutional network as a low-rank product of two smaller matrices [55–60]. Then the calculation of production of one large matrix turns to the production of two smaller matrices. However, the computational cost of tensor decomposition operation is expensive, which is not friendly to train deep CNNs. Besides, there exists an increasing usage of 1×1 convolution kernel in some recent neural networks, such as the bottleneck block structure of ResNet [61], cases where it is difficult to apply matrix decomposition.

4.2.2 Low Precision

Some other researchers focus on low-precision implementation to compress and accelerate CNN models [5, 62–65]. Zhou *et al.* [62] propose trained ternary quantization to reduce the precision of weights in neural networks to ternary values. The authors of [63] present incremental network quantization, targeting to convert pre-trained full-precision CNN model into a low-precision version efficiently. In this

situation, only low-precision weights are stored and used during the inference procedure, with the storage and computation cost being dramatically reduced.

4.2.3 Weight Pruning

Recent work [4, 5, 66] prunes weights of neural networks. For example, [4] proposed an iterative weight pruning method by discarding the small weights whose values are below the threshold. [3, 67] leveraged the sparsity property of feature maps or weight parameters to accelerate the CNN models. However, weight pruning always leads to unstructured models, so the model cannot leverage the existing efficient BLAS libraries in practice. Therefore, it is difficult for weight pruning to achieve realistic speedup. Meanwhile, Bayesian methods [68] are also applied to network pruning. However, these methods are evaluated on rather small datasets such as MNIST [69] and CIFAR-10 [70].

4.2.4 Filter Pruning

Pruning the filters [1,14,15,19] leads to the removal of the corresponding feature maps, thus not only reducing the storage usage on devices but also decreasing the memory footprint consumption. Considering whether to utilize the training data to determine the pruned filters, the filter pruning methods are roughly divided into two categories, data dependent and data independent filter pruning. The latter method is more efficient than the former since training data may not be available during the pruning process.

Data Dependent Filter Pruning. Some approaches [14–17, 19, 21, 22, 71, 72] utilize the training data to determine the pruned filters. The authors of [71] minimize the reconstruction error of activation maps to obtain a decomposition of convolutional layers. Luo *et al.* [19] adopt the statistics information from the next layer to guide the importance evaluation of filters.

Data Independent Filter Pruning. Concurrently with our work, some data independent filter pruning strategies [1, 6, 20, 74] have been explored. Li *et al.* [1] explore the sensitivity of layers for filter pruning and utilize a ℓ_1-norm criterion to prune unimportant filters. Ye *et al.* [20] prune models by enforcing sparsity on the scaling parameters of batch normalization layers. However, for all these filter pruning methods, the representative capacity of the neural network after pruning is seriously affected by smaller optimization space. Besides, the information loss at the beginning is significant and unrecoverable.

4.3 Methodology

4.3.1 Preliminary

We formally introduce the symbol and notations in this section. The deep CNN network can be parameterized by $\{\mathbf{W}^{(i)} \in \mathbb{R}^{N_{i+1}\times N_i\times K\times K}, 1 \leq i \leq L\}$ [*]. $\mathbf{W}^{(i)}$ denotes a matrix of connection weights in the i_{th} layer. N_i denotes the number of input channels for the i_{th} convolution layer. L denotes the number of layers. The shapes of input tensor $\mathbf{U}$ and output tensor $\mathbf{V}$ are $N_i \times H_i \times W_i$ and $N_{i+1} \times H_{i+1} \times W_{i+1}$, respectively. The convolutional operation of the i_{th} layer can be written as:

$$\mathbf{V}_{i,j} = \mathcal{F}_{i,j} * \mathbf{U} \ , \ \textbf{for } 1 \leq j \leq N_{i+1}, \tag{4.1}$$

where $\mathcal{F}_{i,j} \in \mathbb{R}^{N_i\times K\times K}$ represents the j_{th} filter of the i_{th} layer, and $\mathbf{V}_{i,j}$ represents the j_{th} output feature map of the i_{th} layer. $\mathbf{W}^{(i)}$ consists of $\{\mathcal{F}_{i,j}, 1 \leq j \leq N_{i+1}\}$.

Pruning filters can remove the output feature maps. In this way, the computational cost of the neural network will reduce remarkably. Let us assume the pruning rate is P_i for the i_{th} layer. The number of filters of this layer will be reduced from N_{i+1} to $N_{i+1}(1 - P_i)$, thereby the size of the output tensor $\mathbf{V}_{i,j}$ can be reduced to $N_{i+1}(1 - P_i) \times H_{i+1} \times W_{i+1}$.

[*]Fully-connected layers can be viewed as convolutional layers with $k = 1$

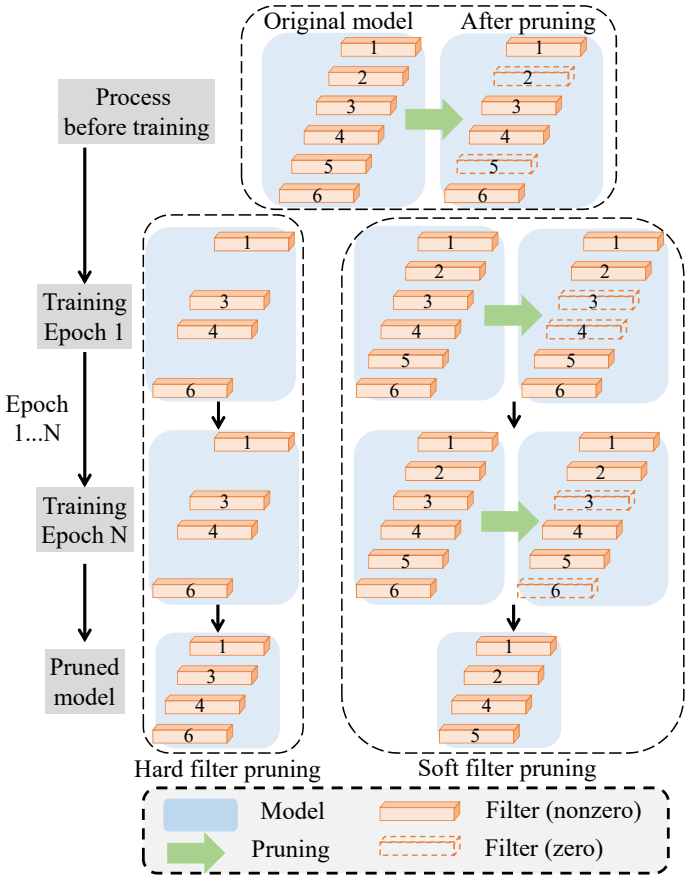


Figure 4.2 : Pruning and training schedule of HFP and SFP. Before training, we first select some filters with pre-defined importance evaluations. HFP directly deletes these filters before training, while for SFP, those are set to zero and kept. During training (epoch 1 to N), the model size is smaller than the original one for HFP. While for SFP, the zero value filters (filter 2 and 5) become non-zero after training epoch 1. Then we evaluate the importance of filters again and prune filter 3 and 4. The model size would not be reduced but be the same as the original one. When training is finished, the final pruned model is the model at epoch N for HFP. While for SFP, we delete the zero value filters (filter 3 and 6) at epoch N to get the final pruned model.

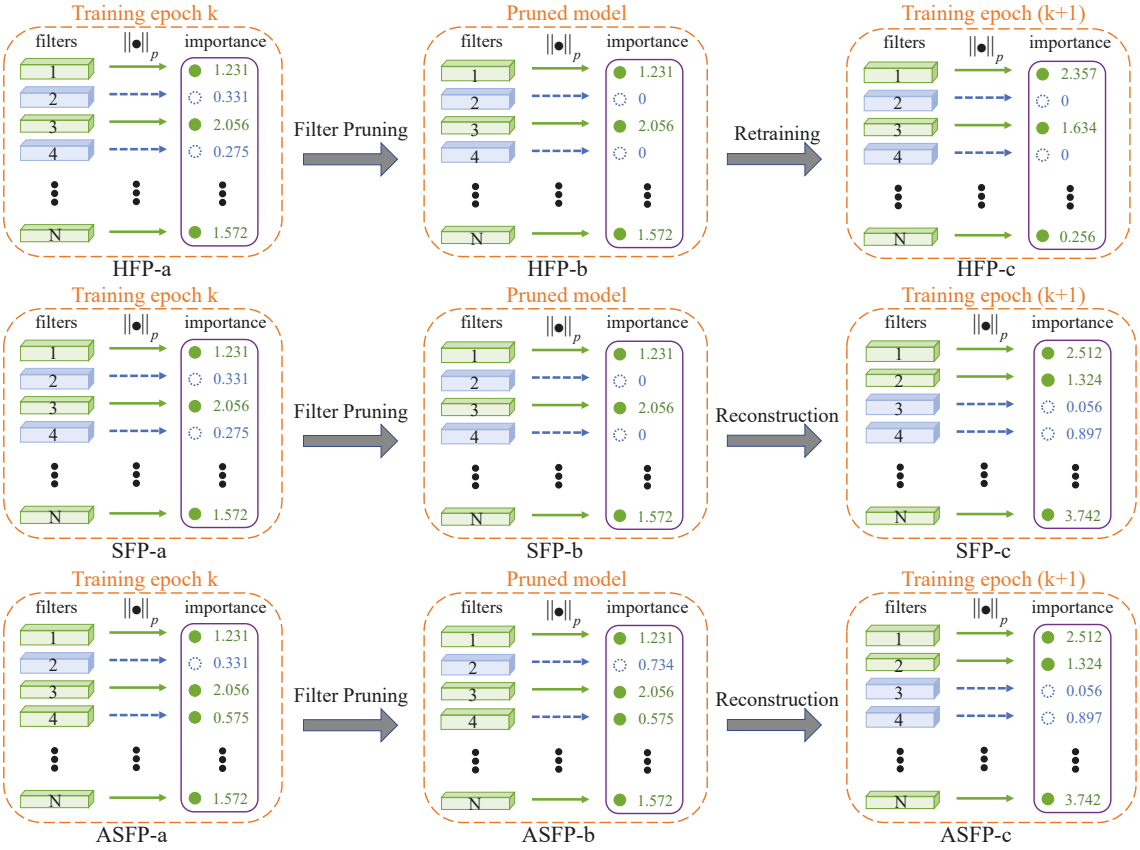


Figure 4.3 : Overview of HFP (first row), SFP (second row) and ASFP (third row). **(a)**: filter instantiations before pruning. **(b)**: filter instantiations after pruning. **(c)**: filter instantiations after reconstruction. The filters are ranked by their ℓ_p-norms and the small ones (purple rectangles) are selected to be pruned.

4.3.2 Pruning with Hard Manner

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$ and a desired sparsity level κ (*i.e.*, the number of remaining non-zero filters), HFP can formulated as:

$$\min_{\mathcal{F}} \ell(\mathcal{F}; \mathcal{D}) = \min_{\mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathcal{F}; (\mathbf{x}_i, \mathbf{y}_i)) \ , \tag{4.2}$$

$$\text{s.t.} \quad \mathcal{F} \in \mathbb{R}^{N \times K \times K}, \quad N(\mathcal{F}) \leq \kappa \ .$$

Here, $\ell(\cdot)$ is the standard loss function (*e.g.*, cross-entropy loss), $\mathcal{F}$ is the set of filters of the neural network, and N is the cardinality of the filter set. Typically,

HFP firstly prunes filters of a single layer of a pre-trained model and fine-tune the pruned model to complement the performance degradation. Then they prune the next layer and fine-tune the model again until the last layer is pruned. Once the filters are pruned, HFP will not update these filters again.

The full training schedule of HFP is shown in the first column of Figure 4.2, and the detailed pruning process of HFP is shown in the first row of Figure 4.3. First, some of the filters with small ℓ_p-norm (filter 2 and 4 in HFP-a, marked in blue) are selected and pruned. After retraining, these pruned filters are not able to be updated again thus the ℓ_p-norm of these filters would be zero during all the training epochs (filter 2 and 4 in HFP-b). Meanwhile, the remaining filters (filter 1, 3 and N in HFP-c, marked in green) might be updated to another value after retraining to make up for the performance degradation due to pruning. After several epochs of retraining to converge the model, a compact model is obtained to accelerate the inference.

4.3.3 Pruning with Soft Manner

SFP can dynamically remove the filters in a soft manner. Specifically, the key is to keep updating the pruned filters in the training stage. The full training schedule of SFP is shown in the second column of Figure 4.2. Such an updating manner brings several benefits. It not only keeps the model capacity of the pruned models as the original models but also avoids the greedy layer by layer pruning procedure and enables pruning *all* convolutional layers at the same time. For SFP, the constrain in the Eq. 4.2 changes to:

$$\|\mathcal{F}\|_0 \leq \kappa, \quad \mathsf{N}(\mathcal{F}) = N_{i+1}. \tag{4.3}$$

Here, $\|\cdot\|_0$ is the standard L_0 norm. After soft pruning, the number of filters $\mathsf{N}(\mathcal{F})$ is still the same as that of the original model (N_{i+1}).

The second row of Figure 4.3 explains the detailed process of SFP. First, the ℓ_2-norms of all filters are computed for each weighted layer and used as our filter selection criterion. Second, some filters with a small ℓ_p-norm (filter 2 and 4 in SFP-a, marked in blue) are selected, and we prune those filters by setting the corresponding filter weights as zero (filter 2 and 4 in SFP-b). Then we retrain the model. As we allow the pruned filters to be updated during retraining, the pruned filters become nonzero again (filter 2 and 4 in SFP-c) due to back-propagation. After iterations of pruning and reconstruction to converge the model, we delete the unimportant filters and obtain a compact and efficient model.

4.3.4 Asymptotic Soft Filter Pruning (ASFP)

It is known that all the filters of the pre-trained model have the information of the training set. Therefore, pruning those informative filters would cause information loss. This situation is especially difficult for pruning the models that pre-trained on large datasets, or pruning a large number of filters of small models, as the lost information is rather massive. Therefore, we propose to pruning the neural network asymptotically.

Specifically, we use a small pruning rate at early epochs, and gradually increase the pruning rate later, until we reach the goal pruning rate P_i at the last retraining epoch. The optimization problem of ASFP:

$$\min_{\mathcal{F}} \ell(\mathcal{F}; \mathcal{D}) = \min_{\mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathcal{F}; (\mathbf{x}_i, \mathbf{y}_i)) \ , \tag{4.4}$$

$$\text{s.t.} \quad \|\mathcal{F}\|_0 \leq \kappa_{epoch}, \quad \mathsf{N}(\mathcal{F}) = N_{i+1}.$$

Here, κ_{epoch} means the sparsity level changes with the training epoch. Comparing Eq. 4.3 and Eq. 4.4, the difference between SFP and ASFP is whether the pruning rate is changing with epochs. For SFP, the κ in Eq. 4.3 is a pre-defined constant regarding the number of remaining non-zero filters, so it would not change during

the training process. In contrast, κ_{epoch} in Eq. 4.4 is a function of *epoch* and would change with the epoch number of training. Therefore, SFP is a special case of ASFP when the pruning rate P_i during all training epochs are the same.

4.3.4.1 Asymptotic Filter Selection

We use the ℓ_p-norm to evaluate the importance of each filter as Eq. 4.5. In general, the convolutional results of the filters with the smaller ℓ_p-norm would lead to relatively lower activation values, and thus have a less numerical impact on the final prediction of deep CNN models. In term of this understanding, such filters of small ℓ_p-norm will be given higher priority of being pruned than those of higher ℓ_p-norm:

$$\|\mathcal{F}_{i,j}\|_p = \sqrt[p]{\sum_{n=1}^{N_i}\sum_{k_1=1}^{K}\sum_{k_2=1}^{K}|\mathcal{F}_{i,j}(n,k_1,k_2)|^p}. \tag{4.5}$$

In practice, we use the ℓ_2-norm based on the empirical analysis.

Different from SFP [6] that the pruning rate equals the goal pruning rate P_i^{goal} during retraining, we use different pruning rate $P_i^{'}$ at every epoch. The definition of $P_i^{'}$ is list as follows:

$$P_i^{'} = \mathcal{H}(P_i^{goal}, D, P_i^{min}, epoch), \tag{4.6}$$

where P_i^{goal} represents the goal pruning rate for the i_{th} layer, D and P_i^{min} are pre-defined parameter which will be explicitly explained later. As exponential parameter decay is widely used in optimization [125] to achieve a stable result, we change the pruning rate exponentially. The equation is as follows:

$$P_i^{'} = a \times e^{-k \times epoch} + b, \tag{4.7}$$

In order to solve three parameters a, k, b of the above exponential equation, three points consists of (epoch, $P_i^{'}$) pair are needed. Certainly, the first point is (0, P_i^{min}),

which means the pruning rate is P_i^{min} for the first training epoch. In addition, to achieve the goal pruning rate P_i at the final retraining epoch, the point ($epoch_{max}$, P_i^{goal}) is essential. Now we have to define the third point ($epoch_{max} \times D$, $3P_i^{goal}/4$), to solve the equation. This means that when the epoch number is $epoch_{max} \times D$, the pruning rate increase to 3/4 of the goal pruning rate P_i^{goal}.

4.3.4.2 Filter Pruning

We set the value of selected $N_{i+1}P_i^{'}$ filters to zero (see the filter pruning step in Figure 4.3). This can temporarily eliminate their contribution to the network output. We prune *all* the weighted layers at the same time. In this way, we can prune each filter in parallel, which would cost negligible computation time. In contrast, previous methods [14, 19] always conduct a greedy layer by layer pruning and retraining, which would cost more computation time, especially when the model depth increases. Moreover, we use the *same* pruning rate for *all* the weighted layers, $P_i^{goal} = P$. Therefore, we only require one hyper-parameter P to balance acceleration and accuracy. This can avoid the inconvenient hyper-parameter search or the complicated sensitivity analysis shown in [1].

4.3.4.3 Reconstruction

After the pruning step, we train the network for one epoch to reconstruct the pruned filters. In order to keep the representative capacity and the high performance of the model, those pruned filters are updated to non-zero by back-propagation, as shown in Figure 4.3. In this way, the pruned model has the same capacity as the original model during the training. In contrast, hard filter pruning leads to a decrease of feature maps, so the model capacity is reduced and the performance is influenced. With large model capacity, we could integrate the pruning step into the normal training schema, that is, training and pruning the model synchronously.

4.3.4.4 Obtaining Compact Model

ASFP iterates over the filter selection, filter pruning, and reconstruction steps. After model convergence, we can obtain a sparse model containing many "zero filters". The features maps corresponding to those "zero filters" will always be zero during the inference procedure. Therefore, there will be no influence to remove these filters as well as the corresponding feature maps.

4.3.5 Pruning Strategy for Convolutional Network

Pruning the traditional convolutional architectures [116, 118] is easy to understand, but it is elusive for pruning recent structural variants, such as ResNet [61]. In Figure 4.4, pruning the residual blocks is illustrated. If we set the pruning rate as 50%, the output dimension of three layers would decrease by 50% (the red numbers). The input dimension of the last two layers would change according to the output dimension of the previous layer (the green numbers). What we should especially care about is the element-wise additive of the residual connections and the output of convolutional layers. In Figure 4.4, the channel number of the residual connections and the output of convolutional layers is 256 and 128, respectively. The element-wise additive should change to:

$$\mathbf{O}_{index} = \begin{cases} \mathbf{R}_{index} + \mathbf{C}_{index} & if \quad index \in \mathbf{I} \\ \mathbf{R}_{index} & else \end{cases} \tag{4.8}$$

where $\mathbf{I}$ is the index of 128 remaining channels, $\mathbf{C}$ is the convolutional output after the batch norm layer (128 channels), $\mathbf{R}$ is the residual output (256 channels), and $\mathbf{O}$ is the output of the whole residual block.

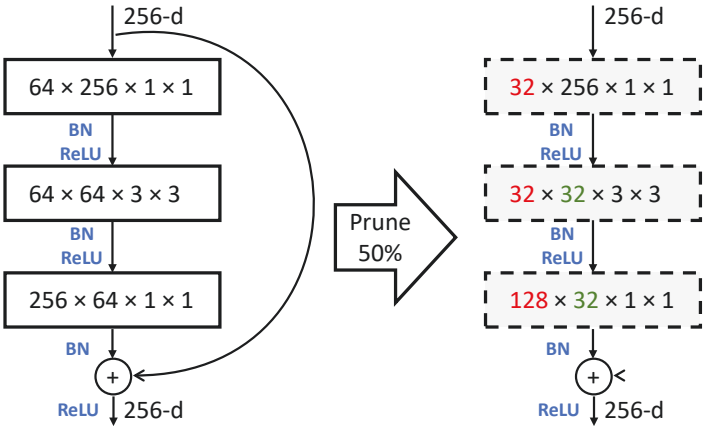


Figure 4.4 : Pruning residual block with pruning rate 50%. Red and green number means the remaining output channel number and input channel number after pruning, respectively. "BN" and "ReLU" represents the batch norm layer and non-linear layer, respectively.

4.3.6 Computation Complexity Analysis

4.3.6.1 Theoretical Speedup Analysis

Suppose the filter pruning rate of the i_{th} layer is P_i, which means the $N_{i+1} \times P_i$ filters are set to zero and pruned from the layer, and the other $N_{i+1} \times (1 - P_i)$ filters remain unchanged. Besides, suppose the size of the input and output feature map of i_{th} layer is $H_i \times W_i$ and $H_{i+1} \times W_{i+1}$. Then after filter pruning, the dimension of useful output feature map of the i_{th} layer decreases from $N_{i+1} \times H_{i+1} \times W_{i+1}$ to $N_{i+1}(1 - P_i) \times H_{i+1} \times W_{i+1}$. Note that the output of i_{th} layer is the input of $(i + 1)_{th}$ layer. And we further prunes the $(i + 1)_{th}$ layer with a filter pruning rate P_{i+1}, then the calculation of $(i + 1)_{th}$ layer is decrease from $N_{i+2} \times N_{i+1} \times k^2 \times H_{i+2} \times W_{i+2}$ to $N_{i+2}(1 - P_{i+1}) \times N_{i+1}(1 - P_i) \times k^2 \times H_{i+2} \times W_{i+2}$. In other words, a proportion of $1 - (1 - P_{i+1}) \times (1 - P_i)$ of the original calculation is reduced, which will make the inference procedure of CNN models much faster.

4.3.6.2 Realistic Speedup Analysis

In theoretical speedup analysis, other operations such as batch normalization and pooling are negligible compared to the convolution operations. Therefore, we consider the FLOPs (Floating-point operations per second) of convolution operations for computation complexity comparison, which is commonly used in the previous work [1, 19]. In the real scenario, reduced FLOPs cannot bring the same level of realistic speedup because non-tensor layers (*e.g.*, batch normalization and pooling layers) also need the inference time on GPU [19]. In addition, the limitation of IO delay, buffer switch, and efficiency of BLAS libraries also lead to the wide gap between theoretical and realistic speedup ratio. We compare the theoretical and realistic speedup in Section 4.4.4.

4.4 Experiment

4.4.1 Benchmark Datasets and Experimental Setting

Dataset. Our method is evaluated on two benchmarks: CIFAR-10 [70] and ILSVRC-2012 [105]. CIFAR-10 contains 50,000 training images and 10,000 test images, which are categorized into ten classes. ILSVRC-2012 is a large-scale dataset containing 1.28 million training images and 50k validation images of 1,000 classes.

Architecture. As discussed in [14, 19, 104], multiple-branch ResNet [61] is less redundant than VGGNet [118], so it is more difficult to accelerate ResNet. Therefore, we focus on pruning the challenging ResNet model. To validate our method on the single-branch network, we also prune the VGGNet following [1].

Training setting. In the CIFAR-10 experiments, we use the default parameter setting in [113] and follow the training schedule in [114]. For CIFAR-10 dataset, we test our ASFP on ResNet-56 and 110. On ILSVRC-2012, we follow the same parameter settings as [61, 113]. The data argumentation strategies are the same as

Table 4.1 : Overall performance of pruning ResNet on CIFAR-10.

Depth	Method	Pre-train?	Baseline Accu. (%)	Accelerated Accu. (%)	Accu. Drop (%)	FLOPs	Pruned FLOPs(%)
56	PFEC [1]	✗	93.04	91.31	1.75	9.09E7	27.6
	CP [14]	✗	92.80	90.90	1.90	-	50.0
	SFP [6]	✗	**93.59** (±0.58)	92.26 (±0.31)	1.33	**5.94E7**	**52.6**
	ASFP (40%)	✗	**93.59** (±0.58)	92.44 (±0.07)	**1.15**	**5.94E7**	**52.6**
	PFEC [1]	✓	93.04	93.06	**-0.02**	9.09E7	27.6
	CP [14]	✓	92.80	91.80	1.00	-	50.0
	SFP [6]	✓	**93.59** (±0.58)	**93.35** (±0.31)	0.24	**5.94E7**	**52.6**
	ASFP (40%)	✓	**93.59** (±0.58)	93.12 (±0.20)	0.47	**5.94E7**	**52.6**
110	PFEC [1]	✗	93.53	92.94	0.61	1.55E8	38.6
	MIL [104]	✗	93.63	93.44	0.19	-	34.2
	SFP [6]	✗	**93.68** (±0.32)	92.62 (±0.60)	1.04	**1.21E8**	**52.3**
	ASFP (20%)	✗	**93.68** (±0.32)	**93.94** (±0.56)	**-0.24**	1.82E8	28.2
	ASFP (40%)	✗	**93.68** (±0.32)	93.20 (±0.10)	0.48	**1.21E8**	**52.3**
	PFEC [1]	✓	93.53	93.30	0.20	1.55E8	38.6
	SFP [6]	✓	**93.68** (±0.32)	**93.86** (±0.21)	**-0.18**	1.50E8	40.8
	SFP [6]	✓	**93.68** (±0.32)	92.90 (±0.18)	0.78	**1.21E8**	**52.3**
	ASFP (30%)	✓	**93.68** (±0.32)	93.37 (±0.12)	0.31	1.50E8	40.8
	ASFP (40%)	✓	**93.68** (±0.32)	93.10 (±0.06)	0.58	**1.21E8**	**52.3**

PyTorch implementation [115]. We test ASFP on ResNet-18, 34, 50 and we use the pruning rate 30% for all the models. We also analyze the difference between pruning the pre-trained model and scratch model. For pruning the model from scratch, We use the normal training schedule without additional fine-tuning process. For pruning the pre-trained model, we reduce the learning rate to one-tenth of the original learning rate. To conduct a fair comparison of pruning from scratch and pre-trained models, we use the same training epochs to train/fine-tune the network. The previous work [1] uses fewer epochs to fine-tune the pruned model, but it converges too early and harms the accuracy, as shown in section 4.4.2.

Pruning setting. For VGGNet on CIFAR-10, we use the same pruning rate as [1]. For experiments on ResNet, we follow [6] and prune *all* the convolutional

layers with the *same* pruning rate at the same time. We do not prune the projection shortcuts for simplification, which only need negligible time and do not affect the overall cost. Therefore, only one hyper-parameter, the pruning rate $P_i = P$ is used to balance acceleration and accuracy.

To asymptotically change the pruning rate, we set the parameters in Eq. 4.6 as $D = 1/8$ and $P_i^{min} = 0$. This setting is denoted as ASFP-P0.[†] For example, if we use the goal pruning rate $P_i^{goal} = 30\%$, then the pruning rate curve according to epoch is shown in Figure 4.5. If we set $P_i^{min} = P_i^{goal}$, ASFP is same as SFP. The pruning operation is conducted at the end of every training epoch. We run some experiments three times and report the "mean ± std". The performance is compared with other state-of-the-art acceleration algorithms, *e.g.*, MIL [104], PFEC [1], CP [14], ThiNet [19], SFP [6], NISP [17]. We choose to directly cite the numbers from original papers for a fair comparison.

Explanation for Tables. The results of ResNet are listed in Table 4.1 and Table 4.3. In "Pre-train?" column, "Y" and "N" indicate whether to use the pre-trained model as initialization or not. The "Accu. Drop" is the accuracy of the pruned model minus that of the baseline model, so negative number means the accelerated model has higher accuracy than the baseline model. A smaller number of "Accu. Drop" is better. For CIFAR-10, we run every experiment for three times to get the mean and standard deviation of the accuracy. For Imagenet, we just list the one-view accuracy.

Explanation for Baselines. The baseline network is the same for different pruning methods, and accuracy numbers are cited from the original paper. The different accuracies are due to different hyper-parameter settings (*e.g.* different data

[†]For the following sections, the "ASFP" (without suffix) means "ASFP-P0" if not particularly indicated.

augmentations, different learning rate schedules, *etc.*) and different implementation frameworks (*e.g.* Caffe, TensorFlow and Pytorch). For example, in Thinet [19], all the images are resized into 256×256, then center-cropped to 224×224. However, in CP [14], the images are resized such that the shorter side equals to 256. In this case, we choose to use the "Accu. Drop" (in Table 4.1) rather than the "Accelerated Accu." (in Table 4.1) to fairly evaluate the effectiveness of our method.

Optimization Time of ASFP. In this paper, we care more about the acceleration during the inference time rather than the training time. However, we would like to show that the additional time cost of ASFP is negligible compared to a hard pruning method. Take the scratch model for an example. HFP and ASFP both need 200 epochs to training CIFAR-10 from scratch to converge, so the additional time cost of ASFP comes from the operation of pruning (Eq. 4.4). Two steps are needed in such a process. 1) Obtaining the pruning rate $P_i^{'}$. 2) Ranking and pruning the filters. For step one, the exponent pattern of the pruning rate is pre-defined, and it could be directly accessed during training. For step two, after we get the importance scores of the filters, we zeroize the filters with smaller importance scores to conduct the pruning operation. All these steps bring minor computation cost. For HFP, it takes about 171.01s for training ResNet-110 for one epoch on GTX 1080. For ASFP, the total time for one epoch is 171.45s. Therefore, the time difference between soft pruning and hard pruning is negligible.

4.4.2 VGGNet on CIFAR-10

The result of pruning from scratch and pre-trained VGGNet is shown in Table 4.2. Not surprisingly, ASFP achieves better performance than [1] in both settings. With our pruning criterion, we could achieve slightly better accuracy than [1] when pruning the random initialized VGGNet (93.37% *vs.* 93.31%). In addition, The pruned model without fine-tuning has better performance than [1] (81.66% *vs.*

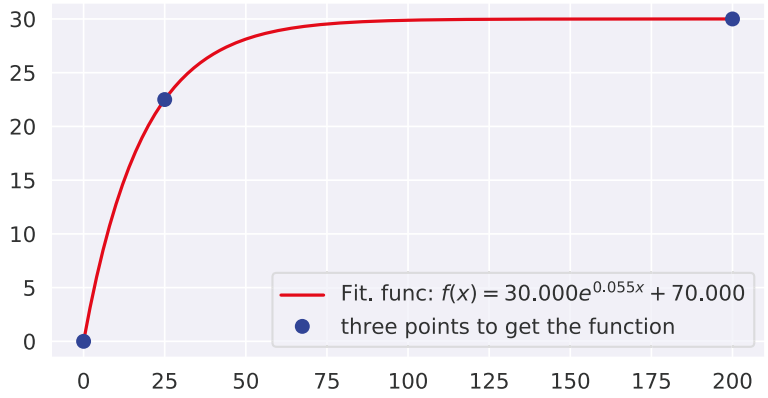


Figure 4.5 : Asymptotically changed pruning rate when the goal pruning rate is 30%. Three blue points are the three pairs to generate the exponential function of pruning rate (the solid curve).

77.45%). After fine-tuning 40 epochs, our model achieves similar accuracy with [1]. Notably, if more fine-tuning epochs (160) are utilized, the accuracy of [1] is almost unchanged (93.28% *vs.* 93.22%), which means their models have no much more capacity to learn. On the contrary, our method could achieve much better performance (94.02% *vs.* 93.28%) with more fine-tuning epochs, which shows the model capacity of our ASFP is much larger than [1].

Table 4.2 : Pruning from scratch and pre-trained VGGNet on CIFAR-10. "FT" means "fine-tuning" the pruned model.

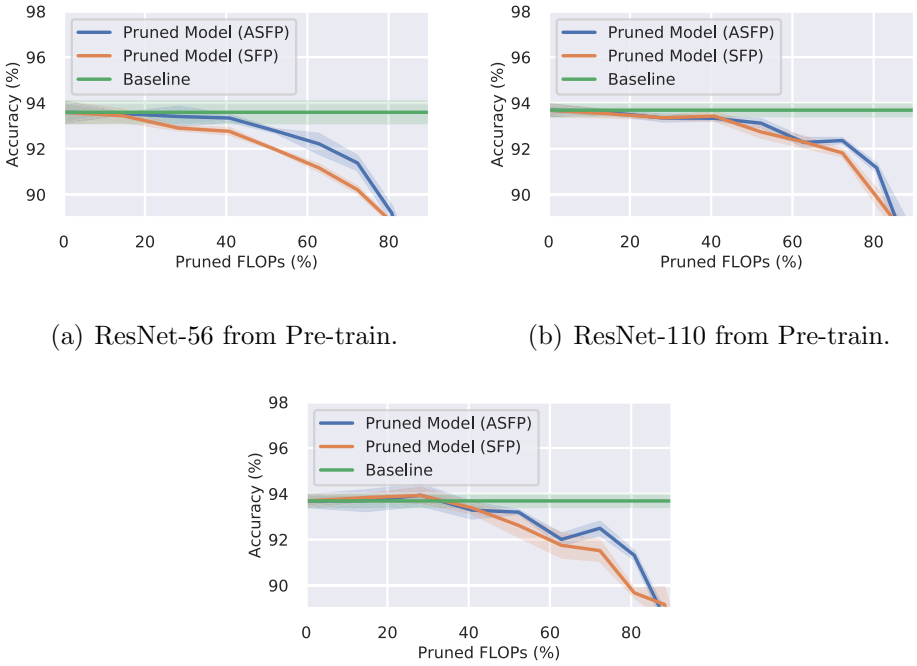
Setting \ Acc (%)	PFEC [1]	Ours
Baseline	93.58 (±0.03)	93.58 (±0.03)
Prune from scratch	93.31 (±0.03)	**93.37** (±0.08)
Prune from pre-train without FT	77.45 (±0.03)	**81.66** (±0.03)
FT 40 epochs	93.22 (±0.03)	**93.27** (±0.08)
FT 160 epochs	93.28 (±0.03)	**94.02** (±0.15)

Table 4.3 : Overall performance of pruning ResNet on ImageNet.

Depth	Method	Pre-train?	Top-1 Accu. Baseline(%)	Top-1 Accu. Accelerated(%)	Top-5 Accu. Baseline(%)	Top-5 Accu. Accelerated(%)	Top-1 Accu. Drop(%)	Top-5 Accu. Drop(%)	Pruned FLOPs(%)
18	MIL [104]	✗	69.98	66.33	89.24	86.94	3.65	2.30	34.6
	SFP [6]	✗	**70.23** (±0.06)	67.25 (±0.13)	**89.51** (±0.10)	87.76 (±0.06)	2.98	1.75	**41.8**
	ASFP (30%)	✗	**70.23** (±0.06)	**67.41**	**89.51** (±0.10)	**87.89**	**2.82**	**1.62**	**41.8**
	SFP [6]	✓	**70.23** (±0.06)	60.79	**89.51** (±0.10)	83.11	9.44	6.40	**41.8**
	ASFP (30%)	✓	**70.23** (±0.06)	**68.02**	**89.51** (±0.10)	**88.19**	**2.21**	**1.32**	**41.8**
34	SFP [6]	✗	**73.92**	**71.83**	**91.62**	90.33	**2.09**	1.29	**41.1**
	ASFP (30%)	✗	**73.92**	71.72	**91.62**	**90.65**	2.20	**0.97**	**41.1**
	PFEC [1]	✓	**73.23**	72.17	-	-	**1.06**	-	24.2
	SFP [6]	✓	**73.92**	72.29	**91.62**	90.90	1.63	0.72	**41.1**
	ASFP (30%)	✓	**73.92**	**72.53**	**91.62**	**91.04**	1.39	**0.58**	**41.1**
50	SFP [6]	✗	**76.15**	74.61	**92.87**	92.06	1.54	0.81	**41.8**
	ASFP (30%)	✗	**76.15**	**74.88**	**92.87**	**92.39**	**1.27**	**0.48**	**41.8**
	CP [14]	✓	-	-	92.20	90.80	-	1.40	**50.0**
	ThiNet [19]	✓	72.88	72.04	91.14	90.67	0.84	0.47	36.7
	NISP [17]	✓	-	-	-	-	-	0.89	44.0
	SFP [6]	✓	**76.15**	62.14	**92.87**	84.60	14.01	8.27	41.8
	ASFP (30%)	✓	**76.15**	**75.53**	**92.87**	**92.73**	**0.62**	**0.14**	41.8

4.4.3 ResNet on CIFAR-10

Table 4.1 shows the results on CIFAR-10. Our ASFP could achieve a better performance than other state-of-the-art hard filter pruning methods. For example, PFEC [1] accelerate ResNet-110 by 38.6% speedup ratio with 0.61% accuracy drop when pruning the scratch models. In contrast, our ASFP can accelerate ResNet-110 to 52.3% speed-up with only 0.48% accuracy drop. When pruning the pre-trained ResNet-110, the accuracy drop of our ASFP is smaller than PFEC [1] when pruning the same number of ratio. When pruning the scratch ResNet-56, we can achieve more acceleration ratio than CP [14] (52.6% *vs.* 50.0%) with less accuracy drop (1.15% *vs.* 1.90%) Notably, we can even improve 0.24% accuracy when pruning 28.2% FLOPs of scratch ResNet-56.



(a) ResNet-56 from Pre-train.

(b) ResNet-110 from Pre-train.

(c) ResNet-110 from Scratch.

Figure 4.6 : Model performance regarding different ratio of pruned FLOPs. The green line indicates the model without pruning. The blue and the orange lines represent the model under ASFP and SFP, respectively.

When the pruning rate is small, we find the performance of SFP [6] and ASFP is competitive. But ASFP outperforms SFP when a large portion of FLOPs are pruned. The comprehensive comparison is shown in Figure 4.6. This is because ASFP is suitable for the situation when a large quantity of the information is removed by pruning. These results validate the effectiveness of our ASFP algorithm, which can produce a more compressed model with comparable performance to the original model.

4.4.4 ResNet on ILSVRC-2012

Result Explanation. Table 4.3 shows that ASFP outperforms other state-of-the-art methods. For pruning from a random initialized ResNet-18, our ASFP

Table 4.4 : Comparison of the theoretical and realistic speedup. We only count the time consumption of the forward procedure.

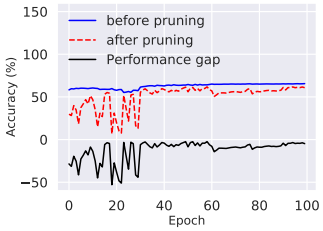
Model	Baseline time (ms)	Pruned time (ms)	Realistic Speed-up(%)	Theoretical Speed-up(%)
ResNet-18	37.10	26.97	27.4	41.8
ResNet-34	63.97	45.14	29.4	41.1
ResNet-50	135.01	94.66	29.8	41.8

achieves more inference speedup than MIL [104] (41.8% v.s. 34.6%), but the top-5 accuracy drop of our pruned model is less than that of their model (1.62% v.s. 2.30%). For pruning pre-trained ResNet-34, our ASFP achieves a much better acceleration than PFEC [1] (41.1% v.s. 24.2%) with comparable accuracy drop. For pre-trained ResNet-50, SFP leads to 8.27% top-5 accuracy drop for 41.8% speedup, but our ASFP could achieve negligible top-5 accuracy drop (0.14%) with the same speedup ratio. The maintained model capacity and asymptotic pruning of ASFP are the main reasons for the improved accuracy and efficiency.

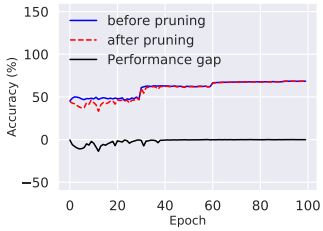
Realistic Acceleration. In order to test the realistic speedup ratio, we measure the forward time of the pruned models on one GTX 1080 Ti GPU with a batch size of 64. The results are shown in Table 4.4. The gap between theoretical and realistic speed may come from non-tensor layers and the limitation of IO delay, buffer switch and the efficiency of BLAS libraries [104].

4.4.5 Comparing SFP and ASFP

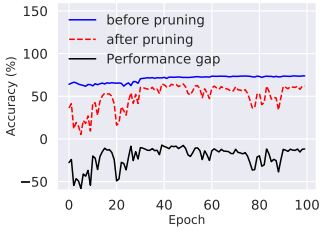
Performance Regrading Ratio of Pruned FLOPs. In Figure 4.6, we test the accuracy of ResNet-56 and ResNet-110 under different ratios of pruned FLOPs. For pruning pre-trained initialization, as shown in Figure 4.6(a) and Figure 4.6(b), ASFP could obtain better performance than SFP on almost all ratio of pruned



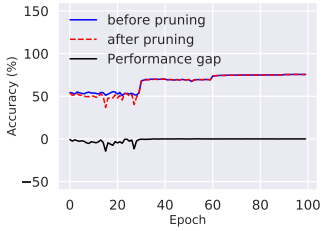
(a) SFP on ResNet-18



(b) ASFP on ResNet-18

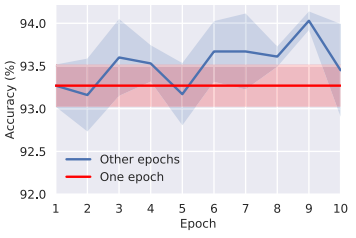


(c) SFP on ResNet-50

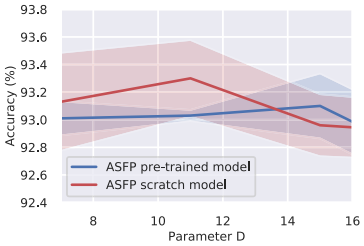


(d) ASFP on ResNet-50

Figure 4.7 : The training process of ResNet-18 and ResNet-50 and on ImageNet regarding SFP and ASFP. The solid blue line and red dashed line indicate the accuracy of the model before and after pruning, respectively. The black line is the performance gap due to pruning, which is calculated by the accuracy after pruning subtracting that before pruning.



(a) Different pruning intervals.



(b) Different parameter D in the Eq. 4.6.

Figure 4.8 : Ablation study of ASFP. (Solid line and shadow denote the mean and standard deviation of three experiments, respectively.)

FLOPs. Even for pruning models with the random initialization, as shown in Figure 4.6(c), our method could still outperform SFP. All the results verify that ASFP provides a more effective way to reduce the information loss and thus improves the network performance.

Stable Training Process of ASFP. The model accuracies during training for SFP and ASFP are shown in Figure 4.7. We run this comparison experiment on ResNet-18 and ResNet-50, and the pruning rate is 30%. We find that the performance gap is not stable for SFP during almost all the 100 retraining epochs. On the

contrary, the performance gap of ASFP is much more stable than that of SFP. For SFP, directly pruning a large number of filters leads to severe information loss. It is difficult for the network to recover from this, and it leads to an unstable training process. In contrast, ASFP would result in a small amount of information loss, consequently increasing the stability of the pruning process.

4.4.6 Ablation Study

Extensive ablation study is also conducted to further analyze each component of our model.

4.4.6.1 Filter Selection Criteria

The magnitude based criteria such as ℓ_p-norm are widely used to filter selection because computational resources cost is small [1]. We compare the ℓ_2-norm and ℓ_1-norm, and the results are shown in Table 4.5. We find that the performance of ℓ_2-norm criteria are slightly better than that of ℓ_1-norm criteria. The result of ℓ_2-norm is dominated by the largest element, while the result of ℓ_1-norm is also largely affected by other small elements. Therefore, filters with some large weights would be preserved by the ℓ_2-norm criteria. Consequently, the corresponding discriminative features are kept so the accuracy of the pruned model is better.

Table 4.5 : Accuracy of CIFAR-10 on ResNet-110 under different pruning rate with different filter selection criteria.

Pruning rate(%)	10	20	30
ℓ_1-norm	93.68 ± 0.60	93.68 ± 0.76	93.34 ± 0.12
ℓ_2-norm	**93.89 ± 0.19**	**93.93 ± 0.41**	**93.38 ± 0.30**

4.4.6.2 Varying Pruned FLOPs

We evaluate the accuracy of different pruned FLOPs for ResNet-110, and show the results in Figure 4.6. When the ratio of pruned FLOPs is less than 40%, ASFP and SFP achieve similar accuracy. However, when the ratio of pruned FLOPs is more than 40%, ASFP could obtain much better performance than SFP. This is because pruning a large number of filters leads to severe information lose and ASFP is especially effective for this case. In contrast, when only a small portion of the information is lost, the maintained model capacity of SFP is enough for good results. For the pruning rate between 0% and about 23%, the accuracy of the accelerated model is higher than the baseline model. This shows that our ASFP and SFP both have a regularization effect on the neural network.

4.4.6.3 Selection of the Pruned Layers

Previous work always prunes a portion of the layers of the network. Besides, different layers always have different pruning rates. For example, [1] only prunes insensitive layers, [19] skips the last layer of every block of the ResNet, and [19] prunes more aggressively for shallower layers and prune less for deep layers.

Similarly, we compare the performance of pruning the first and second layer of all basic blocks of ResNet-110. We set the pruning rate as 30%. The model with all the first layers of blocks pruned has an accuracy of $93.96 \pm 0.13\%$, while the model with all the second layers of blocks pruned has an accuracy of $93.38 \pm 0.44\%$. If we carefully select the pruned layers based on the sensitivity, performance improvement may be potentially obtained. However, tuning these hyper-parameters is not the focus of this manuscript.

4.4.6.4 Sensitivity of the ASFP Interval

By default, we conduct our ASFP operation at the end of every training epoch, we call the ASFP interval equals one under this setting. However, different ASFP intervals may lead to a different performance, so we explore the sensitivity of ASFP interval. We use ResNet-110 under a pruning rate of 30% as a baseline, and change the ASFP interval from one epoch to ten epochs. The result is shown in Figure 4.8(a). We find the model accuracy of most (80%) intervals surpasses the accuracy of one epoch interval. Therefore, we can even achieve better performance if we fine-tune this parameter.

4.4.6.5 Sensitivity of Parameter D of ASFP

We change the parameter D in the Eq. 4.6 to comprehensively understand ASFP, and the results are shown in the Figure 4.8(b). We prune scratch and pre-trained ResNet-56 on CIFAR-10 and set the pruning rate as 40%. When changing the parameter D from 7 to 16, we find the model accuracy has no large fluctuation ($< 0.3\%$). This shows that the final result of pruning is not sensitive to the parameter D.

4.5 Conclusion

In this paper, we propose an asymptotic soft filter pruning approach (ASFP) to accelerate the deep CNNs. As the training procedures go, we allow the pruned filters to be updated and asymptotically adjust the pruning rate. The soft manner could maintain the model capacity, and the asymptotic pruning could make the pruning process more stable. Therefore, our ASFP could achieve superior performance. Remarkably, without using the pre-trained model, our ASFP can achieve competitive performance compared to the state-of-the-art approaches. Moreover, by leveraging the pre-trained model, our ASFP achieves better results.

Chapter 5

Filter Pruning via Geometric Median

5.1 Introduction

The deeper and wider architectures of deep CNNs bring about the superior performance of computer vision tasks. However, they also cause the prohibitively expensive computational cost and make the model deployment on mobile devices hard if not impossible. Even the latest architecture with high efficiencies, such as residual connection [61] or inception module [103], has millions of parameters requiring billions of float point operations (FLOPs) [6]. Therefore, it is necessary to attain the deep CNN models which have relatively low computational cost but high accuracy.

Recent developments on pruning can be divided into two categories, *i.e.*, weight pruning [4, 126] and filter pruning [1, 17]. Weight pruning directly deletes weight values in a filter which may cause unstructured sparsities. This irregular structure makes it difficult to leverage the high-efficiency Basic Linear Algebra Subprograms (BLAS) libraries [19]. In contrast, filter pruning directly discards the whole selected filters and leaves a model with regular structures. Therefore, filter pruning is more preferred for accelerating the networks and decreasing the model size.

Current practice [1,6,20] performs filter pruning by following the "smaller-norm-less-important" criterion, which believes that filters with smaller norms can be pruned safely due to their less importance. As shown in the top right of Figure 5.1(a), after calculating norms of filters in a model, a pre-specified threshold $\mathcal{T}$ is utilized to select filters whose norms are smaller than it.

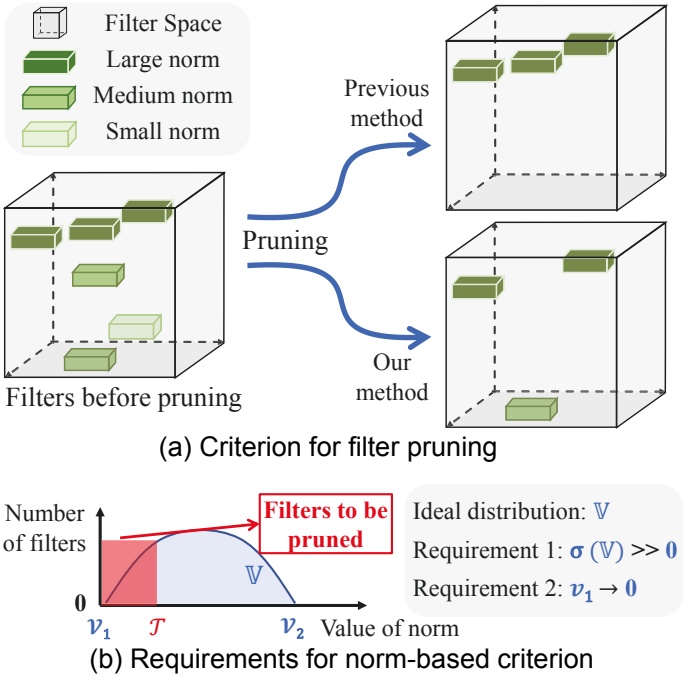


Figure 5.1 : An illustration of (a) the pruning criterion for norm-based approach and the proposed method; (b) requirements for norm-based filter pruning criterion. In (a), the green boxes denote the filters of the network, where deeper color denotes larger norm of the filter. For the norm-based criterion, only the filters with the largest norm are kept based on the assumption that smaller-norm filters are less important. In contrast, the proposed method prunes the filters with redundant information in the network. In this way, filters with different norms indicated by different intensities of green may be retained. In (b), the blue curve represents the ideal norm distribution of the network, and the v_1 and v_2 is the minimal and maximum value of norm distribution, respectively. To choose the appropriate threshold $\mathcal{T}$ (the red shadow), two requirements should be achieved, that is, the norm deviation should be large, and the minimum of the norm should be arbitrarily small.

However, as illustrated in Figure 5.1(b), there are two prerequisites to utilize this "smaller-norm-less-important" criterion. First, the deviation of filter norms should be significant. This requirement makes the searching space for threshold $\mathcal{T}$ wide

enough so that separating those filters needed to be pruned would be an easy task. Second, the norms of those filters which can be pruned should be arbitrarily small, *i.e.*, close to zero; in other words, the filters with smaller norms are expected to make absolutely small contributions, rather than relatively less but positively large contributions, to the network. An ideal norm distribution when satisfactorily meeting those two requirements is illustrated as the blue curve in Figure 5.1. Unfortunately, based on our analysis and experimental observations, this is not always true.

To address the problems mentioned above, we propose a novel filter pruning approach, named Filter Pruning via Geometric Median (FPGM). Different from the previous methods which prune filters with ***relatively less contribution***, FPGM chooses the filters with ***the most replaceable contribution***. Specifically, we calculate the Geometric Median (GM) [127] of the filters within the same layer. According to the characteristics of GM, the filter(s) $\mathcal{F}$ near it can be represented by the remaining ones. Therefore, pruning those filters will not have substantial negative influences on model performance. Note that FPGM does not utilize norm based criterion to select filters to prune, which means its performance will not deteriorate even when failing to meet requirements for norm-based criterion.

Contributions. We have three contributions:

(1) We analyze the norm-based criterion utilized in previous works, which prunes the relatively less important filters. We elaborate on its two underlying requirements which lead to its limitations;

(2) We propose FPGM to prune the most replaceable filters containing redundant information, which can still achieve good performances when norm-based criterion fails;

(3) The extensive experiment on two benchmarks demonstrates the effectiveness and efficiency of FPGM.

5.2 Related Works

Most previous works on accelerating CNNs can be roughly divided into four categories, namely, *matrix decomposition* [56,58], *low-precision weights* [62,63,128], knowledge distilling [129, 130] and *pruning*. *Pruning*-based approaches aim to remove the unnecessary connections of the neural network [1, 4, 131]. Essentially, ***weight pruning*** always results in unstructured models, which makes it hard to deploy the efficient BLAS library, while ***filter pruning*** not only reduces the storage usage on devices but also decreases computation cost to accelerate the inference. We could roughly divide the filter pruning methods into two categories by whether the training data is utilized to determine the pruned filters, that is, *data dependent* and *data independent* filter pruning. Data independent method is more efficient than data dependent method as the utilizing of training data is computation consuming.

Weight Pruning. Many recent works [4–6, 66, 126, 132–134] focus on pruning fine-grained weight of filters. For example, [4] proposes an iterative method to discard the small weights whose values are below the predefined threshold. [126] formulates pruning as an optimization problem of finding the weights that minimize the loss while satisfying a pruning cost condition.

Data Dependent Filter Pruning. Some filter pruning approaches [8, 14–17, 19, 21–23, 71–73] need to utilize training data to determine the pruned filters. [19] adopts the statistics information from the next layer to guide the filter selections. [71] aims to obtain a decomposition by minimizing the reconstruction error of training set sample activation. [8] proposes an inherently data-driven method which use Principal Component Analysis (PCA) to specify the proportion of the energy that should be preserved. [73] applies subspace clustering to feature maps to eliminate the redundancy in convolutional filters.

Data Independent Filter Pruning. Concurrently with our work, some data

independent filter pruning strategies [1, 6, 9, 20] have been explored. [1] utilizes an ℓ_1-norm criterion to prune unimportant filters. [6] proposes to select filters with a ℓ_2-norm criterion and prune those selected filters in a soft manner. [20] proposes to prune models by enforcing sparsity on the scaling parameter of batch normalization layers. [9] uses spectral clustering on filters to select unimportant ones.

Discussion. To the best of our knowledge, only one previous work reconsiders the smaller-norm-less-important criterion [20]. We would like to highlight our advantages compared to this approach as below: (1) [20] pays more attention to enforcing sparsity on the scaling parameter in the batch normalization operator, which is not friendly to the structure without batch normalization. On the contrary, our approach is not limited by this constraint. (2) After pruning channels selected, [20] need fine-tuning to reduce performance degradation. However, our method combines the pruning operation with normal training procedure. Thus extra fine-tuning is not necessary. (3) Calculation of the gradient of scaling factor is needed for [20]; therefore lots of computation cost are inevitable, whereas our approach could accelerate the neural network without calculating the gradient of scaling factor.

5.3 Methodology

5.3.1 Preliminaries

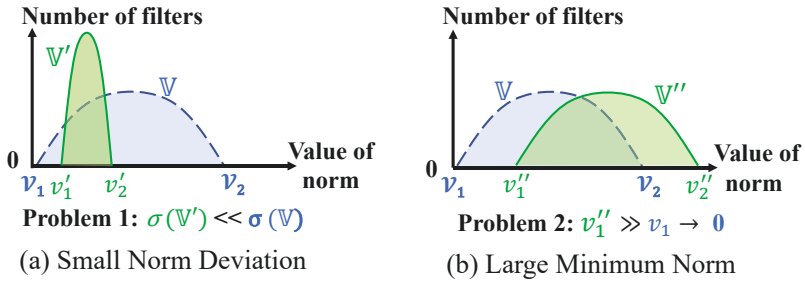
We formally introduce symbols and notations in this subsection. We assume that a neural network has L layers. We use N_i and N_{i+1}, to represent the number of input channels and the output channels for the i_{th} convolution layer, respectively. $\mathcal{F}_{i,j}$ represents the j_{th} filter of the i_{th} layer, then the dimension of filter $\mathcal{F}_{i,j}$ is $\mathbb{R}^{N_i \times K \times K}$, where K is the kernel size of the network*. The i_{th} layer of the network $\mathbf{W}^{(i)}$ could be represented by $\{\mathcal{F}_{i,j}, 1 \leq j \leq N_{i+1}\}$. The tensor of connection of the

*Fully-connected layers equal to convolutional layers with $k = 1$

deep CNN network could be parameterized by $\{\mathbf{W}^{(i)} \in \mathbb{R}^{N_{i+1} \times N_i \times K \times K}, 1 \leq i \leq L\}$.

5.3.2 Analysis of Norm-based Criterion

Figure 5.1 gives an illustration for the two requirements for successful utilization of the norm-based criterion. However, these requirements may not always hold, and it might lead to unexpected results. The details are illustrated in Figure 5.2, in which the blue dashed curve and the green solid curve indicates the norm distribution in ideal and real cases, respectively.



(a) Small Norm Deviation (b) Large Minimum Norm

Figure 5.2 : Ideal and Reality of the norm-based criterion: (a) Small Norm Deviation and (b) Large Minimum Norm. The blue dashed curve indicates the ideal norm distribution, and the green solid curve denotes the norm distribution might occur in real cases.

(1) *Small Norm Deviation.* The deviation of filter norm distributions might be too small, which means the norm values are concentrated to a small interval, as shown in Figure 5.2(a). A small norm deviation leads to a small search space, which makes it difficult to find an appropriate threshold to select filters to prune.

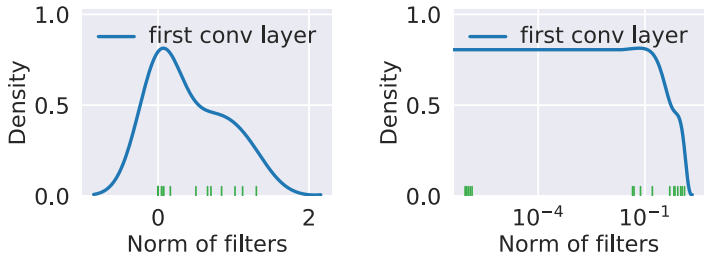
(2) *Large Minimum Norm.* The filters with the minimum norm may not be arbitrarily small, as shown in the Figure 5.2(b), $v_1'' >> v_1 \rightarrow 0$. Under this condition, those filters considered as the least important still contribute significantly to the network, which means every filter is highly informative. Therefore, pruning those filters with minimum norm values will cast a negative effect on the network.

5.3.3 Norm Statistics in Real Scenarios

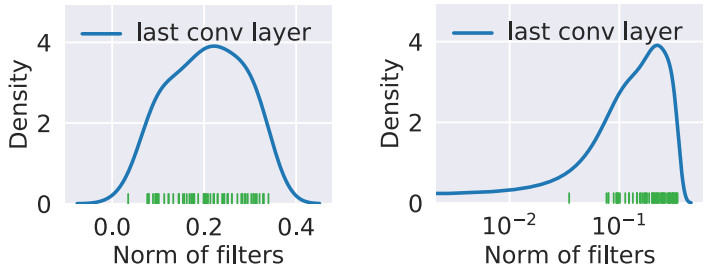
In Figure 5.3, statistical information collected from pre-trained ResNet-110 on CIFAR-10 and pre-trained ResNet-18 on ILSVRC-2012 demonstrates previous analysis. The small green vertical lines show each observation in this norm distribution, and the blue curves denote the Kernel Distribution Estimate (KDE) [135], which is a non-parametric way to estimate the probability density function of a random variable. The norm distribution of first layer and last layer in both structures are drawn. In addition, to clearly illustrate the relation between norm points, two different x-scale, *i.e.*, linear x-scale and log x-scale, are presented.

(1) *Small Norm Deviation in Network.* For the first convolutional layer of ResNet-110, as shown in Figure 5.3(b), there is a large quantity of filters whose norms are **concentrated** around the magnitude of 10^{-6}. For the last convolutional layer of ResNet-110, as shown in Figure 5.3(c), the interval span of the value of norm is roughly **0.3**, which is much smaller than the interval span of the norm of the first layer (**1.7**). For the last convolutional layer of ResNet-18, as shown in Figure 5.3(g), most filter norms are between the interval $[0.8, 1.0]$. In all these cases, filters are distributed too densely, which makes it difficult to select a proper threshold to distinguish the important filters from the others.

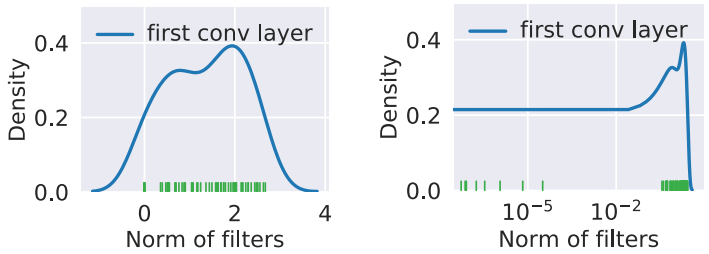
(2) *Large Minimum Norm in Network.* For the last convolutional layer of ResNet-18, as shown in Figure 5.3(g), the minimum norm of these filters is around **0.8**, which is **large** comparing to filters in the first convolutional layer (Figure 5.3(e)). For the last convolutional layer of ResNet-110, as shown in Figure 5.3(c), only one filter is arbitrarily small, while the others are not. Under those circumstances, the filters with minimum norms, although they are relatively less important according to the norm-based criterion, still make significant contributions in the network.



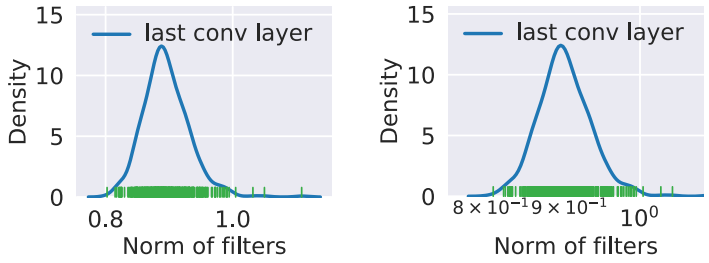
(a) ResNet-110 (linear x-scale) (b) ResNet-110 (log x-scale)



(c) ResNet-110 (linear x-scale) (d) ResNet-110 (log x-scale)



(e) ResNet-18 (linear x-scale) (f) ResNet-18 (log x-scale)



(g) ResNet-18 (linear x-scale) (h) ResNet-18 (log x-scale)

Figure 5.3 : Norm distribution of filters from different layers of ResNet-110 on CIFAR-10 and ResNet-18 on ILSVRC-2012. The small green vertical lines and blue curves denote each norm and Kernel Distribution Estimate (KDE) of the norm distribution, respectively.

5.3.4 Geometric Median

The central idea of geometric median [127] is as follows: given a set of n points $a^{(1)}, \ldots, a^{(n)}$ with each $a^{(i)} \in \mathbb{R}^d$, find a point $x^* \in \mathbb{R}^d$ that minimizes the sum of Euclidean distances to them:

$$x^* = \arg\min_{x \in \mathbb{R}^d} f(x) \quad \text{where} \quad f(x) \stackrel{\text{def}}{=} \sum_{i \in [1,n]} \|x - a^{(i)}\|_2 \tag{5.1}$$

where $[1, n] = \{1, ..., n\}$.

As the geometric median is a classic robust estimator of centrality for data in Euclidean spaces [127], we can use the geometric median to get the common information of all the filters within the single layer.

Geometric median is a non-trivial problem in computational geometry, the previous fastest running times for computing a $(1 + \epsilon)$-approximate geometric median were $\widetilde{O}(dn^{4/3} \cdot \epsilon^{-8/3})$ by [136], $O(nd \log^3(n/\epsilon))$ by [137], and this is time-consuming. We also introduct how to reduce the computational cost in the following section.

5.3.5 Filter Pruning via Geometric Median

To get rid of the constraints in the norm-based criterion, we propose a new filter pruning method inspired from geometric median.

$$x^{GM} = \arg\min_{x \in \mathbb{R}^{N_i \times K \times K}} \sum_{j' \in [1, N_{i+1}]} \|x - \mathcal{F}_{i,j'}\|_2, \tag{5.2}$$

In the i_{th} layer, find the filter(s) nearest to the geometric median in that layer:

$$\mathcal{F}_{i,j^*} = \arg\min_{\mathcal{F}_{i,j'}} \|\mathcal{F}_{i,j'} - x^{GM}\|_2, \text{ s.t. } j' \in [1, N_{i+1}], \tag{5.3}$$

then $\mathcal{F}_{i,j^*}$ can be represented by the other filters in the same layer, and therefore, pruning them has little negative impacts on the network performance.

In our case, as the final result is in a list of known points, that is, the candidate filters in i_{th} layer. We could instead find which filter minimizes the summation of the distance with other filters:

$$\begin{aligned}\mathcal{F}_{i,x^*} &= \arg\min_{x} \sum_{j' \in [1,N_{i+1}]} \|x - \mathcal{F}_{i,j'}\|_2, \text{ s.t. } x \in \{\mathcal{F}_{i,1}, ..., \mathcal{F}_{i,N_{i+1}}\} \\ &\overset{\text{def}}{=} \arg\min_{x} g(x), \text{ s.t. } x \in \{\mathcal{F}_{i,1}, ..., \mathcal{F}_{i,N_{i+1}}\}\end{aligned} \tag{5.4}$$

Note that even if $\mathcal{F}_{i,x^*}$ is not included in the calculation of the geometric median in Equation.5.4[†], we could also achieve the same result.

In this setting, we want to find the filter

$$\mathcal{F}_{i,x^{*\prime}} = \arg\min_{x} g'(x), \text{ s.t. } x \in \{\mathcal{F}_{i,1}, ..., \mathcal{F}_{i,N_{i+1}}\} \tag{5.5}$$

where

$$g'(x) = \sum_{j' \in [1,N_{i+1}], \mathcal{F}_{i,j'} \neq x} \|x - \mathcal{F}_{i,j'}\|_2. \tag{5.6}$$

For each $x \in \{\mathcal{F}_{i,1}, ..., \mathcal{F}_{i,N_{i+1}}\}$:

$$\begin{aligned}g(x) &= \sum_{j' \in [1,N_{i+1}]} \|x - \mathcal{F}_{i,j'}\|_2 \\ &= \sum_{j' \in [1,N_{i+1}], \mathcal{F}_{i,j'} \neq x} \|x - \mathcal{F}_{i,j'}\|_2 + [\|x - \mathcal{F}_{i,j'}\|_2]_{\mathcal{F}_{i,j'}=x} \\ &= g'(x)\end{aligned} \tag{5.7}$$

So we could get:

$$g(x) = g'(x), \ \forall \ x \in \{\mathcal{F}_{i,1}, ..., \mathcal{F}_{i,N_{i+1}}\} \tag{5.8}$$

[†]To select multiple filters, we choose several x that makes $g(x)$ to the smallest extent.

Algorithm 2 Algorithm Description of FPGM

Input: training data: $\mathbf{X}$.

Given: pruning rate P_i

Initialize: model parameter $\mathbf{W} = \{\mathbf{W}^{(i)}, 0 \leq i \leq L\}$

for $epoch = 1$; $epoch \leq epoch_{max}$; $epoch++$ **do**

    Update the model parameter $\mathbf{W}$ based on $\mathbf{X}$

    for $i = 1$; $i \leq L$; $i++$ **do**

        Find $N_{i+1}P_i$ filters that satisfy Equation 5.4

        Zeroize selected filters

    end for

end for

Obtain the compact model $\mathbf{W}^*$ from $\mathbf{W}$

Output: The compact model and its parameters $\mathbf{W}^*$

Thus, we have

$$\mathcal{F}_{i,x^*} = \underset{x \in \{\mathcal{F}_{i,1},\ldots,\mathcal{F}_{i,N_{i+1}}\}}{\arg\min} g(x) = \underset{x \in \{\mathcal{F}_{i,1},\ldots,\mathcal{F}_{i,N_{i+1}}\}}{\arg\min} g'(x) = \mathcal{F}_{i,x^{*\prime}}. \tag{5.9}$$

Since the geometric median is a classic robust estimator of centrality for data in Euclidean spaces [127], the selected filter(s), $\mathcal{F}_{i,x^*}$, and left ones share the most common information. This indicates the information of the filter(s) $\mathcal{F}_{i,x^*}$ could be replaced by others. After fine-tuning, the network could easily recover its original performance since the information of pruned filters can be represented by the remaining ones. Therefore, the filter(s) $\mathcal{F}_{i,x^*}$ could be pruned with negligible effect on the final result of the neural network. The FPGM is summarized in Algorithm 2.

5.3.6 Theoretical and Realistic Acceleration

5.3.6.1 Theoretical Acceleration

Suppose the shapes of input tensor $\mathbf{I} \in N_i \times H_i \times W_i$ and output tensor $\mathbf{O} \in N_{i+1} \times H_{i+1} \times W_{i+1}$. Set the filter pruning rate of the i_{th} layer to P_i, then $N_{i+1} \times P_i$ filters should be pruned. After filter pruning, the dimension of input and output feature map of the i_{th} layer change to $\mathbf{I}' \in [N_i \times (1 - P_i)] \times H_i \times W_i$ and $\mathbf{O}' \in [N_{i+1} \times (1 - P_i)] \times H_{i+1} \times W_{i+1}$, respectively.

If setting pruning rate for the $(i + 1)_{th}$ layer to P_{i+1}, then only $(1 - P_{i+1}) \times (1 - P_i)$ of the original computation is needed. Finally, a compact model $\{\mathbf{W}^{*(i)} \in \mathbb{R}^{N_{i+1}(1-P_i)\times N_i(1-P_{i-1})\times K\times K}\}$ is obtained.

5.3.6.2 Realistic Acceleration

In the above analysis, only the FLOPs of convolution operations for computation complexity comparison is considered, which is common in previous works [1, 6]. This is because other operations such as batch normalization (BN) and pooling are insignificant comparing to convolution operations.

However, non-tensor layers (e.g., BN and pooling layers) also need the inference time on GPU [19], and influence the realistic acceleration. Besides, the wide gap between the theoretical and realistic acceleration could also be restricted by the IO delay, buffer switch, and efficiency of BLAS libraries. We compare the theoretical and practical acceleration in Table 5.5.

Depth	Method	Fine-tune?	Baseline acc. (%)	Accelerated acc. (%)	Acc. ↓ (%)	FLOPs	FLOPs ↓(%)
20	SFP [6]	✗	**92.20** (±0.18)	90.83 (±0.31)	1.37	2.43E7	42.2
	Ours (FPGM-only 30%)	✗	**92.20** (±0.18)	91.09 (±0.10)	1.11	2.43E7	42.2
	Ours (FPGM-only 40%)	✗	**92.20** (±0.18)	90.44 (±0.20)	1.76	**1.87E7**	**54.0**
	Ours (FPGM-mix 40%)	✗	**92.20** (±0.18)	**90.62** (±0.17)	**1.58**	**1.87E7**	**54.0**
32	MIL [104]	✗	92.33	90.74	1.59	4.70E7	31.2
	SFP [6]	✗	**92.63** (±0.70)	92.08 (±0.08)	0.55	4.03E7	41.5
	Ours (FPGM-only 30%)	✗	**92.63** (±0.70)	92.31 (±0.30)	0.32	4.03E7	41.5
	Ours (FPGM-only 40%)	✗	**92.63** (±0.70)	**91.93** (±0.03)	**0.70**	**3.23E7**	**53.2**
	Ours (FPGM-mix 40%)	✗	**92.63** (±0.70)	91.91 (±0.21)	0.72	**3.23E7**	**53.2**
56	PFEC [1]	✗	93.04	91.31	1.75	9.09E7	27.6
	CP [14]	✗	92.80	90.90	1.90	–	50.0
	SFP [6]	✗	**93.59** (±0.58)	92.26 (±0.31)	1.33	**5.94E7**	**52.6**
	Ours (FPGM-only 40%)	✗	**93.59** (±0.58)	**92.93** (±0.49)	**0.66**	**5.94E7**	**52.6**
	Ours (FPGM-mix 40%)	✗	**93.59** (±0.58)	92.89 (±0.32)	0.70	**5.94E7**	**52.6**
	PFEC [1]	✓	93.04	93.06	-0.02	9.09E7	27.6
	CP [14]	✓	92.80	91.80	1.00	–	50.0
	AMC [138]	✓	92.80	91.90	0.90	–	50.0
	Ours (FPGM-only 40%)	✓	**93.59** (±0.58)	**93.49** (±0.13)	**0.10**	**5.94E7**	**52.6**
	Ours (FPGM-mix 40%)	✓	**93.59** (±0.58)	93.26 (±0.03)	0.33	**5.94E7**	**52.6**
110	MIL [104]	✗	93.63	93.44	0.19	-	34.2
	PFEC [1]	✗	93.53	92.94	0.61	1.55E8	38.6
	SFP [6]	✗	**93.68** (±0.32)	93.38 (±0.30)	0.30	1.50E8	40.8
	Ours (FPGM-only 40%)	✗	**93.68** (±0.32)	93.73 (±0.23)	-0.05	**1.21E8**	**52.3**
	Ours (FPGM-mix 40%)	✗	**93.68** (±0.32)	**93.85** (±0.11)	**-0.17**	**1.21E8**	**52.3**
	PFEC [1]	✓	93.53	93.30	0.20	1.55E8	38.6
	NISP [17]	✓	–	–	0.18	–	43.8
	Ours (FPGM-only 40%)	✓	**93.68** (±0.32)	**93.74** (±0.10)	**-0.16**	**1.21E8**	**52.3**

Table 5.1 : Comparison of pruned ResNet on CIFAR-10. In "Fine-tune?" column, "✓" and "✗" indicates 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.

Depth	Method	Fine-tune?	Baseline top-1 acc.(%)	Accelerated top-1 acc.(%)	Baseline top-5 acc.(%)	Accelerated top-5 acc.(%)	Top-1 acc. ↓(%)	Top-5 acc. ↓(%)	FLOPs↓(%)
18	MIL [104]	✗	69.98	66.33	89.24	86.94	3.65	2.30	34.6
	SFP [6]	✗	**70.28**	67.10	**89.63**	87.78	3.18	1.85	**41.8**
	Ours (FPGM-only 30%)	✗	**70.28**	67.78	**89.63**	88.01	2.50	1.62	**41.8**
	Ours (FPGM-mix 30%)	✗	**70.28**	**67.81**	**89.63**	**88.11**	**2.47**	**1.52**	**41.8**
	Ours (FPGM-only 30%)	✓	**70.28**	68.34	**89.63**	**88.53**	1.94	**1.10**	**41.8**
	Ours (FPGM-mix 30%)	✓	**70.28**	**68.41**	**89.63**	88.48	**1.87**	1.15	**41.8**
34	SFP [6]	✗	**73.92**	71.83	**91.62**	90.33	2.09	1.29	**41.1**
	Ours (FPGM-only 30%)	✗	**73.92**	71.79	**91.62**	**90.70**	2.13	**0.92**	**41.1**
	Ours (FPGM-mix 30%)	✗	**73.92**	**72.11**	**91.62**	90.69	**1.81**	0.93	**41.1**
	PFEC [1]	✓	73.23	72.17	–	–	**1.06**	–	24.2
	Ours (FPGM-only 30%)	✓	**73.92**	72.54	**91.62**	**91.13**	1.38	**0.49**	**41.1**
	Ours (FPGM-mix 30%)	✓	**73.92**	**72.63**	**91.62**	91.08	1.29	0.54	**41.1**
50	SFP [6]	✗	**76.15**	74.61	**92.87**	92.06	1.54	0.81	41.8
	Ours (FPGM-only 30%)	✗	**76.15**	**75.03**	**92.87**	**92.40**	**1.12**	**0.47**	42.2
	Ours (FPGM-mix 30%)	✗	**76.15**	74.94	**92.87**	92.39	1.21	0.48	42.2
	Ours (FPGM-only 40%)	✗	**76.15**	74.13	**92.87**	91.94	2.02	0.93	**53.5**
	ThiNet [19]	✓	72.88	72.04	91.14	90.67	0.84	0.47	36.7
	SFP [6]	✓	**76.15**	62.14	**92.87**	84.60	14.01	8.27	41.8
	NISP [17]	✓	–	–	–	–	0.89	–	44.0
	CP [14]	✓	–	–	92.20	90.80	–	1.40	50.0
	Ours (FPGM-only 30%)	✓	**76.15**	**75.59**	**92.87**	**92.63**	**0.56**	0.24	42.2
	Ours (FPGM-mix 30%)	✓	**76.15**	75.50	**92.87**	92.63	0.65	**0.21**	42.2
	Ours (FPGM-only 40%)	✓	**76.15**	74.83	**92.87**	92.32	1.32	0.55	**53.5**
101	Rethinking [20]	✓	**77.37**	75.27	–	–	2.10	–	**47.0**
	Ours (FPGM-only 30%)	✓	**77.37**	**77.32**	**93.56**	**93.56**	**0.05**	**0.00**	42.2

Table 5.2 : Comparison of pruned ResNet on ILSVRC-2012. “Fine-tune?” and ”acc. ↓” have the same meaning with Table 3.1.

5.4 Experiments

We evaluate FPGM for single-branch network (VGGNet [118]), and multiple-branch network (ResNet) on two benchmarks: CIFAR-10 [70] and ILSVRC-2012 [105][‡]. The CIFAR-10 [70] dataset contains $60,000$ 32×32 color images in 10 different classes, in which $50,000$ training images and $10,000$ testing images are included. ILSVRC-2012 [105] is a large-scale dataset containing 1.28 million training images and 50k validation images of 1,000 classes.

5.4.1 Experimental Settings

Training setting. On CIFAR-10, the parameter setting is the same as [113] and the training schedule is the same as [114]. In the ILSVRC-2012 experiments, we use the default parameter settings which is same as [61, 113]. Data argumentation strategies for ILSVRC-2012 is the same as PyTorch [115] official examples. We analyze the difference between starting from scratch and the pre-trained model. For pruning the model from scratch, We use the normal training schedule without additional fine-tuning process. For pruning the pre-trained model, we reduce the learning rate to one-tenth of the original learning rate. To conduct a fair comparison of pruning scratch and pre-trained models, we use the same training epochs to train/fine-tune the network. The previous work [1] might use fewer epochs to fine-tune the pruned model, but it converges too early, and its accuracy can not improve even with more epochs, which can be shown in section 5.4.2.

Pruning setting. In the filter pruning step, we simply prune *all* the weighted layers with the *same* pruning rate at the same time, which is the same as [6]. Therefore, only one hyper-parameter $P_i = P$ is needed to balance the acceleration

[‡]As stated in [1], "comparing with AlexNet or VGG (on ILSVRC-2012), both VGG (on CIFAR-10) and Residual networks have fewer parameters in the fully connected layers", which makes pruning filters in those networks challenging.

and accuracy. The pruning operation is conducted at the end of every training epoch. Unlike previous work [1], sensitivity analysis is not essential in FPGM to achieve good performances, which will be demonstrated in later sections.

Apart from FPGM only criterion, we also use a mixture of FPGM and previous norm-based method [6] to show that FPGM could serve as a supplement to previous methods. FPGM only criterion is denoted as "FPGM-only", the criterion combining the FPGM and norm-based criterion is indicated as "FPGM-mix". "FPGM-only 40%" means 40% filters of the layer are selected with FPGM only, while "FPGM-mix 40%" means 30% filters of the layer are selected with norm-based criterion [6], and the remaining 10% filters are selected with FPGM. We compare FPGM with previous acceleration algorithms, e.g., MIL [104], PFEC [1], CP [14], ThiNet [19], SFP [6], NISP [17], Rethinking [20]. Not surprisingly, our FPGM method achieves the state-of-the-art result.

5.4.2 Single-Branch Network Pruning

VGGNet on CIFAR-10. As the training setup is not publicly available for [1], we re-implement the pruning procedure and achieve similar results to the original paper. The result of pruning pre-trained and scratch model is shown in Table 5.3 and Table 5.4, respectively. Not surprisingly, FPGM achieves better performance than [1] in both settings.

5.4.3 Multiple-Branch Network Pruning

ResNet on CIFAR-10. For the CIFAR-10 dataset, we test our FPGM on ResNet-20, 32, 56 and 110 with two different pruning rates: 30% and 40%.

As shown in Table 5.1, our FPGM achieves the state-of-the-art performance. For example, MIL [104] without fine-tuning accelerates ResNet-32 by 31.2% speedup ratio with 1.59% accuracy drop, but our FPGM without fine-tuning achieves 53.2%

Model \ Acc (%)	Baseline	Pruned w.o. FT	FT 40 epochs	FT 160 epochs
PFEC [1]	93.58 (±0.03)	77.45 (±0.03)	93.22 (±0.03)	93.28 (±0.07)
Ours	93.58 (±0.03)	**80.38** (±0.03)	**93.24** (±0.01)	**94.00** (±0.13)

Table 5.3 : Pruning pre-trained VGGNet on CIFAR-10. "w.o." means "without" and "FT" means "fine-tuning" the pruned model.

Model	SA	Baseline	Pruned From Scratch	FLOPs↓(%)
PFEC [1]	Y	93.58 (±0.03)	93.31 (±0.02)	34.2
Ours	Y	93.58 (±0.03)	**93.54** (±0.08)	34.2
Ours	N	93.58 (±0.03)	93.23 (±0.13)	**35.9**

Table 5.4 : Pruning scratch VGGNet on CIFAR-10. "SA" means "sensitivity analysis". Without sensitivity analysis, FPGM can still achieve comparable performances comparing to [1]; after introducing sensitivity analysis, FPGM can surpass [1].

speedup ratio with even 0.19% accuracy improvement. Comparing to SFP [6], when pruning 52.6% FLOPs of ResNet-56, our FPGM has only 0.66% accuracy drop, which is much less than SFP [6] (1.33%). For pruning the pre-trained ResNet-110, our method achieves a much higher (52.3% v.s. 38.6%) acceleration ratio with 0.16% performance increase, while PFEC [1] harms the performance with lower acceleration ratio. These results demonstrate that FPGM can produce a more compressed model with comparable or even better performances.

ResNet on ILSVRC-2012. For the ILSVRC-2012 dataset, we test our FPGM

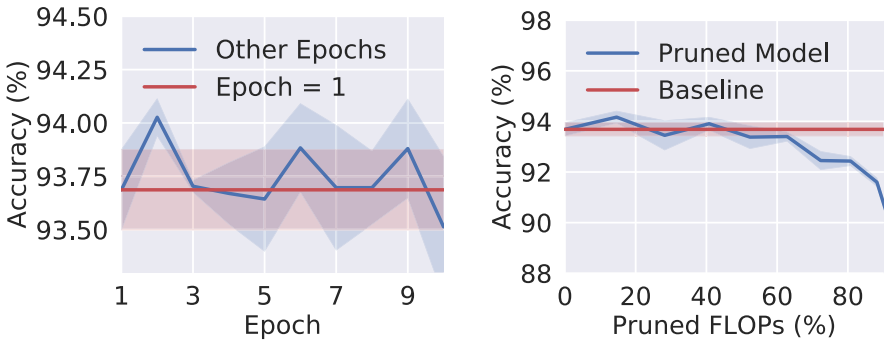
Model	Baseline time (ms)	Pruned time (ms)	Realistic Acc.(%)	Theoretical Acc.(%)
ResNet-18	37.05	26.77	27.7	41.8
ResNet-34	63.89	45.24	29.2	41.1
ResNet-50	134.57	83.22	38.2	53.5
ResNet-101	219.70	147.45	32.9	42.2

Table 5.5 : Comparison on the theoretical and realistic acceleration. Only the time consumption of the forward procedure is considered.

on ResNet-18, 34, 50 and 101 with pruning rates 30% and 40%. Same with [6], we do not prune the projection shortcuts for simplification.

Table 5.2 shows that FPGM outperforms previous methods on ILSVRC-2012 dataset, again. For ResNet-18, pure FPGM without fine-tuning achieves the same inference speedup with [6], but its accuracy exceeds by 0.68%. FPGM-only with fine-tuning could even gain 0.60% improvement over FPGM-only without fine-tuning, thus exceeds [6] by 1.28%. For ResNet-50, FPGM with fine-tuning achieves more inference speedup than CP [14], but our pruned model exceeds their model by 0.85% on the accuracy. Moreover, for pruning a pre-trained ResNet-101, FPGM reduces more than 40% FLOPs of the model without top-5 accuracy loss and only negligible (0.05%) top-1 accuracy loss. In contrast, the performance degradation is 2.10% for Rethinking [20]. Compared to the norm-based criterion, Geometric Median (GM) explicitly utilizes the relationship between filters, which is the main cause of its superior performance.

To compare the theoretical and realistic acceleration, we measure the forward time of the pruned models on one GTX1080 GPU with a batch size of 64. The



(a) Different pruning intervals (b) Different pruned FLOPs

Figure 5.4 : Accuracy of ResNet-110 on CIFAR-10 regarding different hyper-parameters. Solid line and shadow denotes the mean values and standard deviation of three experiments, respectively.

results [§] are shown in Table 5.5. As discussed in the above section, the gap between the theoretical and realistic model may come from the limitation of IO delay, buffer switch, and efficiency of BLAS libraries.

5.4.4 Ablation Study

Influence of Pruning Interval In our experiment setting, the interval of pruning equals to one, *i.e.*, we conduct our pruning operation at the end of every training epoch. To explore the influence of pruning interval, we change the pruning interval from one epoch to ten epochs. We use the ResNet-110 under pruning rate 40% as the baseline, as shown in Fig. 5.4(a). The accuracy fluctuation along with the different pruning intervals is less than 0.3%, which means the performance of pruning is not sensitive to this parameter. Note that fine-tuning this parameter could even achieve better performance.

[§]Optimization of the addition of ResNet shortcuts and convolutional outputs would also affect the results.

Varying Pruned FLOPs We change the ratio of Pruned FLOPs for ResNet-110 to comprehensively understand FPGM, as shown in Fig. 5.4(b). When the pruned FLOPs is 18% and 40%, the performance of the pruned model even exceeds the baseline model without pruning, which shows FPGM may have a regularization effect on the neural network.

Influence of Distance Type We use ℓ_1-norm and cosine distance to replace the distance function in Equation 5.4. We use the ResNet-110 under pruning rate 40% as the baseline, the accuracy of the pruned model is 93.73 $\pm$ 0.23 %. The accuracy based on ℓ_1-norm and cosine distance is 93.87 $\pm$ 0.22 % and 93.56 $\pm$ 0.13, respectively. Using ℓ_1-norm as the distance of filter would bring a slightly better result, but cosine distance as distance would slightly harm the performance of the network.

Combining FPGM with Norm-based Criterion We analyze the effect of combining FPGM and previous norm-based criterion. For ResNet-110 on CIFAR-10, FPGM-mix is slightly better than FPGM-only. For ResNet-18 on ILSVRC-2012, the performances of FPGM-only and FPGM-mix are almost the same. It seems that the norm-based criterion and FPGM together can boost the performance on CIFAR-10, but not on ILSVRC-2012. We believe that this is because the two requirements for the norm-based criterion are met on some layers of CIFAR-10 pre-trained network, but not on that of ILSVRC-2012 pre-trained network, which is shown in Figure 5.3.

5.4.5 Feature Map Visualization

We visualize the feature maps of the first layer of the first block of ResNet-50. The feature maps with red titles (7,23,27,46,56,58) correspond to the selected filter activation when setting the pruning rate to 10%. These selected feature maps contain outlines of the bamboo and the panda's head and body, which can be replaced by remaining feature maps: (5,12,16,18,22, *et. al.*) containing outlines of the

bamboo, and (0,4,33,34,47, *et. al.*) containing the outline of panda.

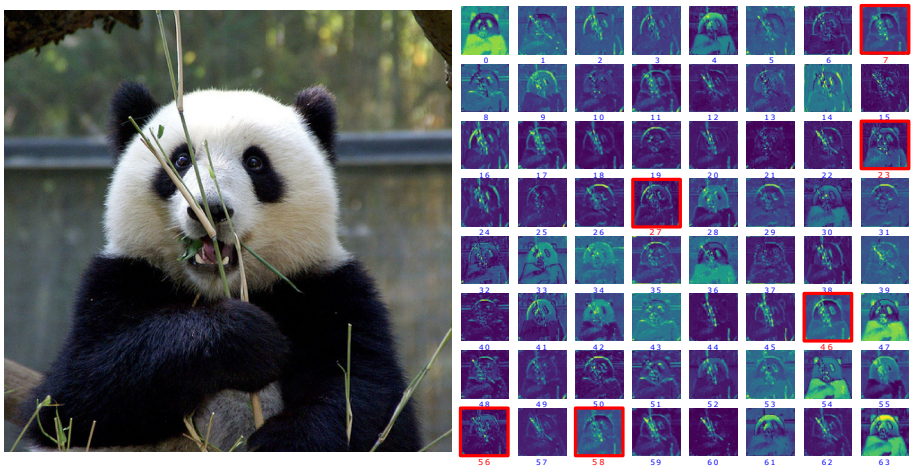


Figure 5.5 : Input image (left) and visualization of feature maps (right) of ResNet-50-conv1. Feature maps with red bounding boxes are the channels to be pruned.

5.5 Conclusion

In this paper, we elaborate on the underlying requirements for norm-based filter pruning criterion and point out their limitations. To solve this, we propose a new filter pruning strategy based on the geometric median, named FPGM, to accelerate the deep CNNs. Unlike the previous norm-based criterion, FPGM explicitly considers the mutual relations between filters. Thanks to this, FPGM achieves the state-of-the-art performance in several benchmarks.

Chapter 6

Learning Filter Pruning Criteria

6.1 Introduction

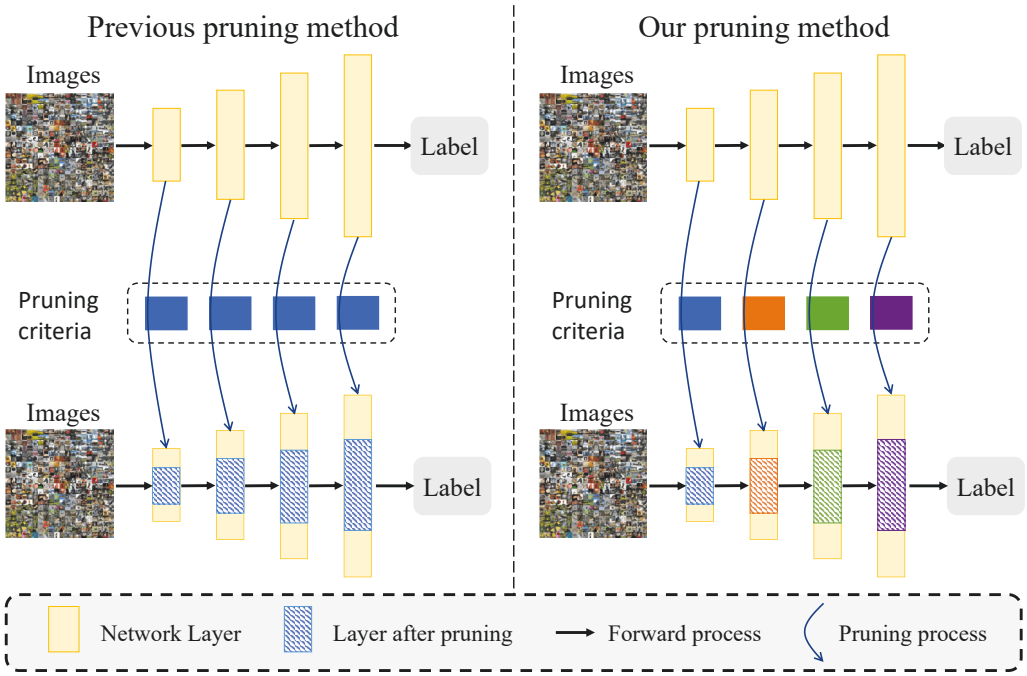


Figure 6.1 : (a) Previous filter pruning methods manually select a criterion and apply it to all layers; (b) our pruning method learns appropriate criteria for different layers based on the filter distribution. In the blue dashed box, the solid boxes of different colors denote different pruning criteria. 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.

Convolutional neural networks have achieved significant advancement in various computer vision research applications [61, 118, 139]. However, most of these man-

ually designed architectures, *e.g.*, VGG [118], ResNet [61], 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 [1, 4, 7].

Recent developments on pruning can be divided into two categories, *i.e.*, weight pruning [4], and filter pruning [1]. Filter pruning is preferred compared to weight pruning because filter pruning could make the pruned model more structural and achieve practical acceleration [7]. 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 [140], 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. 6.3.2.3. 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. 6.1. 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.

6.2 Related Work

Previous work on pruning can be categorized into weight pruning and filter pruning. Weight pruning [4,5,66,126,132–134] focuses on pruning fine-grained weight of filters, so that leading to unstructured sparsity in models. In contrast, filter pruning [1] 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. 6.1.

Algorithms	Criteria W | A	Frequency O | G
PFEC [1], SFP [6], FPGM [7]	W	O
RSA [20], PRE [16]	W	G
SLIM [15], PFA [8], NISP [17], CCP [12], GAL [23]	A	O
CP [14], SLIM [15], ThiNet [19], PRE [16], DCP [21], LPF [22], AOFP [18], GATE [13]	A	G

Table 6.1 : 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 [1, 6, 7, 20, 29, 141] utilize the weights of the filters to determine the importance of the filters. [1] prunes the filters with small ℓ_1-norm. [6] utilizes ℓ_2-norm criterion to select filters and prune those selected filters softly. [20] introduces sparsity on the scaling parameters of batch normalization (BN) layers to prune the network. [7] 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 [8, 12, 14–17, 19, 21–24, 27, 71, 138, 142] utilize the training data and filter activations to determine the pruned filters. [8] adopts the Principal Component Analysis (PCA) method to specify which part of the network should be preserved. [19] proposes to use the information from the next layer to guide the filter selection. [71] minimizes the reconstruction error of training set sample activations and applies Singular Value Decomposition (SVD) to obtain a decomposition of filters. [73] explores the linear relationship in different feature maps to eliminate the redundancy in convolutional filters.

Greedy and One-shot Pruning. Greedy pruning [13, 18], 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 [1, 7] 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 [22, 138] or meta-learning [143] for pruning. In contrast, we focus on learning the proper pruning criteria for different layers via the differential sampler.

[144] proposes centripetal SGD to make several filters to converge into a single point. [17] 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 [75, 145] and Autoaugment [146], the difference is that our search space is the pruning criteria instead of network architectures or augmentation policies.

6.3 Methodology

6.3.1 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:

$$\mathbf{O}_i = \mathbf{W}_i^{(l)} * \mathbf{I} \quad \textbf{for } 1 \le i \le C_O^{(l)}, \tag{6.1}$$

Assume the filter set $\mathcal{F}$ consists all the filters in the network:

$$\mathcal{F} = \left\{ \mathbf{W}_i^{(l)},\ \ i \in [1, C_O^{(l)}],\ \ l \in [1, L] \right\} \tag{6.2}$$

We divide $\mathcal{F}$ into two disjoint subsets: the kept filter set $\mathcal{K}$ and removed filter set $\mathcal{R}$, and we have:

$$\mathcal{K} \cup \mathcal{R} = \mathcal{F}, \quad \mathcal{K} \cap \mathcal{R} = \emptyset. \tag{6.3}$$

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:

$$\begin{aligned} \min_{\mathcal{K}} \mathcal{L}(\mathcal{K}; \mathcal{D}) &= \min_{\mathcal{K}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathcal{K}; (\mathbf{x}_n, \mathbf{y}_n)) \\ \text{s.t. } & \frac{\mathcal{C}(\mathcal{K})}{\mathcal{C}(\mathcal{F})} \leq r \end{aligned} \tag{6.4}$$

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.

6.3.2 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. 6.2.

6.3.2.1 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

$$score^{(l)} = \text{Crit}^{(l)}(\mathcal{F}^{(l)}) \tag{6.5}$$

where $score^{(l)} \in \mathbb{R}^{C_O^{(l)}}$ is the importance score vector of the filters in l_{th} layer. For example, ℓ_1-norm criteria [1] could be formulated as:

$$\text{Crit}^{(l)}(\mathcal{F}^{(l)}) = \left\{ \text{Crit}^{(l)}(\mathbf{W}_i^{(l)}) = \|\mathbf{W}_i^{(l)}\|_1 \ \text{ for } \ i \in [1, C_O^{(l)}] \right\} \tag{6.6}$$

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

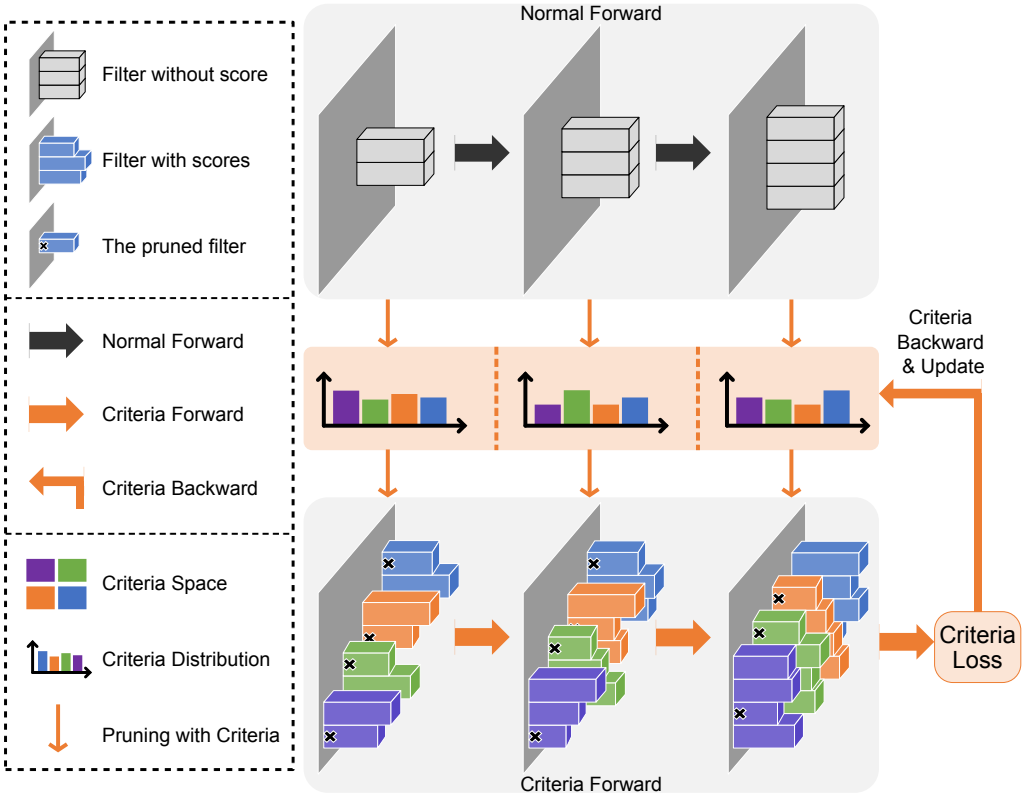


Figure 6.2 : Criteria forward and backward in the network. Grey boxes are the normal filters. The probability distribution of criteria for three layers are initialized, as shown in the big orange shadow. After pruning with four criteria, we obtain four "pruned versions" for every layer, which are denoted 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".

$$
\begin{aligned}
keepid^{(l)} &= \text{Topk}(score^{(l)}, \text{n}^{(l)}) \\
\mathcal{K}^{(l)} &= \text{Prune}(\mathcal{F}^{(l)}, keepid^{(l)}),
\end{aligned}
\tag{6.7}
$$

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

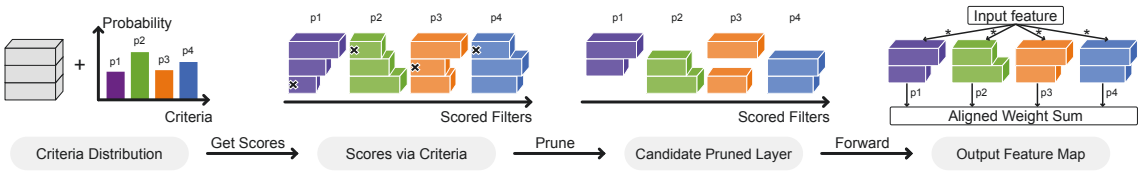


Figure 6.3 : Criteria forward within a layer. Boxes of different colors indicate the different pruning criteria. First, we evaluate the importance of the filter based on different criteria. Second, we prune the filter with small importance scores and get four versions of pruned layers with various probabilities. After that, the output feature map is the aligned weighted sum of four feature maps of the pruned layers.

6.3.2.2 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 [147]. 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 [1, 7], 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.

6.3.2.3 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:

$$p_i = \frac{\exp\left(\alpha_i^{(l)}\right)}{\sum_{j=1}^{S}\exp\left(\alpha_j^{(l)}\right)} \quad \text{where} \quad 1 \le i \le S \tag{6.8}$$

However, since Eq. 6.8 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 [117], we apply Gumbel-Softmax [148, 149] to reformulate Eq. 6.8 as Eq. 6.9:

$$\begin{aligned} \hat{p}_i &= \frac{\exp\left(\left(\log\left(p_i\right) + \boldsymbol{o}_i\right)/\tau\right)}{\sum_{j=1}^{S} \exp\left(\left(\log\left(p_j\right) + \boldsymbol{o}_j\right)/\tau\right)} \\ \text{s.t. } \boldsymbol{o}_i &= -\log(-\log(u)) \ \ \& \ \ u \sim \mathcal{U}(0,1) \end{aligned} \tag{6.9}$$

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 τ is the softmax temperature. We denote $\hat{p} = [\hat{p}_1, \ldots, \hat{p}_j, \ldots]$ as the Gumbel-softmax distribution. Change the parameter τ 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 [148].

Criteria Forward. The illustration of criteria forward for l_{th} layer is shown in Fig 6.3. For simplicity, we rewrite the Eq. 6.5 and Eq. 6.7 as $\mathcal{K}^{(l)} = g(\mathcal{F}^{(l)}, \text{Crit}^{(l)}, \text{n}^{(l)})$. For l_{th} layer, it has S sampled "pruned version" which can be formulated as:

$$\mathcal{K}_s^{(l)} = g(\mathcal{F}^{(l)}, \text{Crit}_s^{(l)}, \text{n}^{(l)}) \ \text{ for } \ s \in [1, S] \tag{6.10}$$

where $\text{Crit}_s^{(l)}$ 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:

$$\begin{aligned} \mathbf{O}^{AWS} &= \sum_{s=1}^{S} \text{Align}(\hat{p}_s \times \hat{\mathbf{O}}_s), \\ \text{Align}(\hat{\mathbf{O}}_{s,i}) &= \hat{\mathbf{O}}'_{s,keepid_s^{(l)}[i]} \qquad i \in [1, \text{n}^{(l)}]. \end{aligned} \tag{6.11}$$

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. 6.7) under s_{th} criteria in l_{th} layer. To explain the Align$(\cdot)$ function, we take the third figure of Fig. 6.3 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:

$$\begin{aligned} \hat{\mathbf{O}}_s &= \left[\hat{\mathbf{O}}_{s,1}, \hat{\mathbf{O}}_{s,2}, ..., \hat{\mathbf{O}}_{s,\mathrm{n}^{(l)}}\right], \\ \hat{\mathbf{O}}_{s,i} &= \mathcal{K}_{s,i}^{(l)} * \mathbf{I}, \qquad s \in [1, S], \;\; i \in [1, \mathrm{n}^{(l)}], \end{aligned} \tag{6.12}$$

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. 6.2.

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 α to give us guidance about which criterion is suitable for different layers. Specifically, α is found by minimizing the validation loss $\mathcal{L}_{val}$ after trained the *criteria network* θ_α by minimizing the training loss $\mathcal{L}_{train}$:

$$\begin{aligned} &\min_{\alpha} \mathcal{L}_{val}\left(\theta_\alpha^*, \alpha\right) \\ \text{s.t.} \quad &\theta_\alpha^* = \arg\min_{\theta_\alpha} \mathcal{L}_{train}(\theta_\alpha, \alpha), \end{aligned} \tag{6.13}$$

where θ_α^* is the optimized *criteria network* under the optimized criteria set α. 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:

$$\mathcal{L}_{val} = \mathcal{L}_{crit} + \lambda_{comp}\mathcal{L}_{comp}, \tag{6.14}$$

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. λ_{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 α for the network under different computation constraints.

Criteria Backward. We backward $\mathcal{L}_{val}$ in Eq. 6.14 to α to update these parameters collaboratively at the same time. The illustration of this process is shown in Fig. 6.2.

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.

6.4 Experiments

6.4.1 Experimental Setting

Datasets. In this section, we validate the effectiveness of our acceleration method on three benchmark datasets, CIFAR-10, CIFAR-100 [70], and ILSVRC-2012 [105]. The CIFAR-10 dataset contains $50,000$ training images and $10,000$ testing images, in total $60,000$ 32×32 color images in 10 different classes. CIFAR-100 has 100 classes, and the number of images is the same as CIFAR-10. ILSVRC-2012 [105] contains 1.28 million training images and 50k validation images of $1,000$ classes.

Architecture Setting. As ResNet has the shortcut structure, existing works [14, 19,104] claim that ResNet has less redundancy than VGGNet [118] and accelerating ResNet is more difficult than accelerating VGGNet. Therefore, we follow [151] to focus on pruning the challenging ResNet.

Depth	Method	Init pretrain	Baseline acc. (%)	Pruned acc. (%)	Acc. ↓ (%)	FLOPs	FLOPs ↓(%)
32	MIL [104]	✗	92.33	90.74	1.59	4.70E7	31.2
	SFP [6]	✗	**92.63** (±0.70)	92.08 (±0.08)	0.55	4.03E7	41.5
	FPGM [7]	✗	**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 [1]	✗	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 [14]	✗	92.80	90.90	1.90	–	50.0
	SFP [6]	✗	**93.59** (±0.58)	92.26 (±0.31)	1.33	5.94E7	52.6
	FPGM [7]	✗	**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 [1]	✓	93.04	93.06	-0.02	9.09E7	27.6
	NISP [17]	✓	–	–	0.03	–	42.6
	Ours	✓	**93.59** (±0.58)	**93.72** (±0.29)	**-0.13**	6.64E7	47.1
	CP [14]	✓	92.80	91.80	1.00	–	50.0
	AMC [138]	✓	92.80	91.90	0.90	–	50.0
	FPGM [7]	✓	**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 [1]	✗	93.53	92.94	0.61	1.55E8	38.6
	MIL [104]	✗	93.63	93.44	0.19	-	34.2
	SFP [6]	✗	93.68 (±0.32)	93.38 (±0.30)	0.30	1.50E8	40.8
	Rethink [150]	✗	**93.77** (±0.23)	93.70 (±0.16)	0.07	1.50E8	40.8
	FPGM [7]	✗	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 [1]	✓	93.53	93.30	0.20	1.55E8	38.6
	NISP [17]	✓	–	–	0.18	–	43.8
	GAL [23]	✓	93.26	92.74	0.81	–	48.5
	FPGM [7]	✓	**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 6.2 : 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.

Normal Training Setting. For ResNet on CIFAR-10 and CIFAR-100, we utilize the same training schedule as [114]. In the CIFAR experiments, we run each setting three times and report the "mean ± std". In the ILSVRC-2012 experiments, we use the default parameter settings, which are the same as [61, 113], and the same data argumentation strategies as the official PyTorch [115] examples.

DCS training Setting. The weight-based criteria are selected as our candidate criteria for their efficiency. Specifically, ℓ_1-norm [1], ℓ_2-norm [6] and geometric median based [7] 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 λ_{comp} of Eq. 6.14 as 2. We randomly split half of the training set as the validation set for Eq. 6.13. 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 τ in Eq. 6.9 is linearly decayed from 5 to 0.1. During training DCS, we fix the pre-trained weights [151] 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 between 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 [7] for pruning. During retraining, we use the cosine scheduler [151, 152] for a stable result. The pruning rate of every layer is sampled in the same way as

DCS*, so we could search the ratio automatically and adaptively [151].

We compare our method with existing state-of-the-art acceleration algorithms, *e.g.*, MIL [104], PFEC [1], CP [14], ThiNet [19], SFP [6], NISP [17], FPGM [7], LFC [153], ELR [73], GAL [23], IMP [27], DDS [24]. Experiments show that our LFPC achieves a comparable performance with those works. Our experiments are based on the PyTorch [115] framework. No significant performance difference has been observed with the PaddlePaddle framework.

6.4.2 ResNet on CIFAR-10

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 [6]	✗	**76.15**	**74.61**	**92.87**	**92.06**	**1.54**	**0.81**	41.8
	FPGM [7]	✗	**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 [19]	✓	72.88	72.04	91.14	90.67	**0.84**	**0.47**	36.7
	SFP [6]	✓	76.15	62.14	**92.87**	84.60	14.01	8.27	41.8
	NISP [17]	✓	–	–	–	–	0.89	–	44.0
	IMP [27]	✓	**76.18**	74.50			1.68		45.0
	CP [14]	✓	–	–	92.20	90.80	–	1.40	50.0
	LFC [153]	✓	75.30	73.40	92.20	91.40	1.90	0.80	50.0
	ELR [73]	✓	–	–	92.20	91.20	–	1.00	50.0
	FPGM [7]	✓	**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 6.3 : Comparison of the pruned ResNet on ImageNet. "Init Pretrain" and "acc. ↓" have the same meaning with Table 3.1.

For the CIFAR-10 dataset, we test our LFPC on ResNet with depth 32, 56, and 110. As shown in Tab. 6.2, the experiment results validate the effectiveness of

*See supplementary material for details.

our method. For example, MIL [104] 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 [7] on ResNet-32, our acceleration ratio is much larger than FPGM [7]. Comparing to SFP [6], when we prune similar FLOPs of the random initialized ResNet-56, our LFPC has 1.07% accuracy improvement over SFP [6]. For pruning the pre-trained ResNet-56, our method achieves a higher acceleration ratio than CP [14] with a 0.65% accuracy increase over CP [14]. Comparing to PFEC [1], our method accelerates the random initialized ResNet-110 by 60.3% speedup ratio with even 0.11% accuracy improvement, while PFEC [1] 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 [1, 6, 14] did this. We notice that pruning from a scratch model sometimes achieves a slightly better performance than pruning a pre-trained model, which is consistent with [150]. Note that we achieve a higher acceleration ratio than [150] 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 [154] 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 6.4. The blue, orange and green denote pruning this layer with ℓ_1-norm, ℓ_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 ℓ_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

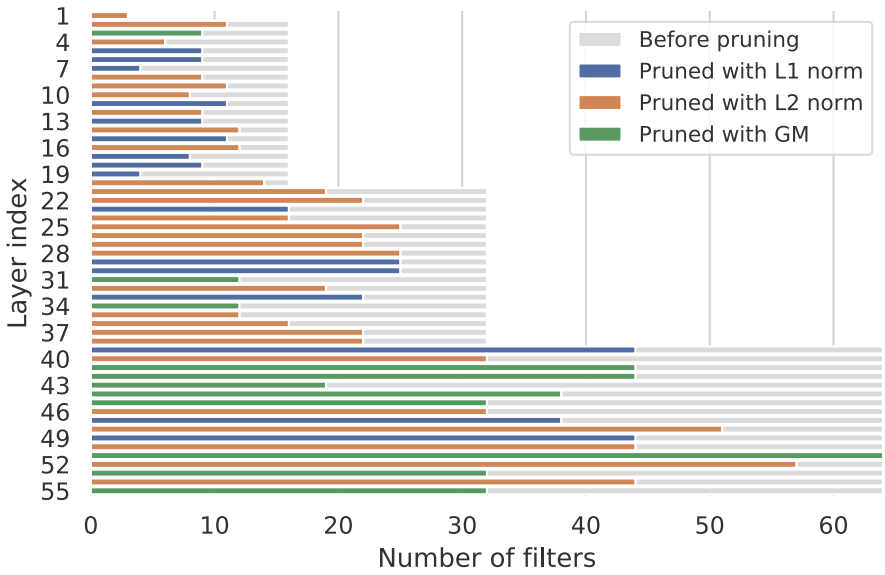


Figure 6.4 : Visualization of the learned criteria and kept filters for ResNet-56 on CIFAR-10. The grey strip indicates the layers before pruning. The blue, orange and green color denote ℓ_1-norm, ℓ_2-norm and geometric median criteria, respectively. For example, the bottom green strip means that for 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.

to each other [155]. Therefore, our LFPC chooses the relation-based criteria instead of magnitude-based criteria when pruning higher layers. [†]

6.4.3 ResNet on CIFAR-100

The results of pruning ResNet-56 on CIFAR-100 is shown in Tab. 6.4. 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 [6] and [7]. This result again validates the effectiveness of our method.

[†]GM is a relation-based criterion, while ℓ_p-norm is a magnitude-based criterion. See supplementary material for different filter distribution.

Depth	Method	Pruned Acc.(%)	Acc. ↓ (%)	FLOPs	FLOPs ↓ (%)
56	MIL [104]	68.37	2.96	7.63E7	39.3%
	SFP [6]	68.79	2.61	5.94E7	52.6%
	FPGM [7]	69.66	1.75	5.94E7	52.6%
	Ours	70.83	0.58	6.08E7	51.6%

Table 6.4 : Comparison of the pruned ResNet-56 on CIFAR-100.

6.4.4 ResNet on ILSVRC-2012

For the ILSVRC-2012 dataset, we test our method on ResNet-50. Same as [7], we do not prune the projection shortcuts. Tab. 6.3 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 [7], the accuracy is even higher than FPGM [7]. 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 [14], LFC [153], ELR [73]) have lower top-5 accuracy when pruning less FLOPs (50%). ThiNet [19] 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.

6.4.5 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** ℓ_p-norm. In contrast, adversarial norm-based criteria keep the filters with

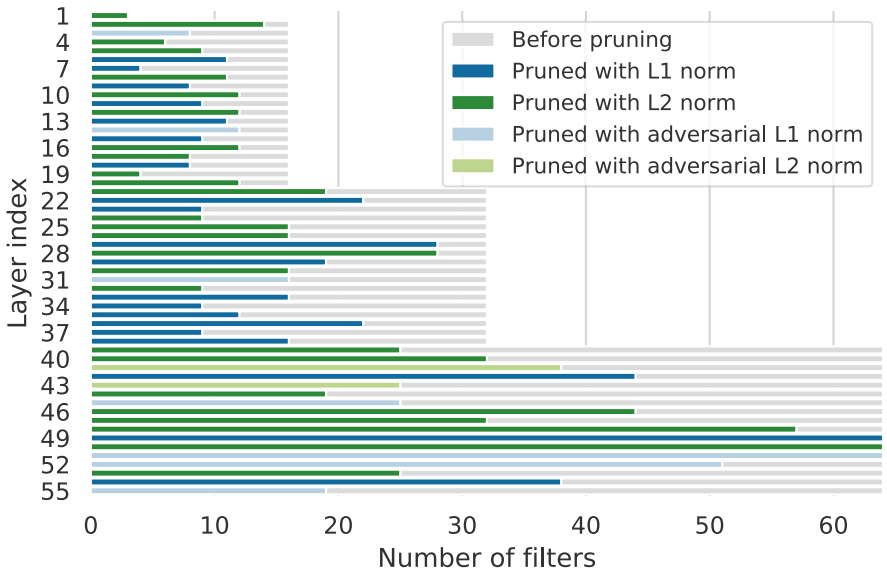


Figure 6.5 : Visualization of the conventional and adversarial criteria for ResNet-56 on CIFAR-10. The grey strip indicates the layers before pruning. Different blue and green colors represent different pruning criteria.

small ℓ_p-norm, which could be formulate as $\text{Crit}^{(l)}(\mathbf{W}_i^{(l)}) = \frac{1}{\|\mathbf{W}_i^{(l)}\|_p}$ for $i \in [1, C_O^{(l)}]$.

The learned criteria for ResNet-56 on CIFAR-10 are shown in Fig. 6.5. In this experiment, we utilize four criteria, including ℓ_1-norm, ℓ_2-norm, adversarial ℓ_1-norm, adversarial ℓ_2-norm. As shown in Tab. 6.5, 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. 6.6. 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.

Retraining Scheduler. We compare the cosine scheduler [152] and step sched-

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

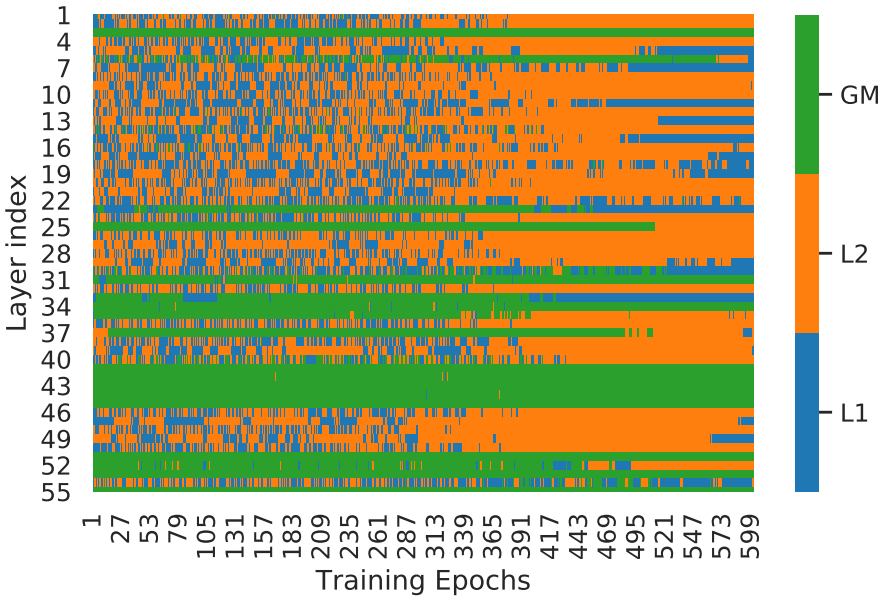


Figure 6.6 : The learned criteria during training the criteria sampler. The L1, L2, and GM denote conventional ℓ_1-norm, ℓ_2-norm, and geometric median criteria, respectively.

uler [7] 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.

6.5 Conclusion

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.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we investigate the filter pruning methods for accelerating deep neural networks. We contribute in four aspects of filter pruning.

- Pruning mechanism.

  We propose a soft filter pruning (SFP) approach improve the pruning mechanism when accelerating the deep CNNs. During the training procedure, SFP allows the pruned filters to be updated. As a result, the wrongly pruned filters would have a chance to come back. Also, this soft manner can maintain the model capacity and thus achieve the superior performance.

- Pruning ratio.

  We propose an asymptotic soft filter pruning approach (ASFP) to asymptotically adjust the pruning rate. As a result, the sudden information loss is avoided and the asymptotic pruning could make the pruning process more stable. Therefore, our ASFP could achieve superior performance.

- Pruning criteria.

  We elaborate on the underlying requirements for norm-based filter pruning criterion and point out their limitations. To solve this, we propose a new filter pruning strategy based on the geometric median, named FPGM, to accelerate the deep CNNs. Unlike the previous norm-based criterion, FPGM explicitly

considers the mutual relations between filters. Thanks to this, FPGM achieves the state-of-the-art performance in several benchmarks.

- Automatic pruning.

  We propose a new learning filter pruning criteria (LFPC) framework to explicitly consider 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.

7.2 Future Work

There are several recommendations for future research.

- More pruning methods. First, the theoretical demonstration of the information loss of the network after pruning need to be investigated. Second, the schedule of the pruning rate is hand-crafted and may not be the best schedule, so this direction can be explored. Third, we could consider utilizing more kinds of criteria into LFPC. Fourth, filter ranking strategies and filter pruning ratios could be optimized jointly. Moreover, it is meaningful to adopt the proposed method to recent compact ConvNets such as MobileNets.

- Combination with other parallel acceleration algorithms. Our pruning methods can be combined with other acceleration algorithms, e.g., matrix decomposition and low-precision weights, to further improve the performance. Also, Neural architecture search is another meaningful direction that can be combined with pruning.

- Applying hardware-aware techniques to other applications. Natural language processing (NLP) has become a crucial research direction for the interac-

tions between computers and human language. As the language models such as BERT and GPT are becoming larger, low-latency inference on resource-constrained hardware platforms desires more attention. Applying deep neural networks to IoT devices could bring about a generation of applications capable of performing complex sensing and recognition tasks. Wearable devices like fitbits and smartwatches provide crucial healthcare services such as position tracking, heart rate monitoring, and sleep monitoring. Hardware-aware techniques can provide fast inference in these applications. Moreover, the small energy consumption means our approach can offer a longer operation time.

- Pruned Network Safety. Deep neural networks are vulnerable to adversarial attacks, and this may cause a serious problem in scenarios such as self-driving cars making a bad reaction. Therefore, special attention should be paid to the safety of machine learning algorithms and the robustness of neural network models.

- Pruned Network Privacy. Model privacy is the protection of intellectual property since a model can be stolen outright or can be reverse-engineered based on its outputs. The challenges of preserving model privacy on hardware are driving the development of new applicable algorithms, where I have a keen interest.

Bibliography

[1] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[2] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 598–605.

[3] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016.

[4] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015.

[5] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proc. Int. Conf. Learn. Represent.*, 2015.

[6] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018.

[7] Y. He, P. Liu, Z. Wang, and Y. Yang, "Pruning filter via geometric median for deep convolutional neural networks acceleration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019.

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

2018.

[9] H. Zhuo, X. Qian, Y. Fu, H. Yang, and X. Xue, "Scsp: Spectral clustering filter pruning with soft self-adaption manners," *arXiv preprint arXiv:1806.05320*, 2018.

[10] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang, "Learning filter pruning criteria for deep convolutional neural networks acceleration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020.

[11] T.-W. Chin, R. Ding, C. Zhang, and D. Marculescu, "Towards efficient model compression via learned global ranking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020.

[12] H. Peng, J. Wu, S. Chen, and J. Huang, "Collaborative channel pruning for deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5113–5122.

[13] 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.

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

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

[16] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[17] 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 *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018.

[18] X. Ding, G. Ding, Y. Guo, J. Han, and C. Yan, "Approximated oracle filter pruning for destructive cnn width optimization," in *Proc. Int. Conf. Mach. Learn.*, 2019.

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

[20] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," in *Proc. Int. Conf. Learn. Represent.*, 2018.

[21] 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 *Proc. Adv. Neural Inf. Process. Syst.*, 2018.

[22] Q. Huang, K. Zhou, S. You, and U. Neumann, "Learning to prune filters in convolutional neural networks," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 709–718.

[23] 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 *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 2790–2799.

[24] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Eur. Conf. Comput. Vis.*, 2018, pp. 304–320.

[25] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 2780–2789.

[26] J. Li, Q. Qi, J. Wang, C. Ge, Y. Li, Z. Yue, and H. Sun, "Oicsr: Out-in-channel sparsity regularization for compact deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 7046–7055.

[27] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 11 264–11 272.

[28] D. Mehta, K. I. Kim, and C. Theobalt, "On implicit filter level sparsity in convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 520–528.

[29] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, and Y. Yang, "Asymptotic soft filter pruning for deep convolutional neural networks," *IEEE Trans. Cybern.*, vol. 50, no. 8, pp. 3594–3604, 2019.

[30] Y. Zhou, Y. Zhang, Y. Wang, and Q. Tian, "Accelerate cnn via recursive bayesian pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 3306–3315.

[31] S. Guo, Y. Wang, Q. Li, and J. Yan, "Dmcp: Differentiable markov channel pruning for neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020.

[32] L. Liebenwein, C. Baykal, H. Lang, D. Feldman, and D. Rus, "Provable filter pruning for efficient neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2020.

[33] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "Apq: Joint search for network architecture, pruning and quantization policy," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 2078–2087.

[34] S. Gao, F. Huang, J. Pei, and H. Huang, "Discrete model compression with resource constraint for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020.

[35] T. Li, J. Li, Z. Liu, and C. Zhang, "Few sample knowledge distillation for efficient network compression," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 14 639–14 647.

[36] Y. Li, S. Gu, C. Mayer, L. V. Gool, and R. Timofte, "Group sparsity: The hinge between filter pruning and decomposition for network compression," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 8018–8027.

[37] J. Guo, W. Ouyang, and D. Xu, "Multi-dimensional pruning: A unified framework for model compression," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 1508–1517.

[38] J.-H. Luo and J. Wu, "Neural network pruning with residual-connections and limited-data," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 1458–1467.

[39] J. T. C. Min and M. Motani, "Dropnet: Reducing neural network complexity via iterative pruning," in *Proc. Int. Conf. Mach. Learn.*, 2020.

[40] M. Kang and B. Han, "Operation-aware soft channel pruning using differentiable masks," in *Proc. Int. Conf. Mach. Learn.*, 2020.

[41] M. Ye, C. Gong, L. Nie, D. Zhou, A. Klivans, and Q. Liu, "Good subnetworks provably exist: Pruning via greedy forward selection," in *Proc. Int. Conf. Mach. Learn.*, 2020.

[42] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," in *Proc. Int. Joint Conf. Artif. Intell.*, 2020.

[43] Y. Wang, Y. Lu, and T. Blankevoort, "Differentiable joint pruning and quantization for hardware efficiency," in *Eur. Conf. Comput. Vis.*, 2020.

[44] Y. Li, S. Gu, K. Zhang, L. Van Gool, and R. Timofte, "Dhp: Differentiable meta pruning via hypernetworks," in *Eur. Conf. Comput. Vis.*, 2020.

[45] X. Ning, T. Zhao, W. Li, P. Lei, Y. Wang, and H. Yang, "Dsa: More efficient budgeted pruning via differentiable sparsity allocation," in *Eur. Conf. Comput. Vis.*, 2020.

[46] B. Li, B. Wu, J. Su, G. Wang, and L. Lin, "Eagleeye: Fast sub-net evaluation for efficient neural network pruning," in *Eur. Conf. Comput. Vis.*, 2020.

[47] W. Kim, S. Kim, M. Park, and G. Jeon, "Neuron merging: Compensating for pruned neurons," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020.

[48] T. Zhuang, Z. Zhang, Y. Huang, X. Zeng, K. Shuang, and X. Li, "Neuron-level structured pruning using polarization regularizer," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020.

[49] Y. Tang, Y. Wang, Y. Xu, D. Tao, C. Xu, C. Xu, and C. Xu, "Scop: Scientific control for reliable neural network pruning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020.

[50] J. Chen, S. Chen, and S. J. Pan, "Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning," in *Proc. Adv.*

Neural Inf. Process. Syst., 2020.

[51] B. R. Bartoldson, A. S. Morcos, A. Barbu, and G. Erlebacher, "The generalization-stability tradeoff in neural network pruning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020.

[52] F. Meng, H. Cheng, K. Li, H. Luo, X. Guo, G. Lu, and X. Sun, "Pruning filter in filter," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020.

[53] Q. Li, C. Li, and H. Chen, "Incremental filter pruning via random walk for accelerating deep convolutional neural networks," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 358–366.

[54] ——, "Filter pruning via probabilistic model-based optimization for accelerating deep convolutional neural networks," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021, pp. 653–661.

[55] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *BMVC*, 2014.

[56] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE T-PAMI*, 2016.

[57] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015.

[58] C. Tai, T. Xiao, Y. Zhang, X. Wang *et al.*, "Convolutional neural networks with low-rank regularization," in *Proc. Int. Conf. Learn. Represent.*, 2016.

[59] J. Park, S. Li, W. Wen, P. T. P. Tang, H. Li, Y. Chen, and P. Dubey, "Faster cnns with direct sparse convolutions and guided pruning," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[60] J.-T. Chien and Y.-T. Bao, "Tensor-factorized neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 5, pp. 1998–2011, 2018.

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

[62] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[63] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[64] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.

[65] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[66] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016.

[67] V. Lebedev and V. Lempitsky, "Fast ConvNets using group-wise brain damage," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016.

[68] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[69] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[70] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

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

[72] Y. He and S. Han, "ADC: Automated deep compression and acceleration with reinforcement learning," in *Eur. Conf. Comput. Vis.*, 2018.

[73] 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.

[74] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4340–4349.

[75] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[76] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *ICML*, 2018.

[77] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," in *ICML*, 2017.

[78] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[79] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[80] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[81] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 734–10 742.

[82] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[83] B. Heo, S. Yun, D. Han, S. Chun, J. Choe, and S. J. Oh, "Rethinking spatial dimensions of vision transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 11 936–11 945.

[84] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, and M. Douze, "Levit: a vision transformer in convnet's clothing for faster

inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 259–12 269.

[85] Y. Rao, W. Zhao, B. Liu, J. Lu, J. Zhou, and C.-J. Hsieh, "Dynamicvit: Efficient vision transformers with dynamic token sparsification," *Advances in neural information processing systems*, vol. 34, 2021.

[86] Y. Liang, G. Chongjian, Z. Tong, Y. Song, J. Wang, and P. Xie, "Evit: Expediting vision transformers via token reorganizations," in *International Conference on Learning Representations*, 2021.

[87] M. Zhu, K. Han, Y. Tang, and Y. Wang, "Visual transformer pruning," *arXiv e-prints*, pp. arXiv–2104, 2021.

[88] M. Chen, H. Peng, J. Fu, and H. Ling, "Autoformer: Searching transformers for visual recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 270–12 280.

[89] W. Chen, W. Huang, X. Du, X. Song, Z. Wang, and D. Zhou, "Auto-scaling vision transformers without training," in *ICLR*, 2022.

[90] S. Yu, T. Chen, J. Shen, H. Yuan, J. Tan, S. Yang, J. Liu, and Z. Wang, "Unified visual transformer compression," in *International Conference on Learning Representations*, 2021.

[91] H. Yang, H. Yin, P. Molchanov, H. Li, and J. Kautz, "Nvit: Vision transformer compression and parameter redistribution," *arXiv preprint arXiv:2110.04869*, 2021.

[92] R.-T. Wu, A. Singla, M. R. Jahanshahi, E. Bertino, B. J. Ko, and D. Verma, "Pruning deep convolutional neural networks for efficient edge computing in condition assessment of infrastructures," *Computer-Aided Civil and Infrastructure Engineering*, vol. 34, no. 9, pp. 774–789, 2019.

[93] G. Nadizar, E. Medvet, F. A. Pellegrino, M. Zullich, and S. Nichele, "On the effects of pruning on evolved neural controllers for soft robots," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2021, pp. 1744–1752.

[94] G. Nadizar, E. Medvet, H. H. Ramstad, S. Nichele, F. A. Pellegrino, and M. Zullich, "Merging pruning and neuroevolution: towards robust and efficient controllers for modular soft robots," *The Knowledge Engineering Review*, vol. 37, 2022.

[95] R. Shi, T. Li, and Y. Yamaguchi, "An attribution-based pruning method for real-time mango detection with yolo network," *Computers and Electronics in Agriculture*, vol. 169, p. 105214, 2020.

[96] D. Wu, S. Lv, M. Jiang, and H. Song, "Using channel pruning-based yolo v4 deep learning algorithm for the real-time and accurate detection of apple flowers in natural environments," *Computers and Electronics in Agriculture*, vol. 178, p. 105742, 2020.

[97] S. Chen, R. Zhan, W. Wang, and J. Zhang, "Learning slimming sar ship object detector through network pruning and knowledge distillation," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 1267–1282, 2020.

[98] A. Negi, P. Chauhan, K. Kumar, and R. Rajput, "Face mask detection classifier and model pruning with keras-surgeon," in *2020 5th IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*. IEEE, 2020, pp. 1–6.

[99] C. Gamanayake, L. Jayasinghe, B. K. K. Ng, and C. Yuen, "Cluster pruning: An efficient filter pruning method for edge ai vision applications," *IEEE*

Journal of Selected Topics in Signal Processing, vol. 14, no. 4, pp. 802–816, 2020.

[100] Y. Zhou, G. G. Yen, and Z. Yi, "Evolutionary compression of deep neural networks for biomedical image segmentation," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 8, pp. 2916–2929, 2019.

[101] M. Hajabdollahi, R. Esfandiarpoor, P. Khadivi, S. M. R. Soroushmehr, N. Karimi, and S. Samavi, "Simplification of neural networks for skin lesion image segmentation using color channel pruning," *Computerized Medical Imaging and Graphics*, vol. 82, p. 101729, 2020.

[102] L. Chen, L. Zhao, and C. Y.-C. Chen, "Enhancing adversarial defense for medical image analysis systems with pruning and attention mechanism," *Medical physics*, vol. 48, no. 10, pp. 6198–6212, 2021.

[103] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015.

[104] X. Dong, J. Huang, Y. Yang, and S. Yan, "More is less: A more complicated network with less inference complexity," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017.

[105] 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," *Int. J. Comput. Vis.*, 2015.

[106] G. Kang, J. Li, and D. Tao, "Shakeout: A new approach to regularized deep neural network training," *IEEE T-PAMI*, 2017.

[107] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time

object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015.

[108] X. Dong, S.-I. Yu, X. Weng, S.-E. Wei, Y. Yang, and Y. Sheikh, "Supervision-by-Registration: An unsupervised approach to improve the precision of facial landmark detectors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018.

[109] T. Shen, T. Zhou, G. Long, J. Jiang, and C. Zhang, "Bi-directional block self-attention for fast and memory-efficient sequence modeling," in *Proc. Int. Conf. Learn. Represent.*, 2018.

[110] Y. Yang, D. Xu, F. Nie, S. Yan, and Y. Zhuang, "Image clustering using local discriminant models and global integration," *IEEE T-IP*, 2010.

[111] T. Shen, T. Zhou, G. Long, J. Jiang, S. Pan, and C. Zhang, "Disan: Directional self-attention network for rnn/cnn-free language understanding," in *AAAI*, 2018.

[112] X. Dong, D. Meng, F. Ma, and Y. Yang, "A dual-network progressive approach to weakly supervised object detection," in *ACM Multimedia*, 2017.

[113] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Eur. Conf. Comput. Vis.*, 2016.

[114] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *BMVC*, 2016.

[115] 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 *Proc. Adv. Neural Inf. Process. Syst. Workshop*, 2017.

[116] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process.*

Syst., 2012.

[117] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 1761–1770.

[118] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015.

[119] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.

[120] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[121] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[122] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Proc. Adv. Neural Inf. Process. Syst.*, 1993, pp. 164–171.

[123] R. Reed, "Pruning algorithms-a survey," *IEEE T-NN*, 1993.

[124] L. Zeng and X. Tian, "Accelerating convolutional neural networks by removing interspatial and interkernel redundancies," *IEEE Trans. Cybern.*, vol. 50, no. 2, pp. 452–464, 2018.

[125] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2015.

[126] M. A. Carreira-Perpinán and Y. Idelbayev, ""learning-compression" algorithms for neural net pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018.

[127] P. T. Fletcher, S. Venkatasubramanian, and S. Joshi, "Robust statistics on riemannian manifolds via the geometric median," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2008.

[128] S. Son, S. Nah, and K. Mu Lee, "Clustering convolutional kernels to compress deep neural networks," in *Eur. Conf. Comput. Vis.*, 2018.

[129] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS*, 2015.

[130] J. Kim, S. Park, and N. Kwak, "Paraphrasing complex network: Network compression via factor transfer," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018.

[131] Z. Liu, J. Xu, X. Peng, and R. Xiong, "Frequency-domain dynamic pruning for convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018.

[132] F. Tung and G. Mori, "Clip-q: Deep network compression learning by in-parallel pruning-quantization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018.

[133] 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," *Eur. Conf. Comput. Vis.*, 2018.

[134] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4857–4867.

[135] B. W. Silverman, *Density estimation for statistics and data analysis*. Routledge, 2018.

[136] H. H. Chin, A. Madry, G. L. Miller, and R. Peng, "Runtime guarantees for regression problems," in *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*. ACM, 2013, pp. 269–282.

[137] M. B. Cohen, Y. T. Lee, G. Miller, J. Pachocki, and A. Sidford, "Geometric median in nearly linear time," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM, 2016, pp. 9–21.

[138] 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 *Proceedings of the European Conference on Computer Vision (Eur. Conf. Comput. Vis.)*, 2018, pp. 784–800.

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

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

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

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

[143] 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.

[144] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal sgd for pruning very deep convolutional networks with complicated structure," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 4943–4953.

[145] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

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

[147] A. T. Benjamin and J. J. Quinn, *Proofs that really count: the art of combinatorial proof*. MAA, 2003, no. 27.

[148] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[149] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *Proc. Int. Conf. Learn. Represent.*, 2017.

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

[151] X. Dong and Y. Yang, "Network pruning via transformable architecture search," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019.

[152] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. Int. Conf. Learn. Represent.*, 2017.

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

[154] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[155] 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.