Deep Neural Network Compression with Filter Pruning

Shuo Zhang

School of Computing and Communications

Lancaster University

A thesis submitted for the degree of

Doctor of Philosophy

April 2023

I would like to dedicate this thesis to my family

Statement of Originality

I, Shuo Zhang, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in this thesis.

Acknowledgements

First of all, I would like to thank my supervisor, Prof. Jungong Han. He provided me with a great deal of professional and careful academic guidance, which let me know how to become a qualified academic researcher. More importantly, He also gave me patience and encouragement when I suffered difficulties and confusion during my doctoral study. Confidence is extremely crucial for a person who is in adversity but is eager to achieve success. He always believed in me and supported me even when I suffered severe academic setbacks. Without his support, I could not complete my Ph.D. study.

Then I would like to thank my supervisor, Prof. Qiang Ni. He gave me a lot of patience and support when my research progress suffered setbacks and delays. He also provided me with a comfortable and carefree research environment where I could focus on my own research during the COVID-19 pandemic.

I would like to thank Dr. Matthew Broadbent, Dr. Zheng Wang, and Dr. Hossein Rahmani in our school for their kind support and suggestions.

I would like to thank Dr. Gengshen Wu, who recommended me to this wonderful research group and started my doctoral life. During my early studies, he gave me a lot of life assistance and academic advice, which helped me quickly adapt to doctoral study and life.

Then I would like to thank Prof. Hai Li, who shared his own academic experience with me and gave me valuable advice that helped me get rid of confusion and overcome academic difficulties in my early study.

I would like to thank two of my best friends, Dr. Matthew Gingfung Yeung and Mingqi Gao. They gave me a lot of academic guidance and advice and helped me quickly have professional academic quality, which greatly improved my research work efficiency. Besides, they also provided me with a lot of help in my daily life, especially when I was facing adversity

and setbacks. They showed me what a friend is. It's an honor to have you as my best man.

I would like to extend my sincerest thanks to the following people, who have all helped in the completion of this thesis.

I would like to thank the previous and current visiting scholars in our lab, including Prof. Yi Liu, Dr. Yuanjun Huang, and Dr. Shanfeng Wang, for their helpful discussions and research ideas.

I would also like to thank all my wonderful and kind friends from Lancaster University, Warwick University, and Aberystwyth University, including Dr. Xiaowei Gu, Dr. Haiyang Liu, Dr. Yao Zhang, Dr. Binbin Su, Dr. Peng Cheng, Dr. Chengcheng Qu, Yunqi Miao, Changrui Chen, Yu Fu, Jingkun Chen, and Zhuang Shao. Thank them for their friendly help during my Ph.D. study.

Lastly, I want to thank the following three people who are the most important in my life.

I would also like to thank my lovely father, Jingjun Zhang, and mother, Shuhong Wu, who are the greatest parents in the world. They have done their best to provide me with unselfish and warm love since I came into this world. I am fortunate and grateful to be born into such a happy and sweet family. Thanks, Mom and Dad. I love you forever!

I would also like to thank my fiancée, Weihua Xu. You are the most important piece of the puzzle in my life. The gorgeous rainbow only appears after the wind and rain. Thank you for your unwavering support over the five years we have known each other. Although I have nothing now, I am very willing to work hard for you and our future sweet life.

Forgive me for being an inarticulate and unromantic person. Although I have imagined many scenarios in which I would propose to you, as a science and engineering student, I can think of nothing more romantic than to permanently leave my proposal vows in my most valuable doctoral dissertation.

Please allow me to propose to you in this way, asking you to be my wife, to complete my life.

Abstract

The rapid development of convolutional neural networks (CNNs) in computer vision tasks has inspired researchers to apply their potential to embedded or mobile devices. However, it typically requires a large amount of computation and memory footprint, limiting their deployment in those resource-limited systems. Therefore, how to compress complex networks while maintaining competitive performance has become the focus of attention in recent years. On the subject of network compression, filter pruning methods that achieve structured compact model by finding and removing redundant filters, have attracted widespread attention. Inspired by previous dedicated works, this thesis focuses on the way to obtain the compact model while maximizing the retention of the original model performance. In particular, aiming at the limitations of choosing filters on the existing popular pruning methods, several novel filter pruning strategies are proposed to find and remove redundant filters more accurately to reduce the performance loss of the model caused by pruning. For instance, the filter pruning method with an attention mechanism (Chapter 3), data-dependent filter pruning guided by LSTM (Chapter 4), and filter pruning with uniqueness mechanism in the frequency domain (Chapter 5).

This thesis first addresses the filter pruning issue from a global perspective. To this end, we propose a new scheme, termed Pruning Filter with an Attention Mechanism (PFAM). That is, by establishing the dependency/relationship between filters at each layer, we explore the long-term dependence between filters via attention module in order to choose the to-be-pruned filters. Unlike prior approaches that identify the to-be-pruned filters simply based on their intrinsic properties, the less correlated filters are first pruned after the pruning step in the current training epoch and then reconstructed and updated during the subsequent training epoch. Thus, the compressed network model can be achieved without the requirement for a pre-trained model since input data can be manipulated with the maximum information maintained when the original training strategy is executed.

Next, it is noticed that most existing pruning algorithms seek to prune the filter layer by layer. Specifically, they guide filter pruning at each layer by setting a global pruning rate, which indicates that each convolutional layer is treated equally without regard to its depth and width. In this situation, we argue that the convolutional layers in the network also have varying degrees of significance. Besides, we propose that selecting the appropriate layers for pruning is more reasonable since it can result in more complexity reduction with less performance loss by keeping and removing more filters in those critical and nonsignificant layers, respectively. In order to do this, long short-term memory (LSTM) is employed to learn the hierarchical properties of a network and to generalize a global network pruning scheme. On top of that, we present a data-dependent soft pruning strategy named Squeeze-Excitation-Pruning (SEP), which does not physically prune any filters but removes specific kernels involved in calculating forward and backward propagations based on the pruning scheme. Doing so can further decrease the model's performance decline while achieving a deep model compression.

Lastly, we transfer the concept of relationship from the filter level to the feature map level because the feature maps can reflect the comprehensive information of both input data and filters. Hence, we propose Filter Pruning with Uniqueness Mechanism in the Frequency Domain (FPUM) to serve as a guideline for the filter pruning strategy by generating the correlation between feature maps. Specifically, we first transfer features to the frequency domain by Discrete Cosine Transform (DCT). Then, for each feature map, we compute a uniqueness score, which measures its probability of being replaced by others. Doing so allows us to prune the filters corresponding to the low-uniqueness maps without significant performance degradation. In addition, our strategy is more resistant to noise than spatial methods, further enhancing the network's compactness while maintaining performance, as the critical pruning clues are more concentrated following DCT.

List of Publications

Journal papers

1. **S. Zhang**, G. Wu, J. Gu and J. Han, "Pruning convolutional neural networks with an attention mechanism for remote sensing image classification", Electronics, vol. 9, no. 8, pp. 1209, 2020.
2. G. Ding, **S. Zhang**, Z. Jia, J. Zhong and J. Han, "Where to prune: Using LSTM to guide data-dependent soft pruning", IEEE Transactions on Image Processing, vol. 30, pp. 293–304, 2020.
3. **S. Zhang**, M. Gao, Q. Ni and J. Han, "Filter pruning with uniqueness mechanism in the frequency domain for efficient neural networks", Neurocomputing, 2023, 530: 116-124.

Contents

List of Tables

List of Figures

List of Acronyms

ABC	Artifical Bee Colony
ADMM	Alternating Direction Methods of Multipliers
BCN	Binary convolutional neural network
BLAS	Basic Linear Algebra Subprograms
BP	Back propagation
BWD	Biased Weight Decay
CNNs	Convolutional neural networks
CPUs	Central Processing Units
CV	Computer vision
DAL	Discrimination-aware loss
DCT	Discrete cosine transform
DHP	Differentaiable Meta Pruning
DMCP	Differentiable Markov Channel Pruning
DNNs	Deep neural networks
FBO	Forward-backward operations
FB	Filter-based
FC	Fully-connected
FD	Frequency domain
FLOPs	Floating Point Operations
FMB	Feature map-based
FP	Filter pruning

FPUM	Filter Pruning with Uniqueness Mechanism
GLP	Group Lasso penalty
GMP	Geometric median points
GPR	Global pruning rate
GPUs	Graphics Processing Units
HET	Huffman encoding technology
HF	Hash function
HP	Hard pruning
IC	Image classification
KD	Knowledge distillation
LRD	Long-range dependency
LSTM	Long short-term memory
LSVS	Large-scale visual search
MACs	Multiply-and-accumulates
MCN	Modular Convolutional Networks
NAS	Network Architecture Search
NC	Network compression
NCR	Network compression rate
NP	Network pruning
NRM	Network recovery mechanism
NS	Network slimming
NS	Network sparsity
NTL	Network training loss
OBD	Optimal Brain Damage
OBS	Optimal Brain Surgeon
OD	Object detection
PB	Parameter binarization

PFAM	Pruning Filter with an Attention Mechanism
PGA	Policy gradient algorithm
PTM	Pre-trained model
RNN	Recurrent neural network
SAM	Self-attention module
SD	Spatial domain
SEP	Squeeze-Excitation-Pruning
SFP	Soft Filter Pruning
SMC	Sparse matrix calculation
SP	Soft pruning
SR	Structural regularization
SS	Semantic segmentation
TAS	Transformable Architecture Search
TEE	Taylor expansion estimation
TF	Tensor factorization
VA	Video analysis
VR	Video retrieval
VT	Visual tracking
WP	Weight pruning

Chapter 1

Introduction and Background Theory

1.1 Research Background

The fast growth of convolutional neural networks (CNNs) has shown astounding performance and efficiency in various computer vision applications [1–6]. In order to further improve performance, the majority of existing algorithms prioritize designing immensely complicated network structures, resulting in a vast number of parameters and massive model sizes. Reference [7] demonstrates that VGG-16 [8] requires 138M weights and 15.5G multiply-and-accumulates (MACs) to process one 224×224 input image. Moreover, we have to utilize at least 2 GTX1080 graphics processing units (GPUs) to successfully train the VGG-16 with a batch size of 128 since it requires around 16GB of GPU memory. Similarly, the sophisticated and lightweight model Resent-152 [9] also needs 231 MB of memory space and over 6 seconds to process one image to release its powerful performance. Such implementations demand intensive computations and large memory footprints, restricting their further deployment in resource-limited systems such as embedded or mobile devices. Therefore, how to compress complicated networks while preserving competitive performance has attracted extensive attention from industry and academia.

To realize the aforementioned goals, several network compression approaches such as knowledge distillation [10–12], parameter quantization [13–15], tensor decomposition [16–18], and network pruning [19–21] have been proposed. In contrast to the first three branches of research, which construct lightweight networks from scratch or alter the parameter storage types, network pruning methods obtain compactness by selecting and pruning unnecessary parameters from the original networks. This way, the pruned networks can retain the most knowledge from the original networks without consuming as much memory and computation resources as before.

Fig.1.1 depicts a simplified flow chart based on the network pruning method (NP) [22] that helps to further illustrate the network pruning process. This flow chart can be easily extended to other types of pruning methods, such as filter pruning. In NP, the pruning algorithm depends on the parameters from the pre-trained model, requiring the initial training of the model to obtain well-trained model parameters. Next, we can determine which connections are not important and need to be removed based on specific importance criteria. Evaluating the magnitude of weight can be viewed as a straightforward and effective criterion for identifying unimportant connections. After the connection selection stage, the connections with low weight are pruned from the model. However, after the pruning process, the structure of the pruned model becomes extremely sparse due to the elimination of a large number of connections, resulting in a significant loss of performance. Therefore, to reduce the performance loss, the pruned network needs to be trained again in the following stage to get the final weights for the remaining sparse connections. It is worth mentioning that this process can be iterated many times to achieve extreme network compression. After several iterations, we end up with a lightweight network without sacrificing too much performance.

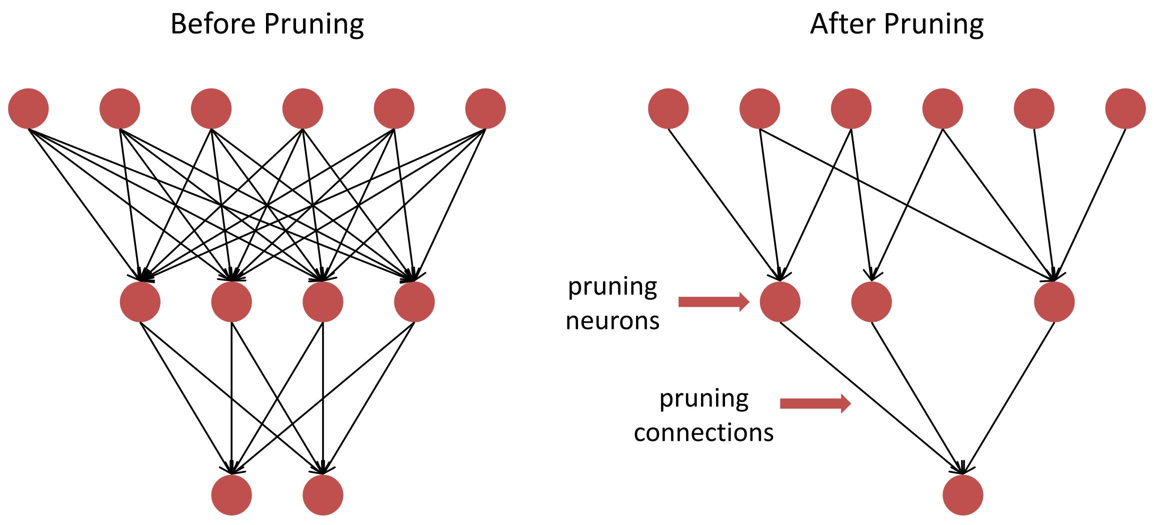


Figure 1.1: Connections and neurons before and after pruning.

The three earliest representative research in the field of network pruning are Biased Weight Decay (BWD) [23], Optimal Brain Damage (OBD) [24], and Optimal Brain

Surgeon (OBS) [25]. In particular, OBD [26] and OBS [25] use the hessian of the loss function rather than the magnitude-based criterion to remove some unimportant connections during the pruning stage. However, such proposed approaches require additional computational costs to accomplish the second-order derivative calculation, which greatly affects the efficiency of pruning. However, the second-order derivative needs additional computation. Later, some classical weight pruning methods [27–29] are proposed to prune unimportant weights instead of connections or neurons. Compared to the previous methods that prune connections, although these methods can reduce the parameters of the network significantly while losing less performance, the pruned network is sparse and unstructured, which cannot fit into the software libraries [30] and hardware architecture [31] to achieve acceleration. Consequently, filter pruning has been developed rapidly since it removes all weights in a filter and then generates a structure-pruned model to make the acceleration.

In this thesis, we will focus on filter pruning for image classification tasks in computer vision. In particular, we propose several new filter pruning algorithms to solve the several major challenges in network pruning, which are shown as filter pruning with an attention mechanism (Chapter 3), data-dependent filter pruning guided by LSTM (Chapter 4), and filter pruning with uniqueness mechanism in the frequency domain (Chapter 5). The proposed methods can generate the structure-pruned model, which could reduce the computation costs and memory usage while maintaining the excellent performance as the original one.

In the following subsections, we first provide further insights into the fundamental theory of network pruning and briefly present the motivations for proposing those novel methods. Then we summarize the contributions of each chapter. Finally, the organization of this thesis is given.

1.2 Formulation of Network Pruning

1.2.1 Expressions of Network Pruning

Given a CNN model with L layers, let $\boldsymbol{W}^l \in \mathbb{R}^{c^l \times c^{l-1} \times h \times w}$ be the filter tensor of the l-th convolutional layer, where h and w, c^l, and c^{l-1} represent the kernel height, kernel width, output channel, and input channel of the filter, respectively. In particular, h and w are set to 1 for the fully-connected layer. To simplify the discussions, only convolutional layers are discussed as instances in the subsequent sections. The goal of network pruning is to select and remove unnecessary parameters from the original network. To visualize the pruning principle in a straightforward way, we can represent it by applying a mask $\boldsymbol{K}^l$ to $\boldsymbol{W}^l$. Notably, $\boldsymbol{K}^l$ is a binary tensor (0 or 1) that indicates the states of network parameters, where 0 means they can be removed, and 1 means they will be kept. Therefore, given a specific pruning rate p, network pruning can be expressed as:

$$\arg\max_{\mathbf{K}^l} \mathcal{R}\left(\mathbf{W}^l \oplus \mathbf{K}^l\right), \text{ s.t. } \left\|K^l\right\|_0 = 1 - p, \tag{1.1}$$

where $\oplus$ denotes the masking operation and $\mathcal{R}(\bullet)$ evaluates the importance of its input. There are several input importance measurements, such as ℓ_1-norm, ℓ_2-norm, etc.

1.2.1.1 Weight Pruning

Fig.1.2 depicts a simplified framework of weight pruning method. The objective of weight pruning is to select and prune individual filter weights. Thus, the weight that needs to be removed could be located anywhere in the parameter matrix of the filter. Consequently, the dimension of pruning mask $\boldsymbol{K}^l$ should coincide with that of $\boldsymbol{W}^l$ at the convolutional layer l. In general, filter pruning formulates the objective function as follows:

$$\arg\max_{\mathbf{K}^l} \sum_{j}^{c^l} \sum_{k}^{c^{l-1}} \sum_{m}^{h^l} \sum_{n}^{w^l} \mathcal{R}\left(\mathbf{W}^l_{j,k,m,n} \cdot \mathbf{K}^l_{j,k,m,n}\right), \text{ s.t. } \left\|K^l\right\|_0 = 1 - p. \tag{1.2}$$

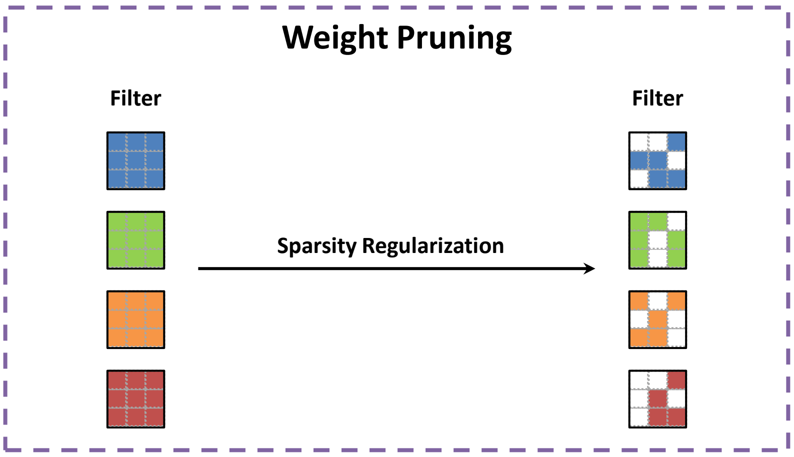


Figure 1.2: Framework of weight pruning using sparsity constraints.

1.2.1.2 Filter Pruning

Instead of pruning individual weights in each filter, filter pruning, which is shown in Fig.1.3 aims to prune the whole filter $\boldsymbol{W}_j^l$, which removes all weights of each filter. Hence, the corresponding pruning mask $\boldsymbol{K}_j^l$ and filter share the same shape $\mathbb{R}^{c^l}$. Filter pruning typically formulates the objective function as follows:

$$\arg\max_{\mathbf{K}^l} \sum_{j=1}^{c^l} \mathcal{R}\left(\mathbf{W}_j^l \cdot \mathbf{K}_j^l\right), \text{ s.t. } \left\|K^l\right\|_0 = 1 - p. \tag{1.3}$$

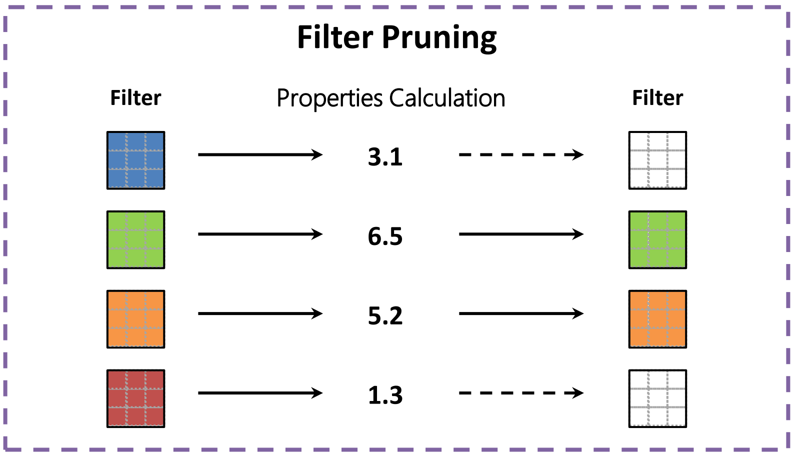


Figure 1.3: Framework of filter pruning with importance criterion. In particular, the smaller norm values (3.1 and 1.3) normally can be treated as unimportant and need to be removed.

In addition, the filter pruning method based on feature maps shows more merits than traditional filter pruning methods for measuring pruning criteria since it embeds more comprehensive information about filters and input data. As a result, it is given more attention as a metric for measuring the importance of filters. Notably, $\boldsymbol{A}_j^l \in \mathbb{R}^{h\times w}$is the j-th feature map from the l-th layer. The objective function can be rewritten as the following form:

$$\arg\max_{\mathbf{K}^l} \sum_{j=1}^{c^l} \mathcal{R}\left(\mathbf{A}_j^l \cdot \mathbf{K}_j^l\right), \text{ s.t. } \left\|K^l\right\|_0 = 1 - p. \tag{1.4}$$

1.2.2 Evaluation Metrics

To pursue maximum performance, most methods are proposed to build sophisticated and heavy network architectures. However, such implementations require intensive computation and memory occupancy due to their number of parameters and complexity, limiting their applications in resource-limited systems (e.g., embedded or mobile devices). Hence, network pruning has gradually become a hot research topic since it aims to generate a lightweight sub-network from the original complicated one with a tiny performance loss.

Compared to large and powerful workstations, the memory space of the embedded systems is designed to be very small to maintain their portability. Moreover, the performance of their processor is also mediocre due to their heat dissipation requirement. Therefore, these two main limitations require the model to have a small size and low computational consumption before being deployed into such resource-constrained systems. To achieve the two main goals above, computation consumption and the number of parameters become the two most important indexes to evaluate the advantages and disadvantages of the network pruning algorithms.

1.2.2.1 Floating Point Operations (FLOPs)

The computation consumption is the computation times required by the model, which reflects the requirements of the model for hardware computing units. It is usually expressed as operations (Ops). Since the most common data format is float32, it is also commonly written as Floating Point Operations (FLOPs). Therefore, FLOPs are one significant factor for network pruning to evaluate the complexity of the network since the pruned model with low FLOPs could decrease the dependency on processor performance. Given one

Given one convolutional layer, the FLOPs are calculated with:

$$\mathrm{FLOPs_{conv}} = [(\mathrm{k_w} * \mathrm{k_h} * \mathrm{c_{in}}) * \mathrm{c_{out}} + \mathrm{c_{out}}] * \mathrm{H} * \mathrm{W}, \tag{1.5}$$

where $\mathrm{c_{out}}$, $\mathrm{c_{in}}$ and $\mathrm{k_w}$ and $\mathrm{k_h}$ are the output channel, input channel, kernel height and kernel width of filter, respectively. H and W are the height and width of input feature map. It can be seen that the number of floating point operations not only depends on the size of the convolution kernel and the number of input and output channels but also depends on the size of the feature map. This metric has been used in the experiments of Chapter 3, Chapter 4 and Chapter 5.

For the fully connected layer, the FLOPs are calculated with:

$$\mathrm{FLOPs_{fc}} = (\mathrm{n_{in}} * \mathrm{n_{out}}) + \mathrm{n_{out}}. \tag{1.6}$$

1.2.2.2 The number of Parameters (Params)

The number of parameters (Params) is a critical factor for evaluating the size of the model. It is the sum of the parameters in the model and directly relates to how much disc space the model needs. For embedded and mobile devices, their memory capacities are small due to their size limitations, which results in a strict limit on the size of model packages. Therefore, network pruning methods need to generate a new sub-network with fewer parameters to meet this requirement. For CNNs, the number of parameters of the model is mainly composed of the weights of the convolutional layers and the fully connected layers.

Given one convolutional layer, the Params are calculated with:

$$\mathrm{param_{conv}} = (\mathrm{k_w} * \mathrm{k_h} * \mathrm{c_{in}}) * \mathrm{c_{out}} + \mathrm{c_{out}} \,, \tag{1.7}$$

where $\mathrm{c_{out}}$, $\mathrm{c_{in}}$ and $\mathrm{k_w}$ and $\mathrm{k_h}$ are the output channel, input channel, kernel height and kernel width of filter, respectively. It can be seen that the number of parameters in the convolution layer is only related to the size of the convolution kernel and the number of input and output channels. This metric has been used in the experiments of Chapter 5.

For the fully connected layer, the Params are calculated with:

$$\mathrm{param_{fc}} = (\mathrm{n_{in}} * \mathrm{n_{out}}) + \mathrm{n_{out}} \,. \tag{1.8}$$

1.3 Research Problems and Challenges

Although existing filter pruning methods have achieved remarkable results in network compression, there are still some limitations in reaching the goal of removing more redundant filters with less performance degradation. Therefore, in this thesis, we mainly analyzed the three potential improvement areas of filter pruning: the relationship between filters, the importance of convolutional layers, and the relationship between feature maps. We elaborate on the challenges and issues of these research points that have not been addressed. These challenges stimulate our research interests and motivate our works.

1.3.1 The Pruning Strategy of Filter

In filter pruning, how to select and remove redundant filters more accurately becomes one of the most important factors that researchers consider when designing pruning criteria. Some works [30, 33, 34] determine and prune unimportant filters according to their intrinsic properties. For example, Li *et al.* [33] proposed to prune parameters at the filter level, which evaluates a filter by calculating the sum of its absolute weights and removes unimportant filters sequentially. Chin *et al.* [34] proposed the filter pruning method to prune unimportant filters according to the magnitude-based ranking after the training stage, and some of them with lower ranks will be pruned. Fig.1.4 shows the traditional filter pruning process.

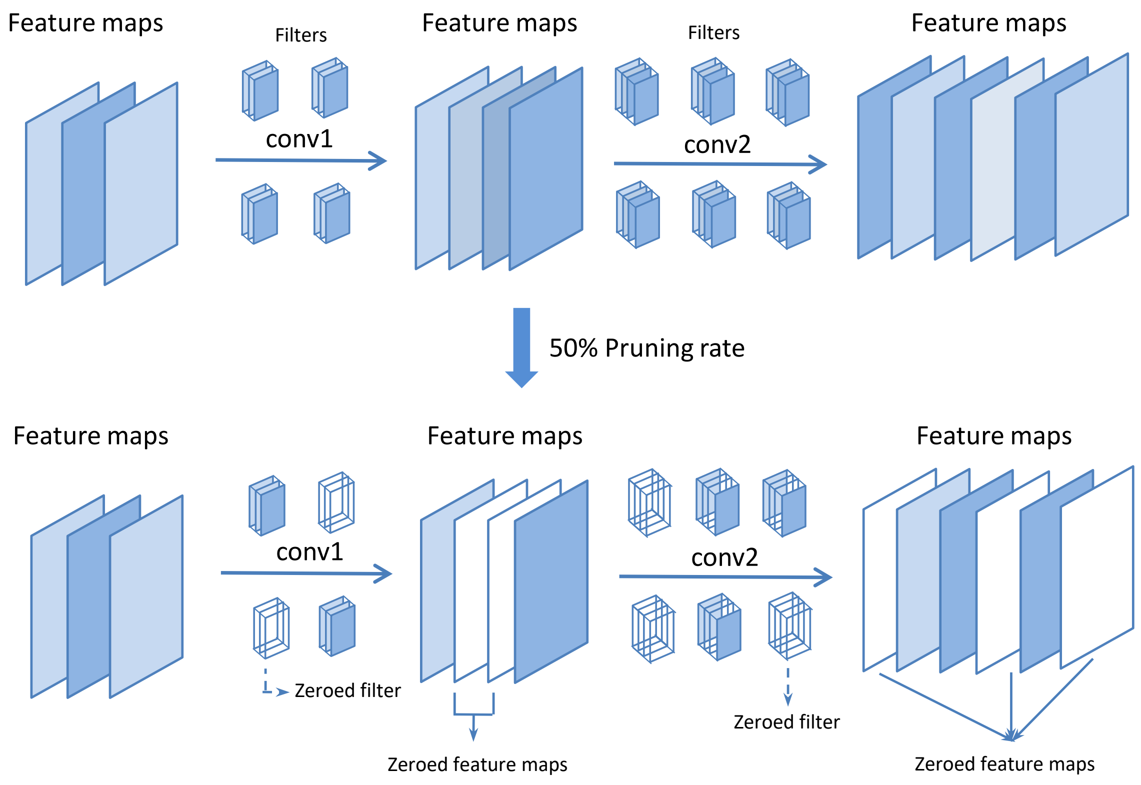


Figure 1.4: A simple example of filter pruning. The CNN comprises two convolutional layers, which originally have four and six filters, respectively. At the 50% pruning rate, the second and third filters in the first layer and the first, fourth, and sixth filters in the second layer are pruned. Therefore, the corresponding output feature maps in the two following layers are eliminated.

Besides, [33,35,36] generally compressed the deep CNNs in a hard manner. They follow the basic three steps to prune redundant filters. However, once the filters are selected, these to-be-pruned filters are abandoned permanently during the pruning process and never recovered again in the following fine-tuning stage. Although the model is dramatically reduced in size due to the removal of filters, such a hard pruning method is more likely to yield unsatisfactory performance due to the shrinkage of model capacity. As a result, these methods may have surprisingly expensive computational resource costs since the fine-tuning stage usually needs plenty of time to re-train the pruned model to recover its performance. Fig.1.5 illustrates the filter pruning process based on properties calculation.

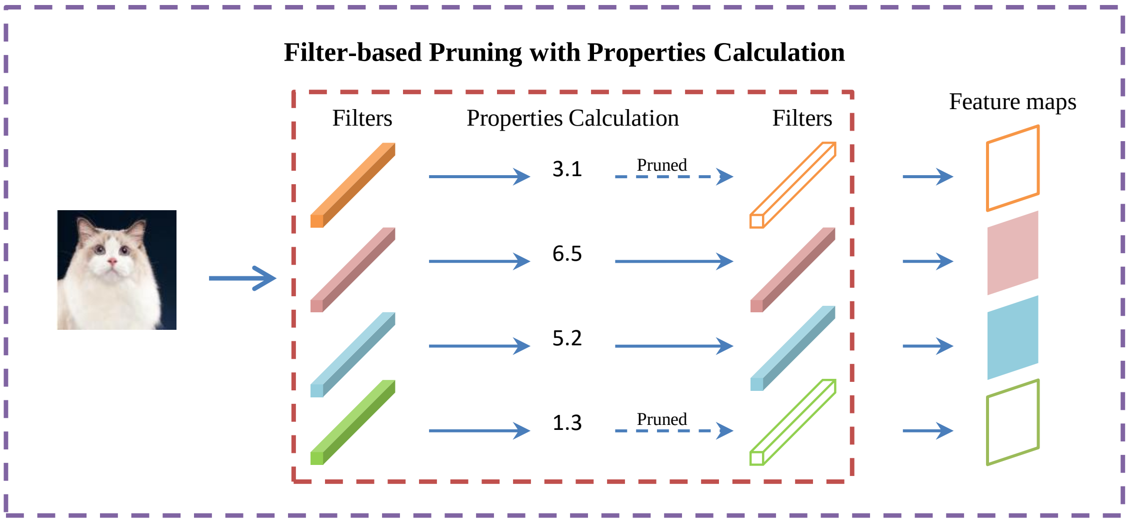


Figure 1.5: The work pipeline of filter pruning process based on properties calculation.

Unlike the hard pruning methods, pruning filters in a soft manner is becoming a popular research topic since the pruned filters from the previous pruning stage can be recovered at the next training epoch to make a new pruning decision. Soft filter pruning [30] dynamically removes redundant filters at each training epoch to avoid some important filters being pruned mistakenly in the early pruning stage. However, similar to previous pruning methods, it still only focused on the importance of individual filters without taking the correlation among filters into consideration, which made the filter selection less discriminative.

He *et al.* [37] argued that the magnitude-based pruning methods that only focus on individual filters make it easier to obtain the sub-model with a locally optimal solution than with a globally optimal solution, thus hindering the model from achieving better performance. Then they proposed a filter pruning method based on correlations among filters rather than their individual properties. To be specific, they aim to find and remove the filters that can be replaced by other filters. They get the replaceable scores by computing the geometric median of the filters, and then they prune the filters with high replaceable scores. Fig.1.6 illustrates the filter pruning process based on correlation calculation.

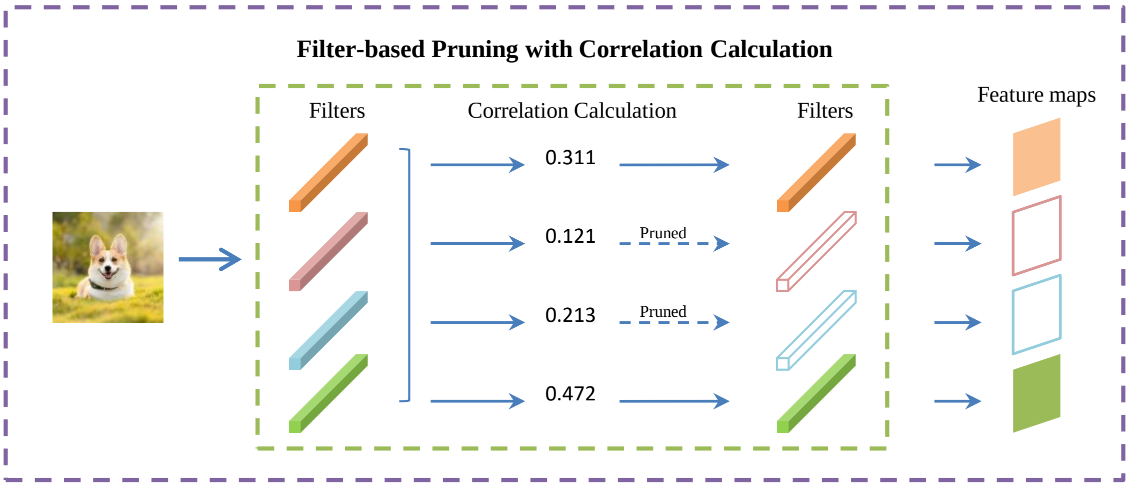


Figure 1.6: The work pipeline of filter pruning process with corrletaion calculation.

However, this method still has some drawbacks. If some filters with large norm values are deemed replaceable and then removed due to the criterion of the geometric median, the actual pruning effect is worse than that of the pruning method that only considers their norm values. Moreover, the correlation proposed by this method only depends on the mathematical distance to the geometric median point and does not consider the interrelation between the filters. Therefore, how to effectively express the correlation of filters is still a challenging topic.

1.3.2 The Importance of Convolutional Layers

As mentioned above, the soft filter pruning approach based on the correlation criterion shows many benefits over traditional filter pruning methods with individual importance criteria. However, previous works still have two major limitations, which impede their performance in filter pruning.

Firstly, most existing filter pruning methods [30,33,36–38] tend to prune filters by setting a global pruning rate. In other words, all convolutional layers are deemed to be of equal importance during the pruning step, and the filters are removed layer-by-layer from top to bottom. However, removing filters in this way contradicts the observation that the layers differ in terms of significance. If it prunes a few filters from a crucial layer, the entire performance of the system may degrade substantially. In contrast, reducing the number of filters in an insignificant layer can drastically reduce the model's complexity while having no effect on its performance. Consequently, pruning filters with only one global pruning rate for all convolutional layers may provide a suboptimal pruned model.

Secondly, a few filter pruning methods empirically [39, 40] set the specific pruning rate for each layer in the pruning stage to improve performance. However, such an empirical setting is modified several times after a large number of experimental results, which undoubtedly increases the time and computational cost significantly. Moreover, it may be a suboptimal pruning decision for the network because enumerating all the parameter combinations is extravagant. In addition, this pruning decision is generated statically and cannot be driven by data, which limits it to prune filters more accurately.

1.3.3 The Relationship between Feature Maps

Due to the excellent acceleration of filter pruning, identifying which filters need to be pruned has become a primary challenge. One intuitive solution is to measure the intrinsic properties of the filter to determine its importance. For example, Fig.1.7(a) calculates the norm value of the filter and then assumes that the filters with low norm values are named unimportant filters and need to be pruned. However, since the importance of filters is measured by their norm values and the correlation between filters is ignored, some filters are still easily replaced by others, which should be treated as redundant even though their norm values are high enough. To solve the above issue, in Fig.1.7(b), some pruning methods [37, 41] are proposed to make the correlation between filters to select redundant filters more accurate than the pruning methods that only focus on the properties of the individual filters. Although it considers the correlation between filters, these filter-based pruning methods still mainly focus on the filters only and ignore the richer and more comprehensive information after the convolutional operation.

Unlike the filter-based pruning methods, Lin *et al.* [39] proposed using the feature map's rank as the filter's importance evaluation. In Fig.1.7(c), compared to filter-guided pruning, feature-guided pruning offers a more accurate guide to determining which filters are redundant because it measures the comprehensive information of both the input data and filters. However, two drawbacks still hinder selecting redundant filters more accurately. First, current methods only calculate the rank based on its intrinsic properties for each feature map, ignoring the correlation between maps. Second, in the spatial domain, the critical information in the feature map is relatively scattered. If their norm or rank is calculated directly, noise information could mess up the calculation, which would affect the selection result of the redundant filters. Therefore, finding redundant filters in the frequency domain seems less efficient.

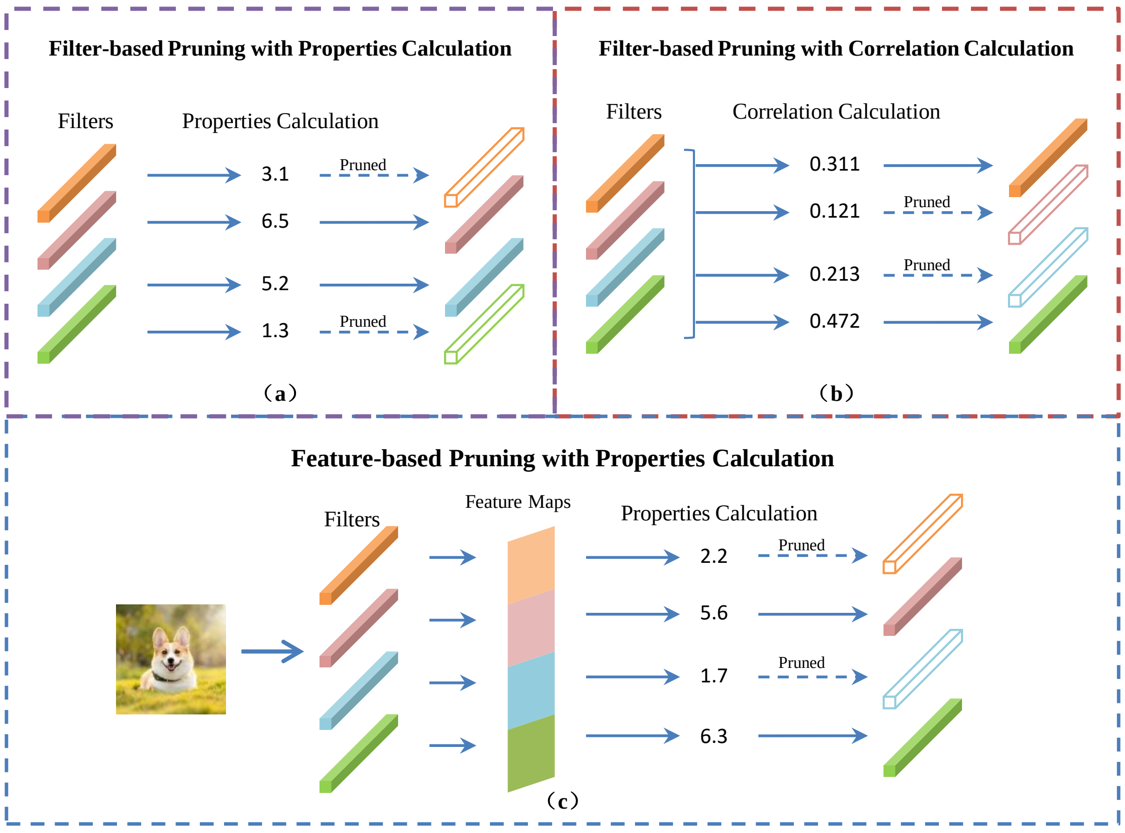


Figure 1.7: A simple example of filter pruning. The CNN comprises two convolutional layers, which originally have four and six filters, respectively. At the 50% pruning rate, the second and third filters in the first layer and the first, fourth, and sixth filters in the second layer are pruned. Therefore, the corresponding output feature maps in the two following layers are eliminated.

In this section, we have concisely discussed the significant research challenges in network pruning. To handle the above problems, several novel solutions are proposed for those problems separately, which will be further detailed in the main chapters. In the following subsections, we briefly summarize the contributions of each work and provide the chapter outlines correspondingly.

1.4 Overview of Contributions

In this thesis, the content covers three main research improvements in filter pruning: the relationship between filters, the importance of convolutional layers, and the relationship between feature maps. The contribution of this thesis can be expressed as proposing novel solutions to tackle the discussed research challenges while achieving superior performance with significantly reduced model size and computational consumption in the above scheme. Without loss of generality, the contributions of each work are briefly summarized as follows:

- **Chapter 3:** To address the challenges above in Section 1.3.1, a novel deep network compression method termed Pruning Filter with an Attention Mechanism (PFAM) is proposed for efficient image classification. The compact network model is obtained by integrating the attention-based filter pruning strategy into a unified end-to-end training process. Besides, a novel correlation-based filter selection criterion is proposed in the filter pruning, where the correlation value of each filter is calculated through the attention module. Then, the less correlated filters are pruned to reduce the network complexity. The proposed attention module efficiently generates the correlation among filters by exploring their long-range dependencies, which is more likely to make wise decisions in selecting the to-be-pruned filters without compromising the network performance.

- **Chapter 4:** The experiment results verified the argumentation that the layers are different in terms of importance, which provide the experimental basis for setting a specific pruning rate in different layers in the filter pruning stage. To this end, an end-to-end framework is proposed to determine the importance of each layer and thus generate the specific pruning decision for each one. The core module LSTM is updated using the policy gradient method with both model performance and complexity as the reward. Besides, we propose the SEP pruning method rather than adopting a hard pruning strategy to prune redundant filters accurately. SEP is a data-dependent soft pruning method, which preserves all the filter parameters, but for each image data, only some important ones participate in calculating forward and backward propagations. Different filters may be softly pruned when the given image is changed according to the SEP selection.

- **Chapter 5:** To tackle the problems in Section 1.3.3, we introduce the concept of uniqueness, a novel criterion for filter pruning. Unlike the intrinsic properties of filters, uniqueness is measured from the correlation between feature maps. It implicitly indicates how uniquely a feature map embeds critical information. Therefore, a more comprehensive pruning strategy can be achieved. In addition, we propose to determine to-be-pruned filters in the frequency domain rather than the spatial domain. With the advantages of frequency-domain operations, our proposed method can find and prune unimportant filters more efficiently without much noise interference as in the spatial domain..

1.4.1 Thesis Outline

The rest of this thesis consists of a comprehensive literature review of the existing works and the proposed state-of-the-art filter pruning frameworks to solve three main issues in network pruning. Firstly, instead of pruning filters based on their individual properties, we propose to select and prune filters by exploring the long-range dependencies among filters via an attention module. Besides, the data-dependent soft pruning guided by LSTM is proposed to let the network provide the specific pruning decision for each layer by itself rather than setting a global pruning rate or empirically setting a specific pruning rate for each layer by the human. Then, we extend the concept of correlation to the feature map domain and propose a novel filter pruning method to choose and remove filters by evaluating the uniqueness of corresponding feature maps in the frequency domain. Finally, we conclude this thesis and discuss potential research directions. The remaining chapters are summarized as follows:

- **Chapter 2: Literature Review on network pruning.** A comprehensive overview of the state-of-the-art network pruning algorithms is given in this chapter, which covers the research works from various domains in the network pruning. Moreover, to address the issues, some research publications related to the proposed algorithm are also mentioned since they give us great inspiration, which helped us solve the issues successfully.

- **Chapter 3: Pruning Convolutional Neural Networks with an Attention Mechanism.** In this chapter, we propose a new scheme, termed Pruning Filter with Attention Mechanism (PFAM), to compress and accelerate traditional CNNs. In particular, a novel correlation-based filter pruning criterion, which explores the long-range dependencies among filters via an attention module, is employed to select the to-be-pruned filters. The extensive experiments involving various network architectures on three public image datasets demonstrate that our proposed method outperforms the state-of-the-art in accuracy and model compression.

- **Chapter 4: Data-Dependent Soft Pruning Guided by LSTM.** In this chapter, we present a novel filter pruning framework to evaluate the importance of each network layer and select the most unimportant layers to prune. Considering the hierarchical structure of CNN, we employ LSTM as an evaluation model to generate pruning decisions. Besides, based on the slimmer architecture generated from LSTM, we further propose the SEP attention mechanism to rebuild the baseline network, which realizes the data-dependent soft pruning. Experiment results show that our pruning method is capable of compressing a variety of network structures with comparable accuracy and works well on both convolutional and fully-connected networks. It also reveals that our method learns the sensitivity of each network layer.

- **Chapter 5: Filter Pruning with Uniqueness Mechanism in the Frequency Domain.** In this chapter, a novel filter pruning method is presented, which operates mainly in the frequency domain and computes uniqueness as the critical criterion for removing filters. We further transform the encoded features into the frequency domain by DCT to mine more valuable and concentrated information from the feature maps. After this, we compute uniqueness scores from each feature map, considering the properties within and across maps. The network pruning is achieved by removing the filters corresponding to the low-uniqueness maps, which can be easily replaced by others. This way, the proposed method can effectively reduce network complexity while maintaining its performance to the largest extent. We evaluated our method with various network architectures on two different scales of datasets. The experimental results showed that our method achieves superior performance compared to the state-of-the-art approaches.

- **Chapter 6: Conclusion and Future Work.** Finally, we summarize the contributions of this thesis in this chapter and present future research interests.

Chapter 2

Literature Review on Network Compression

This chapter provides a comprehensive overview of previous network compression methods, including tensor decomposition, knowledge distillation, quantization techniques and network pruning. Although tensor decomposition, knowledge distillation, and quantization techniques are not the main research directions in this thesis, they all belong to diverse sub-research areas of model compression. Moreover, some of the proposed methods in their research areas inspired the works in my thesis. Therefore, this thesis concludes with the above three sub-topics (Sections 2.1, 2.2, and 2.3) of network compression in this chapter.

In Section 2.4, there are two main branches of network pruning: weight pruning and filter pruning. Notably, weight pruning methods are presented in Section 2.41 since they were proposed and extended in the earliest field of network pruning. Their underlying idea provided heuristic assistance for subsequent research on filter pruning. In addition, to address the three main issues in this thesis, some research areas related to the proposed algorithm are also mentioned to ensure the integrity of the work. Therefore, attention mechanisms, network architecture search, and frequency information utilization are presented as subsections in Sections 2.5, 2.6, and 2.7.

2.1 Tensor Decomposition

In CNNs, convolution operations are typically operated in the form of the multiplication of tensors (matrices). In addition, as the structure of CNNs grows increasingly sophisticated, their network weight matrices become enormous. As a result, huge amounts of storage and computational resources are required for optimal performance. The low-rank approximation technique can reconstruct the large weight matrix by combining several low-rank matrices. Hence, the number of network parameters can be reduced to save on the consumption of storage and computing resources. The framework of the classical low-rank tensor decomposition method (Tucker decomposition) is shown in Fig.2.1.

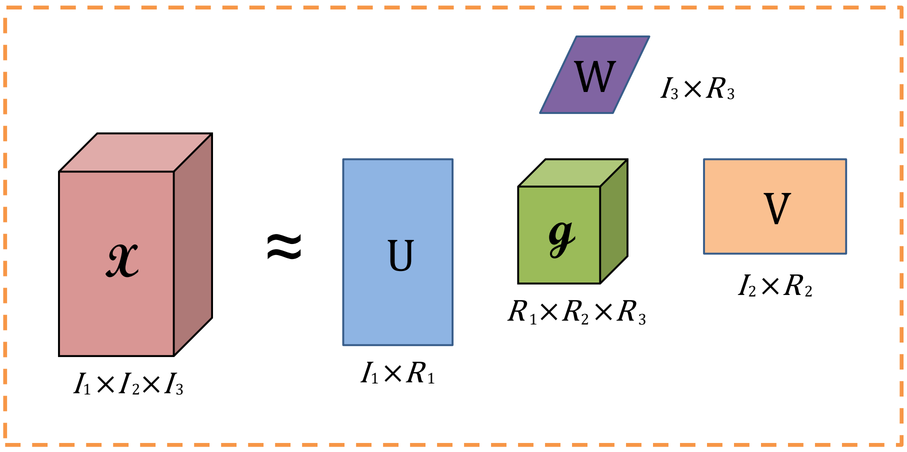


Figure 2.1: The framework of Tucker decomposition. It can be decomposed as a tensor into a set of factor matrices (U, V, W) and one small core tensor (g).

Denton *et al.* [42] proposed a tensor decomposition method to achieve network compression by finding a suitable low-rank approximation to compress each convolution layer. For this purpose, they explored a variety of tensor decomposition methods, such as the singular value decomposition method for two-dimensional tensor decomposition and clustering low-rank decomposition. This method achieves more than twice the acceleration on convolutional layers with only about 1% performance loss. Besides, Jaderberg *et al.* [43] combined two different tensor decomposition strategies to accelerate the network. They first approximated the original filter matrix by linear combinations of fewer filter matrices based on tensor decomposition and then create a low rank of filters with rank-1 to make the further speedup.

Lebedev *et al.* [44] proposed a network compression method that accelerates networks by using tensor decomposition. In particular, they leveraged CP decomposition to decompose the 4D kernel tensor into several small 2D kernel tensors. Then the fine-tuning stage was required to recover the performance loss. Inspired by [43], Tai et al. [45] presented a new low-rank decomposition method to achieve compact networks more efficiently. They calculated exact global optimal solutions rather than approximate local solutions obtained by iterative methods, which are tested effectively for various deep convolutional networks on the large-scale dataset.

Zhang *et al.* [46] proposed a low-rank decomposition method to accelerate nonlinear convolutional neural networks. To be specific, it is based on minimizing the reconstruction error of the nonlinear response and using the low-rank constraint to reduce the computation. To solve the constrained optimization issue, they decomposed it into two feasible subproblems and solve them iteratively. Then, they minimized the asymmetric reconstruction errors, which effectively reduces the cumulative errors of multiple approximation layers. As the extension of previous work, Zhang *et al.* [47] achieved excellent acceleration on the deep convolutional network, proving the effectiveness of their method. Kim *et al.* [48] first leveraged Tucker decomposition to decompose the tensors and then estimate their rank using variational Bayesian matrix factorization. Since the optimization goal of each layer is to minimize the reconstruction error of the parameter tensor, the fine-tuning stage is required to ensure network performance.

The CPD-based methods [49,50] were proposed to compress the networks. Astrid *et al.* [49] combined the Canonical Polyadic Decomposition (CPD) and Tensor Power Method (TPM) to decompose the entire convolutional layer successfully. Phan et al. [50] made the combination of Tucker-2 (TKD) and Canonical Polyadic Decomposition (CPD) to achieve a compact network while maintaining network performance. Yin *et al.* [51] proposed a model compression method based on tensor decomposition to compress the network. Specifically, they leveraged the alternating direction method of multipliers (ADMM) to train the network, which gradually acquires low-tensor rank characteristics. After that, the well-trained network was decomposed in a tensor train (TT) way, and finally, a compact network was obtained after the fine-tuning stage.

Although the tensor decomposition methods can decompose the large weight tensor into two or more small weight tensors to reduce the number of parameters in the network, they still have some weaknesses that limit their development in network compression. For example, the operation of low-rank decomposition often consumes a lot of computing resources, which makes its computing cost very expensive. Second, since different convolution layers contain various information, existing methods typically choose to conduct the low-rank approximation layer by layer as opposed to performing the global parameter compression. Last, the fine-tuning stage is necessary for these methods, which requires a large amount of time to restore performance.

2.2 Knowledge Distillation

Knowledge distillation is a network compression technique in which the knowledge from a well-trained network (teacher model) is transferred into a small or compact one (the student model). Therefore, its goal is to achieve similar performance to the original model with fewer network parameters. The basic framework of knowledge distillation (KD) is shown in Fig.2.2. The most representative work of knowledge distillation for network compression in the earliest years is proposed by Hinton et al. [52]. Generally, knowledge distillation consists of three parts: the distilled knowledge derived from the teacher network, the knowledge distillation algorithm, and the distillation process between the teacher and student networks.

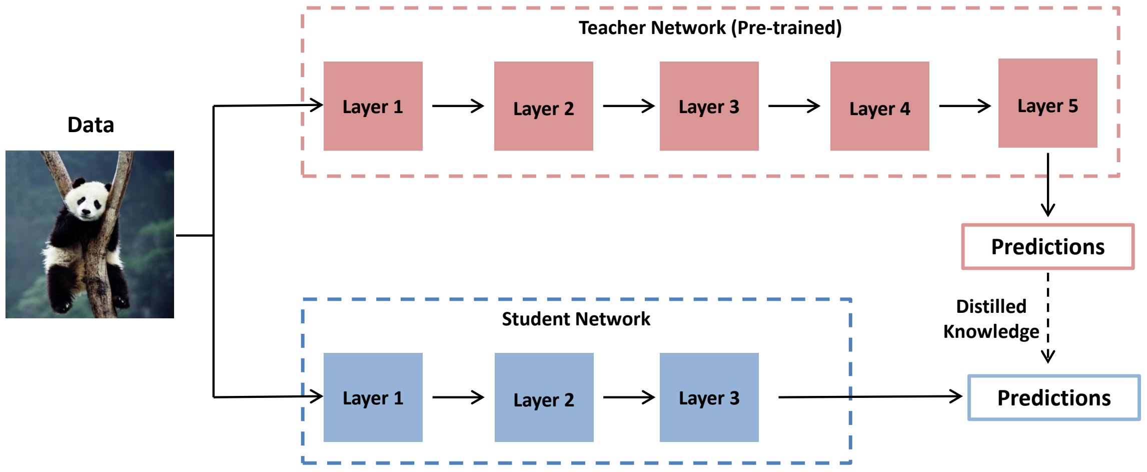


Figure 2.2: The work pipeline of knowledge distillation.

Currently, the knowledge distillation process can be mainly divided into two categories according to different training types. The first category of knowledge distillation aims to distill the knowledge from the teacher network in an offline manner. To be specific, [53–57] proposed offline distillation methods by setting up two stages. In the first stage, they fully trained a powerful original model in the pre-training stage and use it as the teacher model. Afterward, the well-trained weight parameters from the teacher model were frozen and then used to train the small student model through the designed distillation algorithm. Instead of training teacher and student networks separately, the second category of knowledge distillation tends to accomplish the knowledge distillation in an online manner. [58–60] trained both the teacher and student networks to reduce the knowledge distillation time compared to the offline methods. Besides, the original large model is more like a peer than a teacher, and they supervise and influence each other to achieve better learning effects during the training process.

For distilled knowledge, it can be obtained from the teacher network in two ways: the logits-based method and the feature-based method. For example, several logits-based works [1, 52, 61–64] were proposed to train the student network by utilizing the softened outputs of the teacher network. Hinton *et al.* [52] utilized the softened labels to learn the class distribution and then transfered the knowledge to the student network. Instead of statically predefining a one-way transition path between teacher and student networks, Zhang *et al.* [61] proposed deep mutual learning (DML) to allow a set of student networks to learn from and mentor each other during training.

Cho *et al.* [62] argued that the mismatch of model capacity results in the student network not being able to learn well from the powerful teacher network and then proposed an early-stop teacher regularization to solve this issue in the knowledge distillation process. Xie *et al.* [1] trained the student network by using a larger noise dataset. Yang *et al.* [63] achieved optimization by adding constraints to the distillation process. Another series of works distill knowledge from the features of the teacher network. Romero *et al.* [65] first learned knowledge through intermediate feature layers and treated them as a hint to guide the learning process of the student network. Zagoruyko *et al.* [66] trained the student network by extracting the attention generated by the teacher network. Doing so allows the student network to learn more flexible feature information to improve the network's performance.

Although a compact student model can be obtained from the powerful teacher model through knowledge distillation techniques, there are still some limitations to achieving the compact model with excellent performance. For example, in the knowledge distillation process, these methods typically require a lot of prior knowledge to accurately determine and extract the "knowledge" from the teacher model, which significantly increases the time cost and complexity of network training. Besides, different dataset scenarios have various requirements for the specific knowledge, thus limiting the generality and adaptability of the learned student network. Last, in the calculation of distillation losses, many hyper-parameters need to be set empirically, which takes additional time.

2.3 Parameter Quantization

Various methods were proposed to binarize the network models to obtain massive reductions in computation resources and memory costs. For example, BinaryConnect [67] constrained the full-precision 100 weights to two certain values (+1 or -1) during the training. As a result, many multiply-accumulate operations can be replaced by simple additions and subtractions. Although the binarized network only uses 1 bit to represent +1 or -1, which achieves a compression ratio of 32x, the large quantitative loss still exists if the network parameters are represented directly with 1-bit rather than 32-bit.

In [14], BinaryNet was proposed as an extension of BinaryConnect, where both weights and activations are binarized. This method uses binary weights and activation in the gradient computations. However, the real-valued gradients of those weights are still accumulated. Kim *et al.* [68] proposed a bitwise network with binary input and output parameters and binary weights. The forward propagation of the network is operated via XNOR and bit counting rather than the multiplication and addition of real values. Although the bitwise neural network achieves an outstanding effect on model compression, its performance is inferior due to the conversion of parameter types.

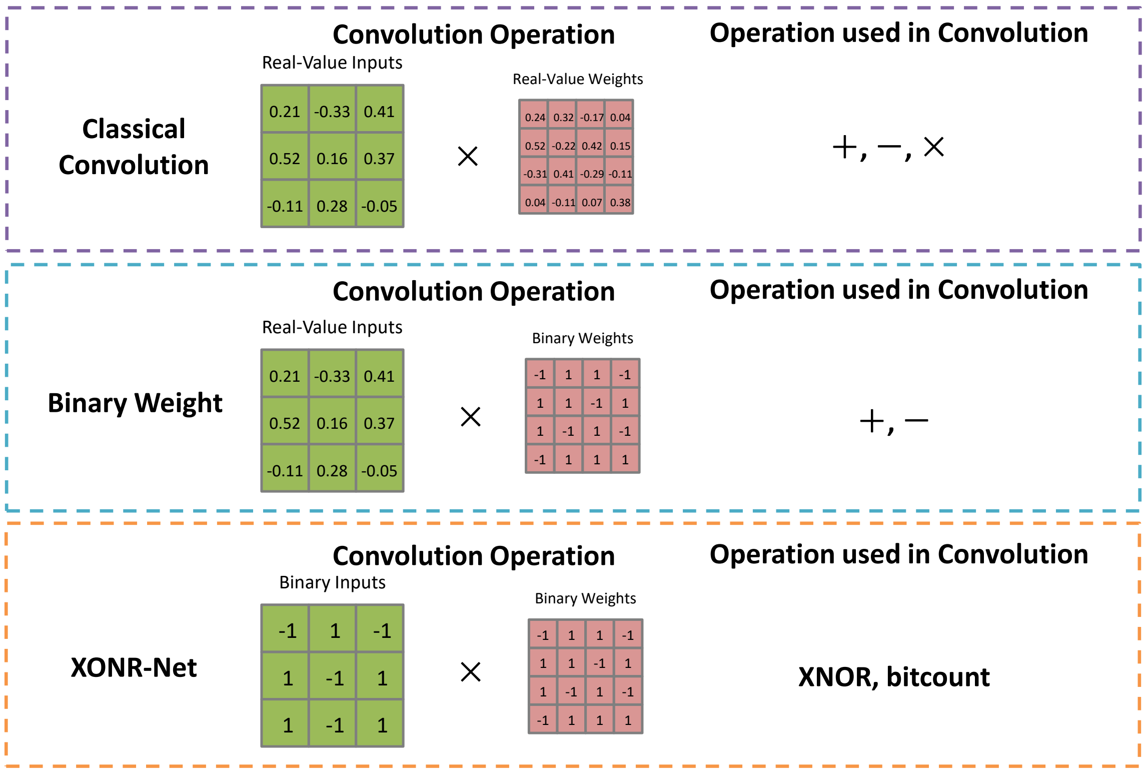


Figure 2.3: Comparison of convolution operation types between the traditional convolution and XNOR-Net.

XNOR-Net [69] was proposed with two efficient sub-binarized networks, namely

Binary Weight Networks (BWN) and XNOR Networks (XNOR), which are shown in Fig.2.3. Particularly, BWN made the combination of a single scaling factor and binary filters, which ends up with 32x memory saving compared to that with real-valued weight values. While for XNOR, they used binary values instead of real values on both the weights and the inputs to the convolution layers, which results in 58x faster convolutional operations compared with full-precision operations and extra 32x memory savings. Despite the fact that XNOR could significantly improve accuracy compared with previous methods, it still has large quantitative losses.

To remedy this situation, [70, 71] aimed to reduce the quantitative losses by setting the scaling parameters. Li *et al.* [70] employed a high-order binarization strategy to convert the initial input data into a sequence of binary inputs with varying orders. Modulated Convolutional Networks (MCNs) [71] was proposed, where M-Filter (a matrix serving as the binarized filter weight) is employed to reconstruct the binarized filters in an end-to-end framework instead of using a scaling factor. Each layer shares only one M-Filter such that the computational costs of the network model can be reduced. Although MCNs obtain better performance than most state-of-the-art methods, it only focuses on the local (i.e., weight) binarization rather than the global (i.e., whole model) binarization.

Lin *et al.* [13] believed that the complex network approximated by the combination of N 1-bit networks has faster computing speed and better performance than the network with N-bits. Therefore, they proposed a method that approximates the full precision network by constructing a linear combination of multiple binary networks. Hu *et al.* [15] transformed the training of binary networks into a hash problem and therefore leveraged the alternate updating strategy to learn hash codes instead of learning binary weights directly.

Wang *et al.* [72] argued that simultaneously quantifying the weight and activation values could seriously harm network performance. Specifically, during SGD training, the slight change in gradient caused by the quantization process may increase the variance of the activation gradient, thus making the training difficult to converge. To solve this issue, they proposed a two-step quantization method to quantify the activation values and weight parameters separately. Mishra *et al.* [73] first widened the activation maps and then divide them into several small parts before the quantization stage to improve the network performace.

Following the previous method [69], Bulat *et al.* [74] aimed to construct a hybrid parameter by merging the activation and weight scaling factors. Besides, they designed several shapes of the hybrid parameter for various application scenarios.

Faraone *et al.* [75] proposed a quantization method that generates specific scaling factors in pixel-wise, row-wise and layer-wise manners to quantify networks efficiently.

Zhou *et al.* [76] proposed an incremental network quantization strategy to compress a full-precision model into a low-precision model with a slight performance loss. Specifically, they first divided each layer of parameters in the pre-trained model into two groups. The parameters in the first group are directly quantized and frozen, and then parameters in another group are retrained to compensate for the accuracy loss caused by quantization. The preceding stages are continued until all model parameters are quantified. [54,77] combined network quantization and knowledge distillation to improve the performance of compressed networks. Xu *et al.* [78] added the network pruning technique to network quantization to achieve higher model compression.

Some works prefer to reduce the number of bits of weights to accelerate the network. For instance, Vanhoucke *et al.* [79] accelerated the network by quantizing the type of parameters as 8-bit. Gong *et al.* [80] and Wu *et al.* [81] used the k-means scalar method to quantize the parameter values. Furthermore, some works [67, 69, 71, 82] binarized the weights and activations in CNNs. They directly quantized the parameter values to 1-bit to achieve extreme model compression. However, these methods change the format of the stored parameters(from 32-bits to only 1-bit), which results in irreversible information loss. Although a series of subsequent approaches have been proposed to compensate as much as possible for the loss of information caused by changes in data formats, their performance is inferior to that of the original model.

2.4 Network Pruning

2.4.1 Weight Pruning

Unlike previous methods that leverage low-rank decomposition, knowledge distillation, or change the type of parameter storage in it, network pruning aims to find and remove redundant parameters to reduce memory and computing resources while achieving similar performance as the original complex network. For example, Based Weight Decay (BWD) [83] was proposed for network pruning to solve the issue of network overfitting in the early time. To be specific, a penalty term is introduced into the normal error function to improve the generalization capabilities of networks [84]. Inspired by this method, they first proposed the concept of weight decay and added it to the back-propagation to decay all weights during the weight update step. Therefore, it can remove some connections and hidden units to reduce the complexity of the network. This work inspires the following network pruning methods that use sparsity constraints.

Besides, Optimal Brain Damage (OBD) [24] and Optimal Brain Surgeon (OBS) [25] reduced the complexity of the network by utilizing the Hessian matrix of the loss function to prune the number of connections. In OBD [24], they believed that parameters with small values have little influence on training errors, and therefore, they aimed to remove the connections that have tiny growth of training error. To reach this goal, they first computed the second derivative for each parameter and then used the second derivative values to calculate the saliencies for each parameter. Finally, parameters with low saliency could be removed safely without much performance degradation. As an extension of OBD [24], OBS [25] analyzed the reason OBD removed the wrong parameters and argued that the Hessian matrix OBD used to remove parameters is actually non-diagonal. Therefore, they calculated the inverse Hessian matrix rather than the Hessian matrix from training data to remove unnecessary parameters more accurately than OBD. However, this method consumes a lot of computing resources when calculating the Hessian matrix, which makes it only applicable to small neural networks.

The superior performance of convolutional neural networks [8, 85, 86] compared to the traditional neural network Multi-Layer Perceptron (MLP) has tremendously benefited the advancement of computer vision. LeNet [85] contains fewer parameters than previous neural networks, yet achieves prominent performance. Although AlexNet [86] and VGG-Net [8] provide stronger performance, the number of model parameters rises rapidly. Consequently, the objective of network pruning is to eliminate the weights of the convolutional neural network rather than the weights of the neural network.

In 2015, Han *et al.* [22] proposed a classical weight pruning method for convolutional neural networks (CNNs) compression. They designed a three-step procedure for eliminating irrelevant connections: pre-training, pruning, and fine-tuning. The L2 norm penalty term is added to the loss function in the pre-training stage to train the network. Thus, the values of some connections will be pushed to zero after training, which can be treated as unimportant. In the pruning phase, a predetermined threshold value is set, and connections below it are removed. The pruned network becomes sparse after the pruning stage, resulting in a significant performance loss. Hence, the fine-tuning stage was required to retrain the pruned network to restore its performance. After several iterations of pruning, a large number of input and output connections were set to zero. Therefore, some neurons linked by those eliminated

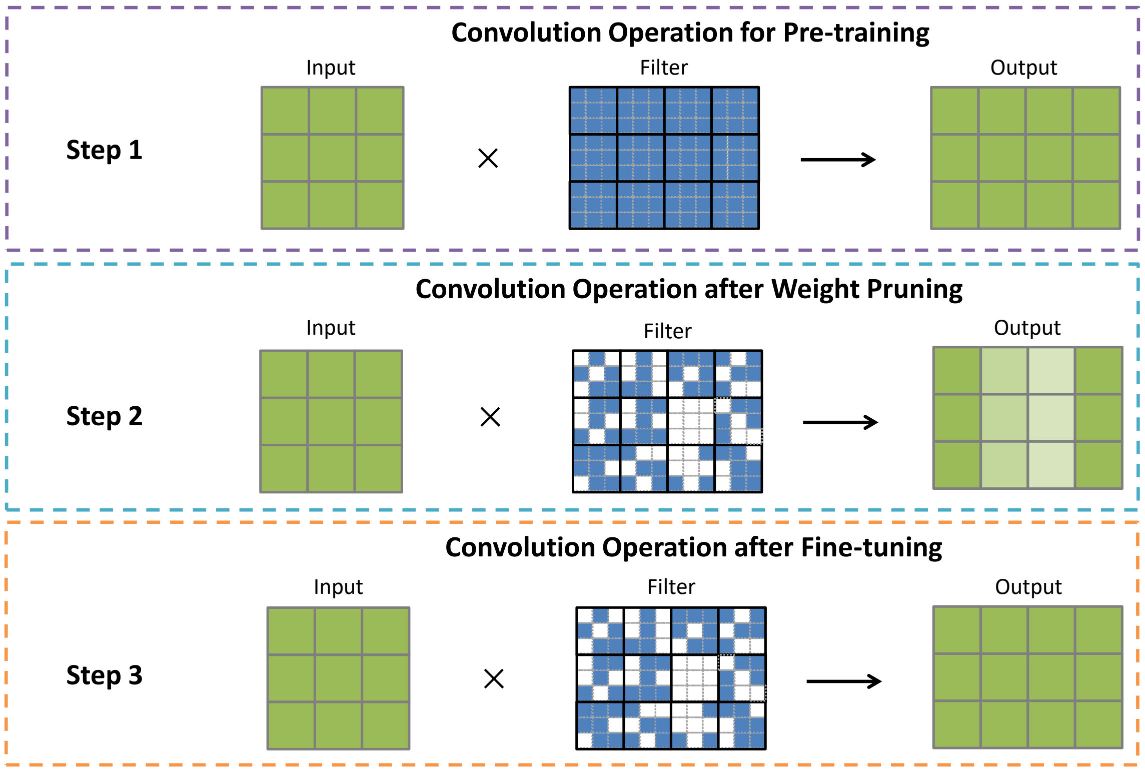

Convolution Operation for Pre-training
Step 1
Input
Filter
Output
Convolution Operation after Weight Pruning
Step 2
Input
Filter
Output
Convolution Operation after Fine-tuning
Step 3
Input
Filter
Output

Figure 2.4: The work pipeline of traditional weight pruning, which contains of three major steps: pre-trained stage, weight pruning stage and fine-tuning stage. The information of the generated feature maps is severely damaged after the pruning process and hence, the fine-tuning stage is needed to recover the representation ability of them.

connections can be safely removed without compromising network performance. The work pipeline of weight pruning is shown in Fig 2.4.

In contrast to the methods that remove unimportant weights, Srinivas *et al.* [87] selected and removed the redundant neurons in a data-free manner. They believed that two similar neurons have similar contributions, which could be merged into one. Therefore, pruning one of them does not affect the final output. Deep Compression [88] achieved considerable energy and memory savings by extending their previous work [16]. In particular, they combined connection pruning with the quantization techniques, where several remaining important connections could share the same weights. Then, Huffman encoding technology was utilized to achieve further compression. Although Deep Compression obtained a high compression ratio on the CNN models by removing unimportant parameters, the parameter importance varied dramatically if the network structure was changed. This implies that this hard pruning method will suffer if the important connections are removed incorrectly during long-term training.

However, Guo *et al.* [89] found that previous method [22] required multiple iterations of pruning and retraining to achieve satisfactory compression rates, which can be extremely time-consuming. In addition, since the connections are pruned permanently, these removed filters will not have a chance to recover after pruning, which may result in a severe loss of accuracy. Therefore, they proposed a weight pruning method named dynamic network surgery to dynamically prune superfluous weights through the cyclical process of pruning and splicing stages. The pruning stage in this method is shown in Fig.2.5. In the pruning stage, redundant connections are eliminated, similarly to prior techniques. However, in the splicing stage, some pruned connections can be recovered and properly spliced into the overall pruning cycle if deemed essential. Park *et al.* [90] argued that previous methods pruned massive parameters at fully connected layers rather than convolutional layers, which requires a lot of computational resources. Although they could achieve excellent parameter reduction, the actual inference speed of the network is tiny. Therefore, they introduced the sparsity constraints to the convolutional layers and proposed guided sparsity learning (GSL) to sparse and remove the weights at the convolutional layers.

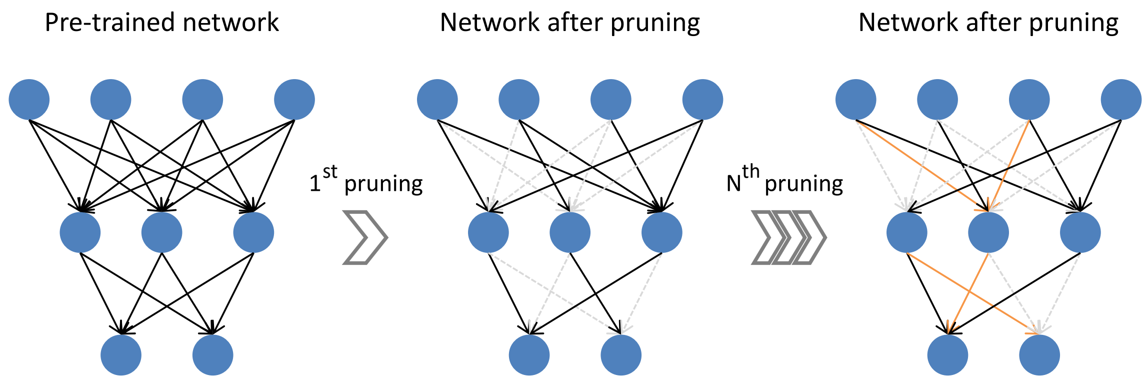


Figure 2.5: The pipeline of dynamic network pruning method. The gray dotted lines show the pruned connections after the pruning stage. The orange line indicates that previously removed lines are restored during subsequent parameter updates.

Different from the methods that reduce the size or computation of the CNNs, Yang *et al.* [91] proposed a new scheme to reduce the energy consumption of the network. This method directly leverages the energy consumption of the network to guide the pruning process. In particular, they estimate the network's energy usage based on the hardware's energy measurements. In addition, redundant weights are determined in the pruning process by minimizing the error of the output feature maps rather than the properties of weights. The redundant weights are first pruned for each layer, and then the fine-tuning stage is required to restore accuracy.

Although OBS [25] has been demonstrated to be effective for shallow neural networks, extending it to deep neural networks remains challenging due to the high computational cost of computing the second derivative (the inverse of the Hessian matrix on all parameters). In this situation, Dong *et al.* [92] proposed a novel strategy for weight pruning to address this issue. They restricted the calculations to the parameters at each layer. Therefore, when calculating the second derivative, the Hessian matrix only aims at the parameters of a specific layer. Doing so can make the calculation easier to solve. In addition, they further reduce the computational complexity of Hessian matrix inversion by using the back-propagation of the fully connected layer in well-trained deep networks. In this way, they successfully deployed the OBS to CNNs.

Xiao *et al.* [93] compressed the model by optimizing a set of trainable and removable weights, which come from the product of the original weights and auxiliary parameters. Ding *et al.* [94] divided all the weights into two subsets through the Taylor expansion estimation. After this, the subsets are trained with different updating rules to achieve more sparse weights. Sanh et al. [95] proposed a first-order weight pruning method, which leverages the movement pruning to make the pre-trained model fine-tuning more adaptive. Lee *et al.* [96] proposed a layer-adaptive and magnitude-based method to approximate the distortion of the pruned network. Net-Trim [97] leveraged sparse constraints to remove unimportant weights. In this method, sparse parameters are learned by minimizing the reconstruction errors of each layer, and then the trained network is pruned layer by layer in a sparse way to obtain a compact network.

Carreira-Perpinán *et al.* [27] introduced constrained optimization to the pruning method. They proposed an alternative optimization strategy consisting of learning and compression stages to remove unimportant weights. Clip-Q [28] combined network pruning and weight quantization to compress the network. Firstly, the redundant weights were selected and removed in the pruning stage. Specifically, they employ the clipping operation to zero out and delete weights defined between two thresholds to obtain a pruned network at each iteration. Finally, the pruned network is further quantified to achieve greater compression.

Similar to the previous method [91] that considers some limitations in the real world, Chen *et al.* [98] proposed a constraint-aware weight pruning method to obtain the compact network that meets some practical requirements of specific hardware. In contrast to prior approaches that pruned weights in the spatial domain, Liu *et al.* [99] proposed a new pruning method in the frequency domain. In particular, they argued that there is spatial redundancy in CNNs since most of the filters tend to be smooth, which is caused by the local smoothness of the image. Consequently, they first express convolution or inner product through DCT domain multiplication. Then, they dynamically prune the DCT coefficients of the network filters using varying pruning rates based on the frequency band's importance.

After comparing the model accuracy of large-sparse and small-dense networks, Zhu *et al.* [20] observed that the performance of large-sparse networks is better than that of small-dense networks. Therefore, they proposed a gradual pruning method, incorporating pruning into the training process to get a large-sparse network. To be specific, they introduced a binary mask matrix to the convolution layer, where "0" signifies pruning and "1" implies preservation. The weights with low absolute values are deemed unimportant, and their corresponding values in the mask matrix are zero. Consequently, these weights will not participate in forward propagation and also will not be updated in back propagation. The model finally reaches the predetermined sparse requirement by removing a portion of the weight with the least absolute value at each iteration. Chen *et al.* [100] employed a low-cost hash function to group weights into hash buckets for parameter sharing. In addition, Ullrich *et al.* [101] proposed a soft weight-sharing method to prune redundant weights. Wen *et al.* [102] reduced network complexity by making the network more sparse. However, the pruned model obtained by weight pruning is unstructured, which cannot fit in the software and hardware architecture to make the acceleration.

2.4.2 Filter Pruning

By observing the acceleration of AlexNet in the actual hardware architectures, Wen *et al.* [102] found that even though the sparsity of unstructured pruning is very high (greater than 95%), the actual acceleration effect of the model after weighted pruning is still minimal. Besides, existing software libraries [30] and hardware architectures [31] usually cannot help accelerate weight pruning methods because of the unstructured sparsity weight matrix generated by weight pruning methods. To address this problem, more works focus on filter-wise pruning. For example, the structured sparsity learning strategy is used for filter pruning to prune the whole filters rather than weights.

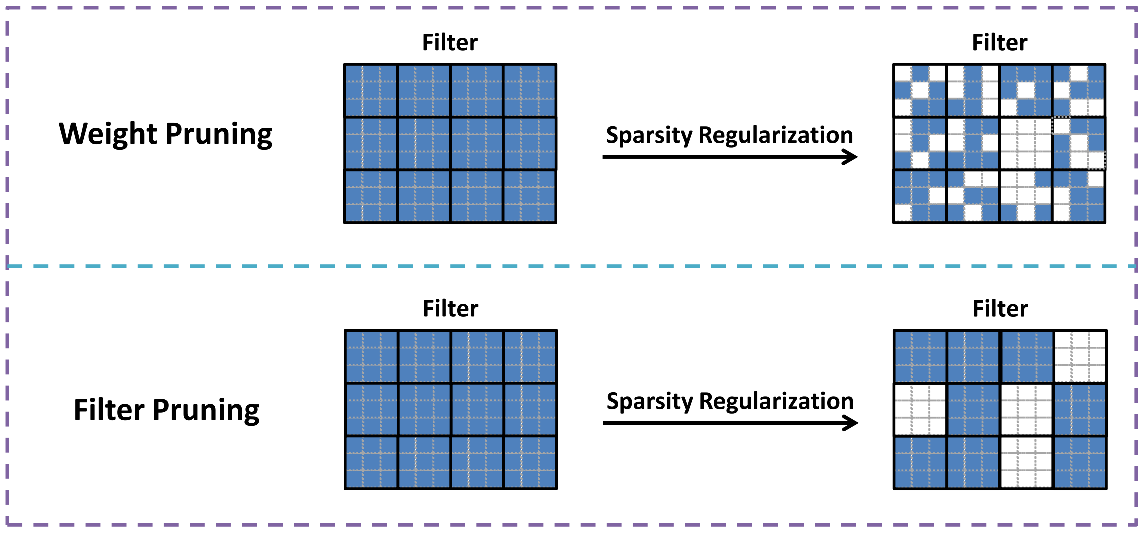


Figure 2.6: Comparison of weight and filter pruning methods based on sparsity constraints.

Since Group Lasso [103, 104] could push a set of parameters close to 0 to generate structured sparsity, doing so can obtain a structured compact model by adding sparsity penalty items to the whole filter, which can achieve significant acceleration in hardware architecture. Fig.2.6 illustrates the difference between weight pruning and filter pruning using sparsity regularization. Liu *et al.* [35] proposed channel pruning focusing on batch normalization layers rather than convolutional ones. To be specific, they employed sparsity regularization to make the values of scaling factors in the batch normalization layer sparser. The fact that the scaling factors in the BN layer are either 0 or very close to zero indicates that the associated convolutional channels are insignificant and could be safely removed. In addition, they established a fine-tuning stage to restore network performance following the pruning stage.

Huang *et al.* [105] argued that the previous pruning approach [35] might be constrained by the structure of the networks and thus lose its effectiveness and versatility since some networks do not have the BN layers. As a result, as a continuation of sparsity-based filter pruning, Therefore, as a continuation of filter pruning that uses sparsity constraints, they proposed a novel filter pruning method that leverages the sparsity regularizations to generate a compact network. Specifically, they added a scaling factor to each output channel of the convolutional layer, making the different output channels more sparse in the learning process by using sparsity regularizations. Finally, the less significant filters were removed based on their contributions, and a compact network was obtained without the need for a step of fine-tuning. Besides, Tartaglione *et al.* [106] analyzed the sensitivity of the final network output to the network weights. They then proposed a network pruning method that incorporates a regularization term to progressively reduce the absolute value of the network weight with low sensitivity. In this method, a large number of weights will be gradually pushed to zero and can be safely removed from the network.

Compared with weight pruning, filter pruning has a coarse-grained pruning effect. Xu *et al.* [107] proposed a hybrid pruning approach by combining the weight pruning and filter pruning to further compress the redundant network. In particular, he first obtained a coarse-grained pruned network through filter pruning, and then achieved a fine-grained compact network through adding the group Lasso regularizer to remove weights smaller than the set threshold during the weight pruning stage. In addition, Ding *et al.* [108] proposed auto-balanced filter pruning to remove redundant filters from the pre-trained network. During the network training phase, an auto-balance regularizer is utilized to enhance the value of some filters while decreasing the value of others to near zero. After several training iterations, it is easy to remove the filters with zero values without significant performance loss.

He *et al.* [109] leveraged LASSO regression to remove redundant filters. To be specific, they added an L1 norm penalty term to the loss function to make filters more sparse after the training stage and then pruned the channels that are zero or close to zero in the pruning stage. Finally, the linear least squares technique was required to restore the performance of the pruned network. Li *et al.* [110] applied the structure regularization to the corresponding out-channels and in-channels in the continuous network layer to achieve a sparse and compact network.

Zhuang *et al.* [111] argued that network slimming [35] employed L1 regularization to put all scaling factors in the BN layer close to zero during sparsity training, resulting in a significant performance loss. Consequently, they presented a novel strategy for channel pruning that leverages polarization regularization to force the updated scaling factors in the direction of polarization during sparsity training. Doing so can make some scaling factors in the BN layer small enough while others still remain large after the sparsity training. As a result, it is simple to determine that some channels with small scaling factors are unimportant and remove them without significantly degrading performance. Wang *et al.* [112] designed a growing L2 regularization for network sparsity. They could use the Hessian information provided by the proposed regularization to prune redundant filters more accurately, resulting in high accuracy while maintaining the same level of compression.

Despite the fact that sparsity-based filter pruning algorithms could generate a structured compact network, such methods normally encounter a dilemma of performance and compression effect during sparse training. In particular, if a strong sparse penalty term is imposed during network training, the generation of a highly compressed pruning network is often accompanied by a severe performance decrease. This is because, under the strong penalty, all the weight parameters are pushed toward zero, resulting in pruned network parameters that are relatively smaller than those of the original network. On the contrary, if the sparsity penalty term is moderate, very few weight parameters will be zeroed out. Therefore, the effect of network compression is greatly reduced even though it can achieve satisfactory network performance. In this case, many filter pruning methods are proposed to remove redundant filters based on their importance while some approaches try to combine other research areas such as network architecture search, tensor decomposition and reinforcement learning to produce compact networks.

Lin *et al.* [113] used Markov decision making processes and reinforcement learning to dynamically prune unimportant filters. Ding *et al.* [114] implemented a Centripetal SGD for network training to drive the values of the updated filters closer together. After the training stage, the filters with similar values were grouped into the same cluster. Each cluster only retains one filter, while others can be pruned safely to compress the network. Liu *et al.* [115] proposed a meta-learning based pruning method to generate optimal sub-networks. Dong *et al.* [116] proposed a network pruning that minimizes the computational costs rather than finding the optimal sub-networks by transformable architecture search (TAS).

[117–120] employed the network architecture search (NAS) technique to the pruning method to explore the optimal compact networks. To be specific, Lin *et al.* [117] proposed a channel pruning method based on an artifical bee colony (ABC) to efficiently find the optimal number of channels in each layer rather than selecting important channels. Guo *et al.* [118] argued that the substructure obtained by network pruning is more important than the inherited weight, and then they proposed differentiable Markov channel pruning to explore optimal sub-structures efficiently. AutoCompress [119] leveraged alternating direction methods of multipliers (ADMM) to achieve the structured optimal compact network.

In addition, DSA [121] and DHP [122] pruned reduntant filters in differentiable way to accelerate networks. [123] combined filter pruning and knowledge distillation technique to compress while [124–126] added the tensor decomposition to filter pruning to accelerate networks. N2N [127] utilized reinforcement learning to learn two separate policies to remove and shrink layers respectively in tiny datasets. PCAS [128] used attention modules as the criterion to evaluate the importance of channels and prune channels with low correlations to accelerate networks. Tang *et al.* [129] proposed a filter pruning method that adds counterpart features to reduce the interference of various irrelevant factors in the pruning process and improve the reliability of the pruning results. Chen *et al.* [130] proposed a one-shot pruning method to compress network without the requirement of fine-tuning stage.

Li *et al.* [131] proposed an adaptive BN technology for filter pruning to solve the network performance degradation caused by the mismatching of BN layer parameters druing the pruning process. [132–134] compressed the network by using reinforcement learning technique. Pahwa *et al.* [132] trained the reinforcement learning agent to predict actions like whether to remove one layer in the network and uses a reward function to update their agent. Chen *et al.* [133] proposed a dynamic channel pruning, which leverages deep reinforcement learning techniques to search for optimal pruning schemes based on the input data. He *et al.* [134] leveraged deep reinforcement learning techniques to determine the specific pruning rate for each layer.Zhuang *et al.* [135] assessed the channels by introducing additional discrimination-aware losses to increase the discriminative power of the intermediate layers.

Li *et al.* [33] proposed to prune parameters at the filter level, which evaluates a filter by calculating its absolute weights sum and removes unimportant filters sequentially. Molchanov *et al.* [36] presented a filter pruning method that determines the importance of the filter by the change of loss function before and after pruning. Moreover, they provided a pruning criterion based on Taylor expansion to approximate the change of loss function caused by pruning network parameters to improve pruning efficiency. ThiNet [38] considered filter pruning to be an optimization problem and proposes a greedy method to prune filters using their statistics information in the next layer.

GDP [21] empirically removed the unimportant filters from the entire network through the global discriminative function during the pruning stage. Then they leveraged the recovery mechanism to recover the pruned filter with high saliency. Finally, the fine-tuning stage was required to restore network accuracy. Ding *et al.* [136] presented the filter pruning that evaluates the importance of filters by the degree of change of feature maps in the next convolutional layer rather than that of the loss from the final output. Inspired by SENet [137], Gao *et al.* [138] proposed a channel pruning method that evaluates the importance of channels by generated feature maps. To be specific, they first extracted and subsample feature maps to generate the saliency of channels. Then some channels with low saliency are zeroed out while others with high saliency are enhanced.

Molchanov *et al.* [139] proposed the filter pruning approach that estimates the importance of filters by Taylor expansions rather than their intrinsic properties. You *et al.* [140] suggested a method for filter pruning that adds a gate decorator to each channel and removes channels whose corresponding gates are zero. Liebenwein et al. [141] proposed a data-driven pruning method to remove redundant filters. The saliency of filters can be determined by a small portion of the input dataset. The filters with high saliency can be deemed important and need to be retained in the pruning stage. He *et al.* [142] argued that it was inappropriate for all layers to share the same pruning criteria during the pruning process. Consequently, they employed the differentiable network architecture search (NAS) technique to explore the suitable pruning criteria for each layer, resulting in a more accurate filter pruning effect.

In contrast to the prior importance-based pruning methods, Gao *et al.* [40] introduced a data-driven pruning strategy to dynamically prune unimportant channels. They added a discrete gate on each channel and remove some channels that close for the given dataset. Chin *et al.* [34] proposed the filter pruning method to prune unimportant filters in a global view. To be specific, all filters in the entire network are ranked globally according to the magnitude-based ranking after the training stage, and some of them with lower ranks will be pruned. DropNet [143] reduced network complexity by iteratively removing filters with the lowest mean values after processing all training samples.

Tang *et al.* [144] argued that the redundant filters depend on the input data. Therefore, they proposed filter pruning to dynamically prune redundant filters according to the given dataset. Wang *et al.* [145] believed that pruning filters in the most redundant layer can achieve better than removing the least important filters in all layers. As a result, they presented a filter pruning method that estimates the redundancy score for each layer and removes filters in the layers with high redundancy scores.

Unlike previous research works that utilized hard filter pruning technology, He *et al.* [30] proposed the soft pruning method, where the pruned convolution filters are recovered and involved in the next training iteration rather than being deleted once and gone permanently. Therefore, there is no need to go through the fine-tuning process on the pre-trained model to compensate for the accuracy drop after the pruning. Since the importance of filters in this method was measured by their norm value but failed to consider the correlations between them, it was still easy to get a sub-optimal pruned model after pruning.

2.5 Attention Mechanism

Several approaches were proposed to obtain excellent network performance in many applications. For example, Mnih *et al.* [146] utilized an attention module in the network training that pays more attention to the local areas with high-correlated weights from the whole target areas, which simulates the human's visual attention behavior when observing images. Following previous works [147–149], they adopted a self-attention module that calculates the response at a specific position as a weighted sum of the features at all positions [150–153]. In this case, the weights or attention vectors were calculated with low computational costs, while a good balance between the long-range dependency modeling ability and computational efficiency was achieved. As a result, the attention mechanism is applied to generate the correlations between filters to better determine and choose redundant filters.

2.6 Network Architecture Search

In order to reduce labor costs, automatic design/search of the network structure by machine has received worldwide attention in both academia and industry [154–159]. For instance, evolutionary techniques [154–156] discovered target models from trivial initial architectures by setting up the vast search space, which requires enormous computing resources. Alternatively, [157, 158] utilized the reinforcement learning mechanism to train the recurrent neural network (RNN) controller to generate the neural networks automatically, which has achieved good results on various datasets. To reduce the search space, Nasnet [159] searched for an architectural building block on a small dataset and then transfered it to a larger one.

2.7 Frequency Information Utilization

Frequency information has recently attracted attention in the computer vision community due to its unique data representation. For example, Gueguen *et al.* [160] incorporated frequency information into the decoding stage to accelerate the model training. Besides, Ehrlich and Davis [161] ran ResNet in the frequency domain to improve the inference speed. Xu *et al.* [162] employed DCT transformation to replace the original spatial sub-sampling approach to better preserve the image information in the pre-processing stage. In addition to better performance, these works reveal that low-frequency feature channels are usually more informative than high-frequency ones in visual reasoning tasks.

The following works further extend the advantages of the frequency information to multiple applications. To be specific, Qin *et al.* [163] proposed channel attention based on the features in the frequency domain and achieve excellent performance in classification, detection, and segmentation tasks. Jiang *et al.* [164] developed a frequency-domain loss for image reconstruction and synthesis, optimized by weighting different frequencies. Cai *et al.* [165] used frequency information to enhance the image generation process. For model compression, Chen *et al.* [166] first observed that the trained weights are usually low-frequency and they prune the filters containing high-frequency components using hashing techniques. Differently, Liu *et al.* [99] pruned filters dynamically by converting convolution operations into DCT multiplications. Although the above methods employ frequency to refine pruning, they ignore the properties and correlations of filters and corresponding feature maps in the frequency domain. However, this work finds they are valuable for pruning and bring superior performance to our proposed method.

2.8 Chapter Summary

In this chapter, we have reviewed the research background of network compression technology and its four hot research areas. More importantly, we have reviewed weight pruning and filter pruning, which are the two most important research branches in network pruning technology. Besides, in dealing with the challenge, several works in other fields gave us great inspiration, which helped us solve the issues successfully. Our works in this thesis are greatly inspired by the related works discussed in previous sections. In the main chapters, we further analyze some of these works and use them as state-of-the-art baselines for our experiments.

In the following chapters, we focus on detailing our proposed methods regarding different challenges in filter pruning and testing the system performance on public datasets. In particular, Chapter 3 presents a novel soft filter pruning method to select and prune redundant filters based on their correlations rather than their intrinsic properties. This work is inspired by the idea of an attention mechanism, which can effectively capture the long-range dependencies between layers. In Chapter 4, we combine the LSTM and attention module to propose a new data-dependent soft pruning strategy that can generate the specific pruning strategy for each layer and prune filters guided by input data. In Chapter 5, we view the pruning strategy with the vision of feature maps rather than the perspective of filters. Besides, we improve our pruning efficiency by introducing the feature map from the spatial domain to the frequency domain.

Chapter 3

Pruning Convolutional Neural Networks with an Attention Mechanism

3.1 Introduction

In recent years, Convolutional Neural Networks (CNNs) have shown appealing performance on various computer vision tasks [5, 167–171]. However, large amounts of computational resources from high-performance GPUs are required to run the complicated CNNs. Moreover, traditional CNNs usually contain many network parameters, even millions, which indicates that embedded equipment suffers from high demands for large memory storage space. Most embedded devices, such as traffic cameras, prefer to conduct real-time data collection and analysis on themselves rather than deciding on a workshop. In this context, the computing power in existing embedded devices is quite limited due to the embedded low-level Central Processing Units (CPUs) and GPUs, thus making it infeasible to deploy deep learning techniques on those machines directly. Consequently, the aforementioned research issues inspired us to develop a lightweight CNN model for embedded systems, which significantly reduces the required hardware resources, such as memory costs and FLOPs (Floating-Point Operations), so that real-time images can be processed in the smart sensor without sending them back to the data center for additional processing.

Many approaches have been proposed for deep network compression and acceleration, where some mainstream categories are discussed: network quantization, compact block, and filter pruning. In network quantization, deep networks [172, 173] were compressed by employing binarized weights and activations to reduce memory space while obtaining relatively good performance. This contradicts the conclusion of prior work [174] that a model with high binarization may achieve inferior performance at an early time. After that, BinaryConnect [67] constrained the full precision weights of the neural network filters to the discrete values (+1 or -1) during propagations. BinaryNet [82] extended the work of BinaryConnect by binarizing both weights and activations. However, arbitrary binarization significantly impairs the capacity of deep neural networks to represent features, resulting in subpar performance. XNOR-Net [69] employed a single scaling factor with binary filters to decrease quantization errors due to binarization, whereas Modular Convolutional Networks (MCN) [71] combined a real-valued matrix with binary filters to reconstruct unbinarized filters. When conducting image classification tasks, both methods compress the models while retaining a high degree of accuracy. Beyond this, several studies are devoted to deploying compact blocks (e.g., convolutional filters with small receptive fields) in deep network structures to reduce computational costs while avoiding excessive quantization errors that diminish the expressiveness of the original network. For instance, Network in Network [175] employed 1×1 convolution kernels to reduce network parameters. ResNet [9] reduced a significant number of network parameters by incorporating residue modules. In addition, ShuffleNet [176] proposed point-wise group convolution and channel shuffle to build an efficient network structure that can operate on mobile devices with limited Computational resources. MobileNet [177] utilized depth-wise and point-wise convolution instead of normal convolution to construct light, deep neural networks. However, a small receptive field focuses unduly on local details without taking global information into account, hence impairing classification performance.

Apart from these above methods, some prior works adopted filter pruning (i.e., channel pruning or network slimming), which is also the focus of this work, to compress the deep network. The core idea behind filter pruning is that the small-valued (i.e., unimportant) activation and connection can be pruned during the iterative training processes to obtain more compact and efficient models [22,25,26,83,178–180]. For example, in [180], they compressed the deep neural networks by simply discarding unnecessary connections that were less than the default threshold. However, it was still required to retrain the sparse network model to compensate for the accuracy decline caused by the pruning. Instead of merely discarding the network parameters, recent works [33, 35, 38, 109, 110, 181, 182] compressed the complicated deep models via pruning the less important filters, thus reducing the computation costs dramatically due to fewer feature maps involved in the subsequent calculations. Subsequently, the work in [33] proposed a filter pruning method to remove the filters with the smallest absolute values and their corresponding feature maps in the following convolutional process for a compact network model. Nevertheless, most of the previous filter pruning works compressed the deep CNNs based on the pre-trained models. They removed those filters permanently and then fine-tuned the pruned models to recover the huge accuracy drop, which was computationally expensive and inefficient. He *et al.* [30] adopted a soft pruning method to dynamically remove redundant filters, where the significance of a filter was evaluated by calculating the norm value of each filter and making a comparison among them. However, they only focused on the importance of individual filters without considering the correlation/dependency among filters, which made the filter selection less discriminative. Moreover, they evaluated the filter significance based on shallow strategies like thresholding, absolute, or norm values, which affected the classification performance considerably because of mistaken pruning.

In this work, we propose a novel filter pruning method, termed Pruning Filter with Attention Mechanism (PFAM), which integrates the attention module with softly pruning the less correlated filters to obtain a compact deep network model for image classification. Specifically, by employing the attention module, target filters with lower correlation values than others are removed at the current pruning stage. The pruned filters from the previous pruning step are retrieved and updated in the subsequent training epoch to prevent accuracy loss due to the pruning process. Through such iterative training steps, a compact deep network model with satisfactory performance can be generated. The contributions of our work are illustrated in the following two aspects:

- A novel deep network compression method termed Pruning Filter with Attention Mechanism (PFAM) is proposed for efficient remote sensing image classification. The compact network model is obtained by integrating the attention-based filter pruning strategy into a unified end-to-end training process.
- A novel correlation-based filter selection criterion is proposed in the filter pruning, where the correlation value of each filter is calculated through the attention module, and then, the less correlated filters are pruned to reduce the network complexity. By using the proposed attention module, it models the correlation among filters efficiently via exploring their long-range dependencies, which is more likely to make wise decisions in selecting the to-be-pruned filters without compromising the network performance.

The rest of this work is organized as follows. In Section 3.2, the proposed method is elaborated along with the comprehensive analysis. Extensive experimental results are provided and analyzed in Section 3.3. Finally, the conclusion is given in Section 3.4.

3.2 Methodology

3.2.1 Motivation

As stated previously, early filtering pruning work [33, 35, 38] typically compresses deep CNNs in a hard manner. To implement pruning strategies, these algorithms require pre-trained models. Specifically, they first load the pre-trained model, then directly evaluate the significance of each filter by computing its norm value from the loaded model parameters, and afterward eliminate the filters in each convolutional layer deemed unnecessary. The pruned model must undergo a fine-tuning stage to compensate for the severe performance loss. However, since there is only one chance to keep or remove a filter, the important evaluation of a single filter is frequently not appropriately judged. Once the filter has been selected, these to-be-pruned filters are permanently deleted throughout the pruning process and are never recovered during the subsequent fine-tuning phase. Although the size of the model is drastically decreased when the filters are removed, this type of hard pruning method is more likely to result in inadequate performance due to the diminished model capacity. In addition, it is important to note that these algorithms will require expensive computational resources and training time during the fine-tuning step, as they have to spend a substantial amount of additional time on the training data to regain a portion of the performance.

In contrast to hard pruning approaches, Reference [30] proposes dynamically removing filters in a soft way by individually computing the norm value of each filter. Although it employs the soft pruning mode to eliminate dependence on the fine-tuning stage and thereby reduce model training time, this pruning mechanism, which only evaluates the individual performance of a single filter in a single convolutional layer without considering their global relationship, is still considered suboptimal. In other words, a filter that is deemed unimportant based solely on the calculation of its norm is not necessarily unimportant from a global perspective of all filters. From another perspective, it might be useful if we investigate the long-range dependency of filters and involve these dedicated dependencies/relationships among filters in the to-be-pruned filter selection. In traditional CNNs, however, the long-range dependency among layers can only be obtained by repeatedly backpropagating through the stacking convolutional layers. That is to say, the current deep networks are usually inefficient at capturing such long-range dependencies, and it is difficult to operate locally when the information needs to be passed back and forth between relatively distant locations [183]. To tackle the above problems, the attention mechanism was applied in many applications to obtain better network performance. For example, Mnih *et al.* [146] utilized an attention module in the network training that pays more attention to the local areas with high-correlated weights from the whole target areas, which simulates the human's visual attention behavior when observing images. Following previous works [147–149], they adopted a self-attention module that calculates the response at a specific position as a weighted sum of the features at all positions [150–153]. In this case, the weights or attention vectors were calculated with low computational costs, while a good balance between the long-range dependency modeling ability and computational efficiency was achieved. Inspired by the above approach, if the long-distance dependence relationship in CNN is integrated into the filter level, the established dependence/relationship between filters can be leveraged to enhance the information transmission efficiency between them, allowing filters at each layer to generate correlation effectively.

Motivated by prior research, we present a novel approach for filter pruning called Filter Pruning with Attention Mechanism (PFAM), which incorporates an attention mechanism into the filter pruning. To be specific, the attention module is employed to enumerate and collect the correlation value of each filter. Then it selects the least correlated filters based on these values; consequently, the overall correlations among all filters in one convolutional layer are considered to achieve minimal accuracy loss globally. In the pruning stage, the values of those deemed to be less correlated are set to zero, indicating that these filters are removed. In the subsequent training epoch, the values of pruned filters are recovered from zero to non-zero and then updated by forward-backward operations. Doing so allows the training data to be processed by the original training strategy without compromising performance. At the same time, the compressed network model can be obtained in the end with no need for the pre-trained model. Fig.3.1 demonstrates the general process of the proposed filter pruning method.

3.2.2 Filter Selection with Attention-Based Correlation

Fig.3.2 demonstrates the workflows of the proposed attention module for computing the correlation values between filters in one convolutional layer, which shares a similar structure to many computer vision tasks. In particular, these filters are treated as feature maps since flattening a filter into a one-dimensional vector is similar to

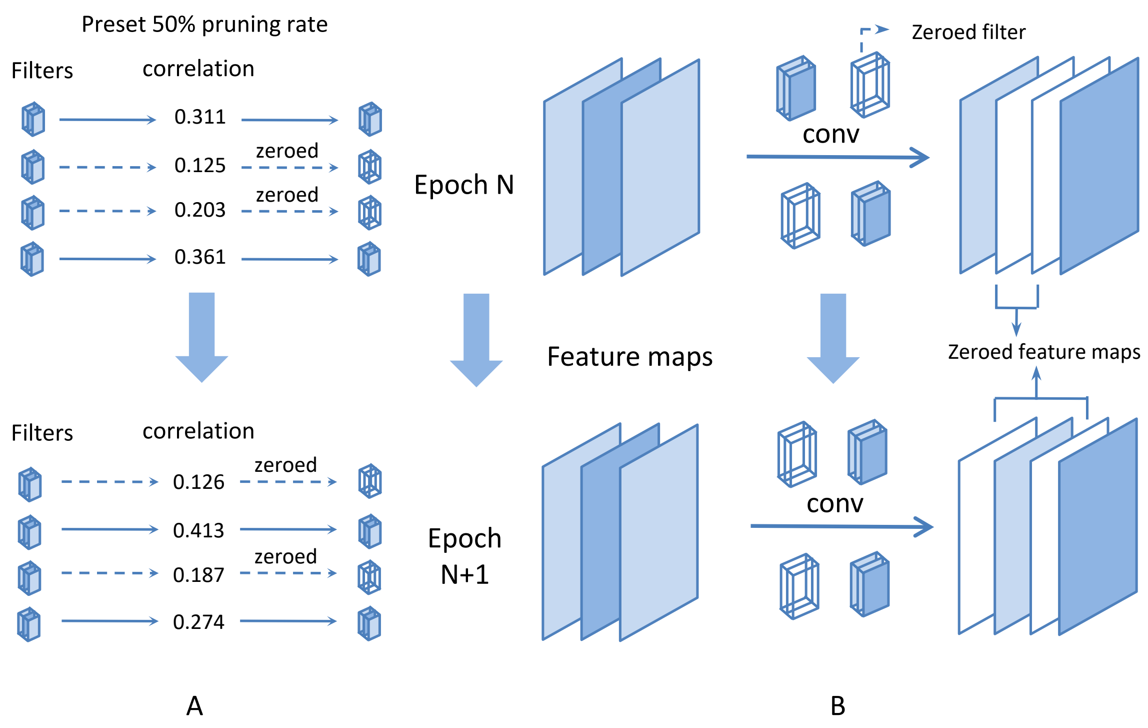


Figure 3.1: The overview of PFAM. A: Filter selection based on the correlation criterion. The filters with less correlation values will be pruned based on the pre-set pruning ratio. B: An example of filter pruning in convolutional layers. The pruned filters are allowed to be updated to non-zero during each training epoch prior to the next pruning stage to maintain the model capacity.

flattening a feature map into a one-dimensional vector, but the vector length is different. In our work, as opposed to finding the most attractive feature maps as was done in prior research, our target is to choose the least correlated filters from a certain number of candidates and prune these filters to build a compact model.

Without loss of generality, we first define some mathematical symbols following [30] to ease the explanation. To be specific, the dimension of filter tensor with $k \times k$ filter size in the i-th convolutional layer is defined as $W^{(i)} \in R^{C_{i+1} \times C_i \times k \times k}$, where $1 \leq i \leq L$. C_{i+1} and C_i mean the number of output and input channels separately for the i-th convolutional layer and L is the number of layers. Then all the filters are flattened in the i-th layer as $W^{(i)} \in R^{C_{i+1} \times V}$, where V denotes the shape of each filter in the i-th convolutional layer and equals to $C_i \times k \times k$. The filters in the i-th layer are first transformed into two weight spaces $F(w) = W_f^{(i)}$ and $G(w) = W_g^{(i)}$ to calculate the attention map, which can be formulated as below:

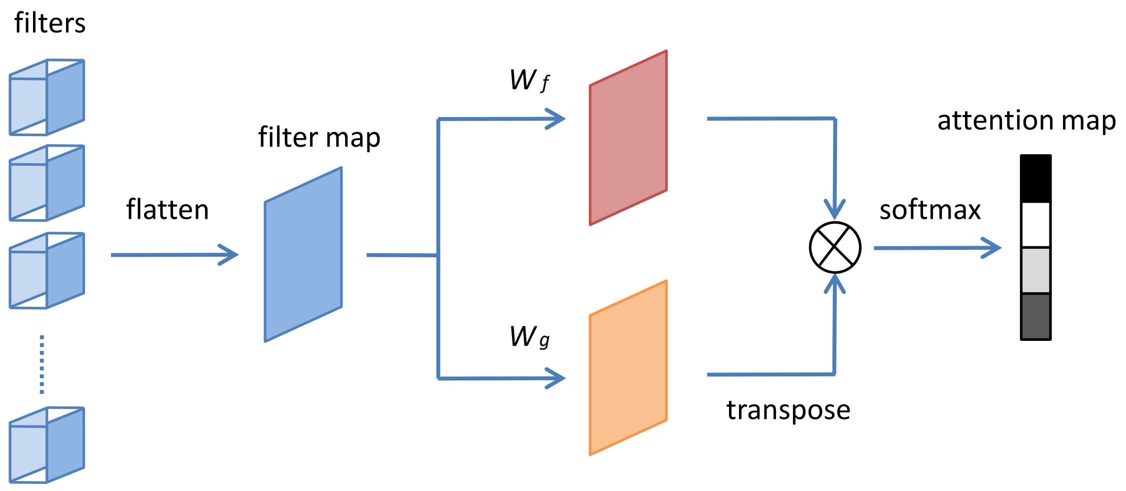


Figure 3.2: The proposed attention module for the PFAM. $\otimes$ denotes the matrix multiplication. The softmax operation is performed on each row.

$$\theta_{j,k} = \frac{e^{s_{jk}}}{\sum_{k=1}^{M} e^{s_{jk}}}, \quad \text{where} \quad s_{jk} = F(w_j)G(w_k)^T. \tag{3.1}$$

$F(w_j)$ and $G(w_k)$ represent the values of j-th filter in $W_f^{(i)}$ and k-th filter in $W_g^{(i)}$ weight spaces, respectively. $\theta_{j,k}$ represents the correlative extent between the j-th and k-th filter. M is the number of filters in i-th convolutional layer. Therefore, the output of the attention value is $\partial = (\partial_1, \partial_2, ...\partial_k, ...\partial_M) \in R^{1\times M}$, where

$$\partial_k = \sum\nolimits_{j=1}^{M} \theta_{j,k}. \tag{3.2}$$

As discussed above, the attention module is used to evaluate the correlation of each filter based on Eq.(3.2) in the pruning stage. The filters with smaller attention values can be pruned, because it turns out that they have less impact on the network performance, as opposed to other high-correlated filters.

3.2.3 Filter Pruning and Reconstruction

In the pruning stage, all the candidate convolution layers with the same pruning rate $P_i = P$ are pruned at the same time, which saves a large amount of computations, compared to the hard pruning methods. Particularly, a pruning rate P_i is set to select a total number of $C_{i+1}P_i$ less-correlated filters in the i-th convolution layer [30, 37]. After pruning the filters in each convolution layer, existing methods always require extra training to converge the network [178, 179]. During the training process, these selected filters are first zeroed out that means they have no contribution to the network output in the current pruning stage.

In most filter pruning methods, however, the pruned filters and their associated feature maps are removed permanently during the pruning process, which could affect the performance significantly. To deal with this problem, these pruning methods usually are conducted based on the pretrained model and they also need to spend extra finetuning time to make accuracy compensation. To get rid of the heavy dependences on the pretrained model and the time-consuming finetune process, we follow the same reconstruction strategy as [181] at this stage, where the pruned filters in the previous pruning process will be reconstructed during one epoch of retraining. To be specific, these pruned filter values are updated from the zero to non-zero after the backpropagation [30,37]. By doing so, the pruned model still has the same capacity as the original model during the training process. More importantly, each of those filters can still contribute to the final prediction. As a result, we can train our network from either scratch or the pretrained model and obtain competitive results even without the need for the finetuning stage.

3.2.4 Compact Network Creation

In Fig.3.3, it illustrates the flowchart of the proposed PFAM, in which iterative training is continued until the accuracy loss converges following several training epochs. When the model converges, a sparse model with several zeroes filters can be obtained. Since the iteration has concluded, these selected filters will remain unchanged. Given that each filter corresponds to a single feature map, the feature maps corresponding to these zeroes filters will always be zero during the inference operation. Removing these zeroed filters and their corresponding feature maps will have no impact. Following the iteration of the preceding processes, the compact model is formed. The entire procedure is explained concisely in Algorithm 1.

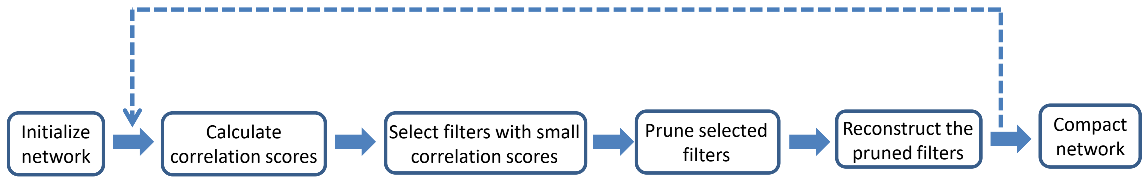

Initialize network
Calculate correlation scores
Select filters with small correlation scores
Prune selected filters
Reconstruct the pruned filters
Compact network

Figure 3.3: The flow chart of the proposed filter pruning. The dotted line means the iterative training scheme.

Algorithm 1 Algorithm Description of PFAM

```
Input: Training data $X$; Pruning rate $P_i$; Training epoch number $epoch$;
Output: The compact model $W^*$ from $W$;
 1: Randomly initialize the network parameters $W = \{W^{(i)}, 0 \leq i \leq L\}$;
 2: for $epoch = 1$; $epoch \leq epoch_{max}$; $epoch++$ do
 3:     Update the model parameter $W$ based on data $X$;
 4:     for $i = 1$; $i \leq L$; $i++$ do
 5:         Obtain the attention values of filters based on Equation 3.2;
 6:         Find $N_{i+1}P_i$ filters with lowest attention values;
 7:         Zeroize selected filters;
 8:     end for
 9: end for
10: return the compact model $W^*$ from $W$.
```

3.3 Experiments and Analysis

In this section, we provide extensive experimental results and analysis to illustrate the network performance of our algorithm on three general image classification benchmarks: CIFAR-10, CIFAR-100 [184] and ImageNet [185] and three remote sensing image classification benchmarks: NWPU-RESISC45 [186], AID [187] and RSSCN7 [188].

3.3.1 Benchmark Datasets

3.3.1.1 General Image Datasets

Two CIFAR datasets contain 50000 training images and 10000 test images. On CIFAR, all images are cropped randomly into 32×32 with four paddings during the training process. Horizontal flip is also adopted. In ImageNet, there are over 1.28 million training images and 50K validation images of 1,000 classes.

3.3.1.2 Remote Sensing Image Datasets

In this section, we provide extensive experimental results and analysis to illustrate the system performance of our algorithm, which outperforms the state-of-the-art methods dramatically on three popular remote sensing benchmarks: NWPU-RESISC45 [186], AID [187] and RSSCN7 [188].

NWPU-RESISC45[1] [186] is a popular public dataset for the remote sensing image scene classification, which is extracted from Google Earth by experts in Northwestern Polytechnical University (NWPU). This dataset is made up of a total of $31,500$

[1]http://www.escience.cn/people/JunweiHan/NWPU-RESISC45.html

images, which are categorized into 45 scene classes as shown in Figure 3.4. Each class includes 700 images with a size of 256×256 pixels in the RGB color space.

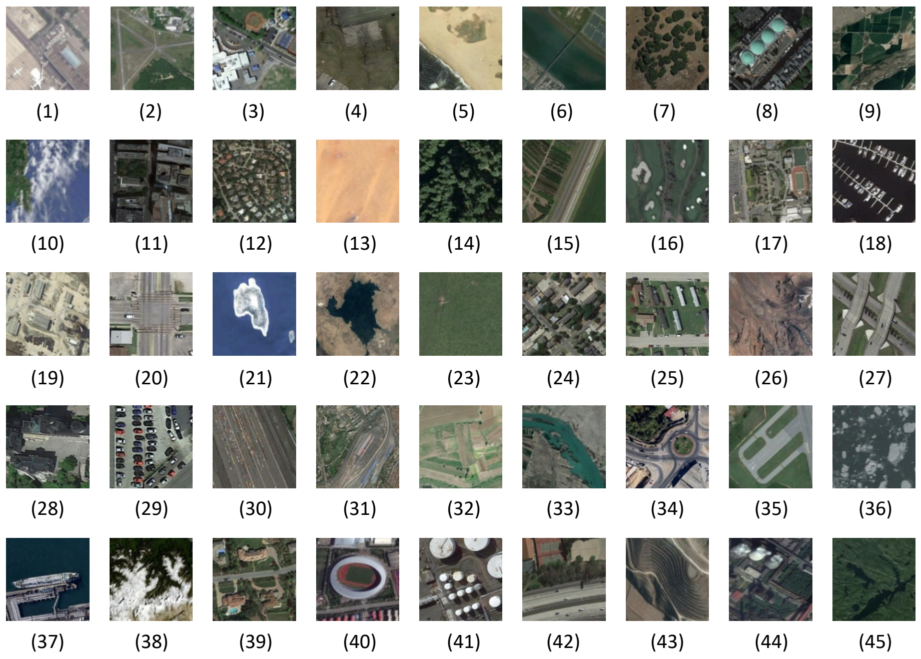


Figure 3.4: Example images of the NWPU-RESISC45 dataset: (1) airplane; (2) airport; (3) baseball diamond; (4) basketball court; (5) beach; (6) bridge; (7) chaparral; (8) church; (9) circular farmland; (10) cloud; (11) commercial area; (12) dense residential; (13) desert; (14) forest; (15) freeway; (16) golf course; (17) ground track field; (18) harbor; (19) industrial area; (20) intersection; (21) island; (22) lake; (23) meadow; (24) medium residential; (25) mobile home park; (26) mountain; (27) overpass; (28) palace; (29) parking lot; (30) railway; (31) railway station; (32) rectangular farmland; (33) river; (34) roundabout; (35) runway; (36) sea ice; (37) ship; (38) snow berg; (39) sparse residential; (40) stadium; (41) storage tank; (42) tennis court; (43) terrace; (44) thermal power station; (45) wetland.

AID[2] [187] is large-scale aerial image dataset, which is selected from Google Earth imagery. This dataset contains in total $10,000$ images with a fixed size of 600×600 pixels within 30 classes as shown in Figure 3.5. Comparing to other classic datasets, the number of images for each category is not equal and different scene types range from 220 to 420, which makes it more challenging in the image classification. Although images in this dataset are acquired at different times with different imaging conditions, some classes are quite similar and therefore make the differences between classes smaller.

[2]https://captain-whu.github.io/AID/

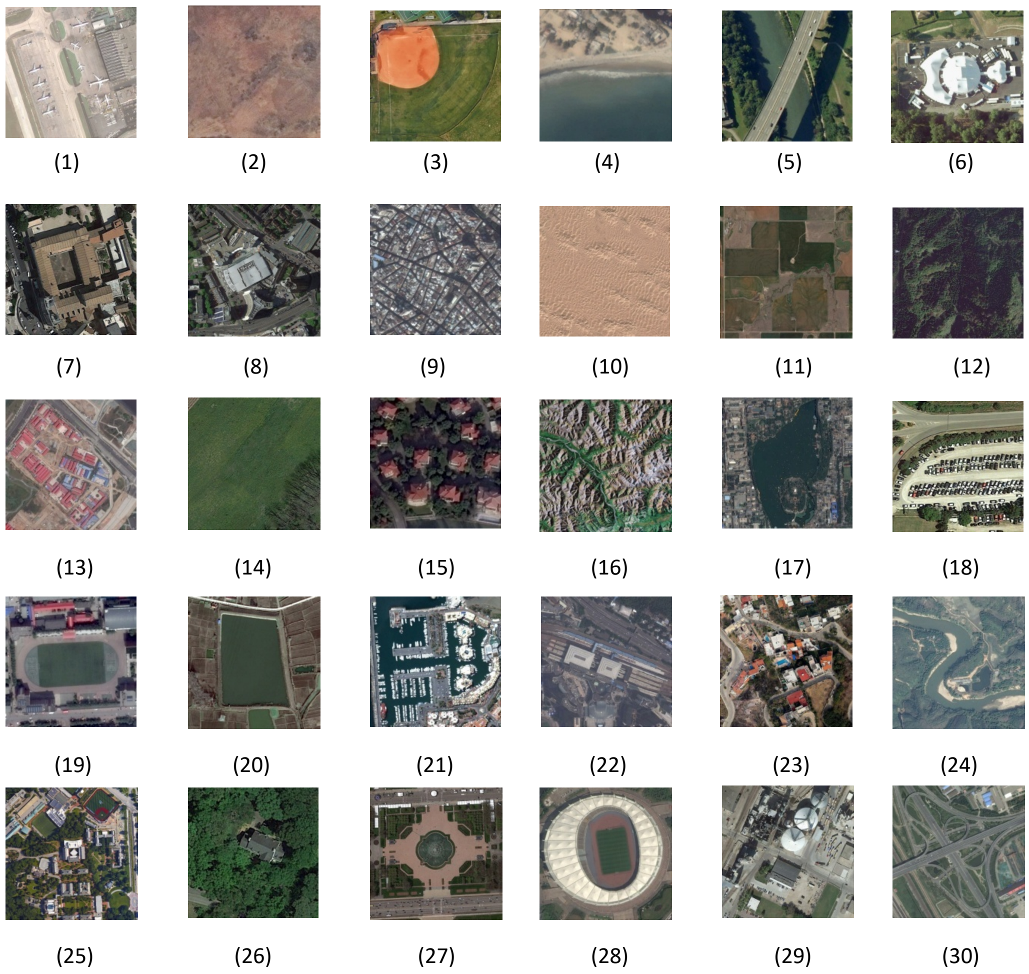


Figure 3.5: Example images of the AID dataset: (1) airport; (2) bare land; (3) baseball field; (4) beach; (5) bridge; (6) centre; (7) church; (8) commercial; (9) dense residential; (10) desert; (11) farmland; (12) forest; (13) industrial; (14) meadow; (15) medium residential; (16) mountain; (17) park; (18) parking; (19) playground; (20) pond; (21) port; (22) railway station; (23) resort; (24) river; (25) school; (26) sparse residential; (27) square; (28) stadium; (29) storage tanks; (30) viaduct.

RSSCN7[3] [188] dataset is released in 2015 by Wuhan University, China, which contains 2,800 remote sensing images in total from seven typical scene categories: grasslands, forests, farmland, car parks, residential areas, industrial areas and rivers and lakes, as shown in Figure 3.6. For each category, there are 400 images with the size 400 × 400 collected from Google earth and these pictures are sampled at four different scales. It is also a challenging dataset because the remote images were taken in different seasons and weather conditions with various sampling scales.

[3]https://github.com/palewitout/RSSCN7

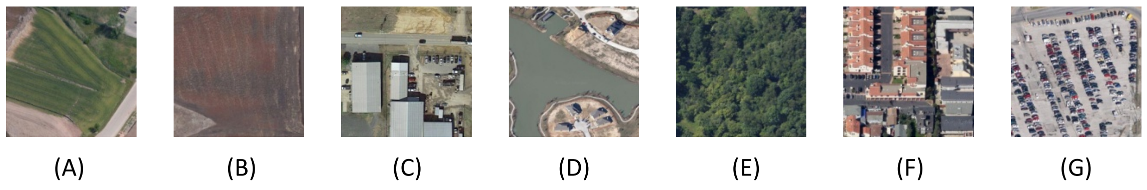


Figure 3.6: Example images of the RSSCN7dataset: (A) Grass; (B) Field; (C) Industry; (E) River Lake; (E) Forest; (F) Resident; (G) Parking.

3.3.2 Experimental Settings

We conducted experiments on the VGG models to show our performance compared to the state-of-the-art methods. To further demonstrate the superiority of our method, ResNet [9] was also chosen as the backbone network in the experiment due to its more complicated and less redundant structure compared to VGG models [8].

3.3.3 Implementation Details

The proposed approach was implemented using the PyTorch framework in this work. We followed the PyTorch instructions [189] to perform data argumentation during dataset preprocessing. The hardware configurations were the Linux Ubuntu 14.04 operating system with i7-5960X CPU, 64GB RAMs, and one NVIDIA GTX1080Ti GPU. After completing each training epoch, just one hyper-parameter $P_i = P$ was used to prune all convolutional layers during the pruning stage. During the entire pruning process, the pruning rate strikes a balance between compression and accuracy [30]. Notably, projection shortcuts do not need to be pruned for compression due to their negligible impact on total costs when evaluated with ResNet.

3.3.3.1 Implementation Details on General Image Datasets

For the general image datasets, all images in the CIFAR-10 dataset were randomly resized to 32×32 pixels, which follows the same image size in the CIFAR-100 dataset. Regarding the ImageNet dataset, we resized all images from various pixels to 224×224 pixels for the same reason.

Stochastic Gradient Descent (SGD) was employed as the optimizer during network training, with the weight decay and momentum set to 0.0001 and 0.90, respectively. The total number of training epochs for small-scale datasets (CIFAR-10 and CIFAR-100) is 300, and our technique was evaluated on VGG-16 and ResNet-20, -32, -56, and -110, respectively. The learning rate is initially set as 0.01. Then it will be changed to 0.001 after the 150 epochs and 0.0001 for the last 75 epochs. As soon as the training losses converged, the training processes for all network models ended. In addition, ResNet-18 and ResNet-50 were trained for 100 epochs with the batch size of 256, weight decay of 1e-4, and momentum of 0.9 on the ImageNet dataset to evaluate their performance. Our strategy eliminates the requirement for fine-tuning following scratch-wise model training, in contrast to many earlier approaches that utilized the hard pruning method.

3.3.3.2 Implementation Details on Remote Sensing Image Datasets

For the remote sensing datasets, all images in the AID dataset were resized to 256×256 pixels from the original 600×600 pixels, which follows the same image size in NWPU-RESISC45 dataset. Regarding RSSCN7 dataset, we resized all images from 400×400 pixels to 256×256 pixels for the same reason. We applied the same training ratio (80% to make fair comparisons of the experiments and all three datasets were randomly divided into training and testing sets based on the pre-set ratios to calculate the overall classification accuracy. All the experiments were conducted in three times to get fair and reliable results.

In the network training, we used SGD as the optimizer with weight decay and momentum as 0.0005 and 0.9, respectively. The learning rates were set separately for two training phases: 0.01 in the first 50 epochs and 0.002 for the last 50 epochs. The training processes for all network models were terminated when the training losses converged. Moreover, the batch-size was set to 64 for NWPU-RESISC45 dataset, 32 for AID dataset and RSSCN7 dataset to balance the requirements of the computer memory and the image number contained in the training and test sets.

3.3.4 Evaluation Metrics

For image classification tasks, we tested the pruned models on CIFAR-10, CIFAR-100, NWPU-RESISC45, RSSCN7 and AID and calculated the Top-1 accuracy. Experiments utilized four distinct evaluation metrics, namely Accuracy (Acc.), Accuracy Drop (Acc.Drop), FLOPs, and pruning ratio (Pruning(%)), where the symbols in the tables are denoted in italics. Pruning ratio (Pruning(%)) means the FLOPs reduction of pruned model. If the result in Accuracy Drop (Acc.Drop) is negative, it indicates that the model accuracy after pruning exceeds the baseline accuracy. Pruning ratio and model accuracy are two most common evaluation metrics for almost all network pruning approaches. The global pruning rate of each algorithm is adjusted in the comparison so that the model sizes of all models pruned by different approaches are roughly equivalent, e.g., 40%. Moreover, we measure the Top-1 and Top-5 accuracies on the large-scale dataset ImageNet. Specifically, Top-1(%) and ΔTop-5(%) demonstrate the classification Top-1 and Top-5 accuracy produced by the given approach, which is expressed as accuracy after running the experiments. ΔTop-1(%) and ΔTop-5(%) are calculated by subtracting the Top-1 and Top-5 accuracy of the pruned model from that of the baseline model, with a negative value indicating that the pruned model achieves an even greater level of accuracy than the baseline model. If the Acc. Drop or ΔTop(%) is smaller, it means the pruned model obtains better performance. The pruning rate represents the actual compression ratio of the network model. Large pruning rate indicates that the pruned model is more compact. The total number of floating-point operations (FLOPs) is utilized as a reference metric when assessing the pruning method. FLOPs(%)$\downarrow$ means the FLOPs reduction of pruned model.

3.3.5 Experiments Results on General Image Datasets

3.3.5.1 Results on CIFAR-10

Table 3.1: Comparison of pruning various networks on the CIFAR-10 dataset.

Method	model	Acc.(%)	Acc. Drop(%)	FLOPs	Pruning (%)
SFP [30]	20	$92.20 \rightarrow 90.83$	1.37	2.43×10^7	42.2
PFAM(ours)		$\mathbf{92.20 \rightarrow 91.32}$	**0.88**	2.43×10^7	42.2
FPGM [37]		$92.20 \rightarrow 90.44$	1.76	1.87×10^7	54.0
PFAM(ours)		$\mathbf{92.20 \rightarrow 90.52}$	**1.68**	1.87×10^7	54.0
MIL [190]	32	$92.33 \rightarrow 90.74$	1.59	4.70×10^7	31.2
SFP [30]		$92.63 \rightarrow 92.08$	0.55	4.03×10^7	41.5
FPGM [37]		$92.63 \rightarrow 91.93$	0.70	3.23×10^7	53.2
TAS [116]		N/A	0.73	3.50×10^7	49.4
PFAM(ours)		$\mathbf{92.63 \rightarrow 92.22}$	**0.41**	3.23×10^7	53.2
PFEC [33]	56	$93.04 \rightarrow 91.31$	1.73	9.09×10^7	27.6
CP [109]		$92.80 \rightarrow 90.90$	1.90	N/A	50.0
SFP [30]		$93.59 \rightarrow 92.26$	1.33	5.94×10^7	52.6
Rethink [191]		$93.80 \rightarrow 92.80$	1.00	5.94×10^7	52.6
FPGM [37]		$93.59 \rightarrow 92.93$	0.66	5.94×10^7	52.6
TAS [116]		N/A	0.77	5.95×10^7	52.7
PFAM(Ours)		$\mathbf{93.59 \rightarrow 93.30}$	**0.29**	5.94×10^7	52.6
MIL [190]	110	$93.63 \rightarrow 93.44$	0.19	N/A	34.2
PFEC [33]		$93.53 \rightarrow 92.94$	0.59	1.55×10^8	38.6
GAL [192]		$93.26 \rightarrow 92.74$	0.52	N/A	40.5
Rethink [191]		$93.77 \rightarrow 93.70$	0.07	1.50×10^8	40.8
SFP [30]		$93.68 \rightarrow 93.38$	0.30	1.50×10^8	40.8
PFAM(Ours)		$\mathbf{93.68 \rightarrow 93.90}$	**-0.22**	1.21×10^8	52.3
FPGM [37]		$93.68 \rightarrow 93.73$	-0.05	1.21×10^8	52.3
TAS [116]		N/A	0.64	1.19×10^8	53.0
PFAM(Ours)		$\mathbf{93.68 \rightarrow 93.66}$	**0.02**	9.40×10^7	62.8
PFEC [33]	VGG-16	$93.58 \rightarrow 93.28$	0.30	2.06×10^8	34.2
FPGM [37]		$93.58 \rightarrow 93.23$	0.35	2.01×10^8	35.9
PFAM(Ours)		$\mathbf{93.58 \rightarrow 93.66}$	**-0.09**	2.01×10^8	35.9

Table 3.1 shows the results on the CIFAR-10 dataset when applying ResNet-20, -32, -56, and -110 respectively for image classification. For ResNet-20, PFAM yields a greater classification accuracy (91.32% as opposed to 90.83%) with the same FLOPs reduction as SFP [30]. Similarly, our method also obtains a better performance (90.52%) than FPGM (90.44%) while keeping the same FLOPs reduction [37].On ResNet-32, PFAM establishes the best compromise between model compression and image classification precision than MIL [190], SFP [30], FPGM [37] and TAS [116]. For ResNet-56, compared to the classical hard pruning approach PFEC [33], PFAM achieves a considerable top-1 accuracy improvement and decreases FLOPs by a large

margin. Besides, although the FLOPs reduction of PFAM is similar as that of CP [109], SFP [30], Rethink [191], FPGM [37], and TAS [116], it results in a smallest accuracy drop (0.29% versus 1.90%, 1.33%, 1.00%, 0.66%, and 0.77%). Similar results happen on ResNet-110. PFAM is superior than MIL [190], PFEC [33], GAL [192], SFP [30], and Rethink [191] in both Top-1 accuracy and FLOPs reduction. To be specific, PFAM yields greater accuracy than the baseline while getting the most significant reduction in FLOPs. However, the other approaches lose accuracy to varying degrees when attaining a small model compression ratio. Besides, PFAM obtains the highest accuracy with similar FLOPs reduction compared to FPGM [37] and TA [116]. Notably, our method obtains the highest model compression ratio and achieves almost the same accuracy as the baseline. Lastly, we utilize the VGG-16 model to run experiments on the CIFAR-10 dataset. It can be seen that PFAM achieves superior performance compared to the other state-of-the-art filter pruning methods under similar FLOPs reduction.

3.3.5.2 Results on CIFAR-100

Table 3.2: Comparison of pruning various networks on the CIFAR-100 dataset.

Method	model	Acc. Drop(%)	Pruning (%)
MIL [190]		2.96	39.3
SFP [30]	56	2.61	52.6
FPGM [37]		1.75	52.6
PFAM(ours)		**0.32**	52.6
FPGM [37]	VGG-16	-0.20	35.9
PFAM(ours)		**-0.44**	35.9

Similarly, as demonstrated in Table 3.2, our proposed method consistently outperformed the other state-of-the-art methods on the CIFAR-100 dataset. In particular, the FLOPs reduction of PFAM on ResNet56 is over 52%, while its accuracy decreases by just 0.32%. MIL [190] reduces the model's FLOPs by 39.3% but results in a nearly 3% performance decrease. Although SFP [30] and FPGM [37] also achieved a significant reduction in FLOPs, their accuracy decreased more than ours (0.32 versus 2.61% and 1.75%). For VGG-16, PFAM achieves the same pruning rate but improves performance more than FPGM [37].

3.3.5.3 Results on ImageNet

Table 3.3: Comparison of pruning ResNet on the ImageNet dataset.

Method	model	Top-1(%)	ΔTop-1(%)	Top-5(%)	ΔTop-5(%)	FLOPs(%)↓
MIL [190]	18	$69.98 \rightarrow 66.33$	3.65	$89.24 \rightarrow 86.94$	2.30	34.6
SFP [30]	18	$70.28 \rightarrow 67.10$	3.18	$89.63 \rightarrow 87.78$	1.85	41.8
FPGM [37]	18	$70.28 \rightarrow 67.78$	2.50	$89.63 \rightarrow 88.01$	1.62	41.8
PFAM(ours)	18	$\mathbf{70.28 \rightarrow 67.92}$	**2.36**	$\mathbf{89.63 \rightarrow 87.95}$	**1.68**	**53.3**
SFP [30]	50	$76.15 \rightarrow 74.61$	1.54	$92.87 \rightarrow 92.06$	0.81	41.8
CP [109]	50	N/A	N/A	N/A	1.40	50.0
GDP [21]	50	$75.13 \rightarrow 71.89$	3.24	$92.30 \rightarrow 90.71$	1.59	51.3
FPGM [37]	50	$76.15 \rightarrow 74.13$	2.02	$92.87 \rightarrow 91.94$	0.93	53.5
PFAM(ours)	50	$\mathbf{76.15 \rightarrow 74.76}$	**1.39**	$\mathbf{92.87 \rightarrow 92.05}$	**0.82**	53.5

For the ImageNet dataset, we evaluated the performance of our PFAM on ResNet-18 and ResNet-50. In Table 3.3, PFAM has the best result on ResNet-18 compared to the other three approaches. PFAM significantly outperforms MIL [190] in terms of Top-1 accuracy,Top-5 accuracy, and FLOP reduction. In addition, in comparison to SFP [30] and FPGM [37], which similarly prune filters in a soft-manner, PFAM achieves the greatest Top1 accuracy (2.36% vs. 2.50% and 3.18%) and the biggest FLOPs reduction (53.3% vs. 41.8%). Besides, PFAM delivers the best performance on ResNet50 in terms of both Top–1 and Top–5 accuracy with an elevated model compression rate. Even if the Top-1 accuracy of SFP [30] is comparable to that of PFAM, it is unable to achieve a FLOPs reduction as high as ours. GDP [21] and FPGM [37] can accelerate ResNet50 to a similar level to PFAM, while TOP1 accuracy and Top-5 accuracy loss are not yet at our level. Therefore, It can be seen that our proposed method also showed a high effect on selecting and pruning redundant filters both on ResNet-18 and ResNet-50.

3.3.6 Experiments Results on Remote Sensing Image Datasets

In this section, the comparison results for the proposed method and the baselines on three datasets are discussed. We trained the original models of VGG [8] and ResNet [9] from scratch and set their results as the baseline and three state-of-the-arts: PFEC [33], SFP [30] and FPGM [37], are further applied in the performance comparison.

3.3.6.1 Results on NWPU-RESISC45

Table 3.4-3.7 show the results on NWPU-RESISC45 dataset when applied ResNet-18, 34, 50 and 101 respectively for remote sensing image classification, where our proposed method achieves superior performance compared to the other state-of-the-art filter pruning methods. Particularly, in Table 3.4 and Table 3.5, PFEC [33] used the hard pruning manner in pruning ResNet-18 and 34 with the accuracies of 89.78% and 90.75% independently whereas the figures from SFP [30] are 0.29% and 0.42% lower than them. However, the soft manner methods such as FPGM [37] and PFAM obtain better results than these hard pruning ones, where PFAM achieves the highest accuracies (92.56% and 93.46%) among them with the second biggest compression ratios of 42.3% and 40.6%. As shown in Table 3.6 and Table 3.7, the soft pruning methods consistently outperform the hard pruning ones, where PFAM achieves the lowest accuracy drops of 0.27% and 0.5% on ResNet-34 and 50, respectively. FPGM [37] achieves the best accuracy (92.64%) between four methods when pruning ResNet-101, which is slightly better than our PFAM (92.52%). It is noted that, in this case, the pruning ratio of our PFAM (39.6%) is larger than that of FPGM [37] (38.1%). To sum up the above experimental results, our PFAM obtains a highly compressed network model with competitive performance, given the NWPU-RESISC45 dataset.

Table 3.4: Comparison of pruning ResNet-18 on the NWPU-RESISC45 dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	93.40(±0.32)	0	6.05E8	0
PFEC [33]	89.78(±0.15)	3.62	3.78E8	37.5
SFP [30]	89.49(±0.27)	3.91	3.41E8	43.6
FPGM [37]	92.28(±0.18)	1.12	3.68E8	39.2
PFAM(ours)	**92.56**(±0.23)	**0.84**	3.49E8	42.3

Table 3.5: Comparison of pruning ResNet-34 on the NWPU-RESISC45 dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	93.73(±0.11)	0	1.22E9	0
PFEC [33]	90.75(±0.24)	2.98	7.47E8	38.8
SFP [30]	90.33(±0.17)	3.40	6.81E8	44.2
FPGM [37]	93.24(±0.08)	0.49	7.39E8	39.4
PFAM(ours)	**93.46**(±0.14)	**0.27**	7.25E8	40.6

Table 3.6: Comparison of pruning ResNet-50 on the NWPU-RESISC45 dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	93.37(±0.21)	0	1.36E9	0
PFEC [33]	90.65(±0.33)	2.72	8.50E8	37.5
SFP [30]	91.35(±0.23)	2.02	8.49E8	37.6
FPGM [37]	92.57(±0.12)	0.80	8.69E8	36.1
PFAM(ours)	**92.87**(±0.25)	**0.50**	8.44E8	37.9

Table 3.7: Comparison of pruning ResNet-101 on the NWPU-RESISC45 dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	93.17(±0.13)	0	2.60E9	0
PFEC [33]	88.49(±0.35)	4.68	1.43E9	45.0
SFP [30]	91.90(±0.28)	1.27	1.58E9	39.2
FPGM [37]	92.64(±0.06)	0.53	1.61E9	38.1
PFAM(ours)	92.52(±0.14)	0.65	1.57E9	39.6

3.3.6.2 ResNet on AID Dataset

Similarly, our proposed method consistently outperforms the other state-of-the-arts on AID dataset as shown in Table 3.8, 3.9, 3.10, 3.11. Although SFP [30] achieves the largest pruning ratios on ResNet-18, 34, 50 and 101, much higher accuracies are achieved by our proposed PFAM, where the gaps are 4.01%, 4.26%, 3% and 3.17%, respectively. FPGM [37] obtains the second-best results in these experiments and the accuracy is at least 0.19% (on ResNet-18: 85.08% vs. 85.27%) lower than that of PFAM. It is worth mentioning that our filter pruning method even outperforms the original model when pruning Resnet-101 in terms of accuracy, which is 84.17% and 84.10% separately. That indicates the powerful ability of PFAM in producing a highly compressed model while maintaining competitive performance.

To provide more comprehensive performance verification of our method, we also tested our method in pruning VGG-16 with training from scratch (Table 3.12) and ResNet based on the pretrained model (Table 3.13) on AID dataset. The results in Table 3.12 describe the effectiveness of the soft pruning methods compared to the hard ones on VGG-16 model, where PFAM still achieves the best accuracy (86.03%) under the same compression ratio among four pruning methods. The performance degradation is inevitable for the hard filter pruning method like PFEC [33] while the soft filter pruning methods enable to keep the strong network expressive ability after the reconstruction stage on the pruned filters to obtain better performance. In Table 3.13, the proposed PFAM achieves the best accuracies in pruning ResNet-18 (89.86%), 34 (89.77%) and 101 (90.57%), which demonstrate the effectiveness of our method in pruning ResNet with the pretrained model.

Table 3.8: Comparison of pruning ResNet-18 on the AID dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	85.56(±0.12)	0	6.05E8	0
PFEC [33]	81.95(±0.23)	3.61	3.79E8	37.4
SFP [30]	81.26(±0.27)	4.30	3.38E8	44.1
FPGM [37]	85.08(±0.13)	0.48	3.68E8	39.2
PFAM(ours)	**85.27**(±0.11)	**0.29**	3.60E8	40.5

Table 3.9: Comparison of pruning ResNet-34 on the AID dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	85.83(±0.11)	0	1.22E9	0
PFEC [33]	81.63(±0.22)	4.20	7.47E8	38.8
SFP [30]	79.96(±0.25)	5.87	6.78E8	44.4
FPGM [37]	83.55(±0.07)	2.28	7.39E8	39.4
PFAM(ours)	**84.22**(±0.16)	**1.61**	7.32E8	40.0

Table 3.10: Comparison of pruning ResNet-50 on the AID dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	84.86(±0.27)	0	1.36E9	0
PFEC [33]	81.60(±0.19)	3.26	8.50E8	37.5
SFP [30]	81.43(±0.33)	3.43	8.44E8	37.9
FPGM [37]	83.76(±0.15)	1.10	8.69E8	36.1
PFAM(ours)	**84.43**(±0.22)	**0.43**	8.54E8	37.2

Table 3.11: Comparison of pruning ResNet-101 on the AID dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	84.10(±0.19)	0	2.60E9	0
PFEC [33]	83.27(±0.11)	0.83	1.69E9	35.0
SFP [30]	81.00(±0.32)	3.10	1.57E9	39.6
FPGM [37]	83.87(±0.07)	0.23	1.61E9	38.1
PFAM(ours)	**84.17**(±0.13)	**-0.07**	1.59E9	38.8

Table 3.12: Comparison of pruning VGG-16 on the AID dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	86.55(±0.11)	0	5.12E9	0
PFEC [33]	82.31(±0.35)	4.24	3.26E9	36.3
SFP [30]	85.58(±0.23)	0.97	3.08E9	39.8
FPGM [37]	85.88(±0.11)	0.67	3.08E9	39.8
PFAM(ours)	**86.03**(±0.14)	0.52	3.08E9	39.8

Table 3.13: Accuracies (%) of pruning ResNet based on pretrained model on the AID dataset

Method	ResNet-18 (%)	ResNet-34 (%)	ResNet-50(%)	ResNet-101 (%)
Baseline	90.07(±0.11)	90.53(±0.28))	90.73(±0.33)	90.84(±0.27)
PFEC [33]	88.10(±0.35)	88.34(±0.25)	88.49(±0.26)	89.76(±0.15)
SFP [30]	89.03(±0.23)	89.15(±0.32)	89.57(±0.44)	90.13(±0.35)
FPGM [37]	89.44(±0.11)	89.71(±0.19)	90.53(±0.21)	90.55(±0.17)
PFAM(ours)	**89.86**(±0.14)	**89.77**(±0.23)	90.32(±0.28)	**90.57**(±0.29)

3.3.6.3 Results on RSSCN7

For the RSSCN7dataset, we tested our PFAM on ResNet-18, 34, 50 and 101 and VGG-16 with 40% pruning ratio to provide comprehensive insights on the performance. From Table 3.14 to Table 3.18, unlike previous results on NWPU-RESISC45 and AID datasets, SFP [30] obtains the worse performances in ResNet experiments even though it gets the highest compression ratios. However, FPGM [37] and our proposed method generally achieve better experimental results than the norm-criterion methods like PFEC [33] and SFP [30]. The reason for the worse performance is that the norm-criterion methods only focus on pruning each individual filter without considering the global correlation among all filters. Therefore, it leads to the suboptimal performance. Except for the result in pruning ResNet-50, PFAM achieves the best performances in the rest experiments on all ResNet models. Compared to the methods selecting filters based on the norm-based criterion and the geometric median, the proposed attention module is utilized in PFAM to find the correlation between filters globally, which enables to yield superior performance because of advanced pruning strategy.

Table 3.14: Comparison of pruning ResNet-18 on the RSSCN7 dataset.

Method	Acc.(%)	Acc. Drop(%)	FLOPs	Pruning (%)
Baseline	86.43(±0.23)	0	6.05E8	0
PFEC [33]	83.57(±0.13)	2.68	3.79E8	37.4
SFP [30]	82.86(±0.16)	3.57	3.35E8	44.6
FPGM [37]	85.00(±0.22)	1.43	3.68E8	39.2
PFAM(ours)	**85.51**(±0.17)	**0.92**	3.63E8	40.0

Table 3.15: Comparison of pruning ResNet-34 on the RSSCN7 dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	86.07(±0.14)	0	1.22E9	0
PFEC [33]	83.39(±0.28)	2.68	7.47E8	38.8
SFP [30]	81.83(±0.22)	4.24	6.66E8	45.4
FPGM [37]	85.37(±0.18)	0.70	7.39E8	39.4
PFAM(ours)	**85.71**(±0.25)	**0.36**	7.36E8	39.7

Table 3.16: Comparison of pruning ResNet-50 on the RSSCN7 dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	85.28(±0.21)	0	1.36E9	0
PFEC [33]	80.71(±0.17)	4.57	8.50E8	37.5
SFP [30]	80.36(±0.31)	4.92	8.42E8	38.1
FPGM [37]	84.64(±0.25)	0.64	8.69E8	36.1
PFAM(ours)	83.93(±0.28)	1.35	8.66E8	36.3

Table 3.17: Comparison of pruning ResNet-101 on the RSSCN7 dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	85.35(±0.15)	0	2.60E9	0
PFEC [33]	83.39(±0.24)	1.96	1.69E9	35.0
SFP [30]	82.93(±0.41)	2.42	1.56E9	40.0
FPGM [37]	83.92(±0.22)	1.43	1.61E9	38.1
PFAM(ours)	**83.93**(±0.26)	**1.42**	1.61E9	38.1

Table 3.18: Comparison of pruning VGG-16 on the RSSCN7 dataset.

Method	**Acc.(%)**	**Acc. Drop(%)**	**FLOPs**	**Pruning (%)**
Baseline	90.00(±0.17)	0	5.12E9	0
PFEC [33]	85.18(±0.33)	4.82	3.26E9	36.3
SFP [30]	85.18(±0.13)	4.82	3.08E9	39.8
FPGM [37]	87.68(±0.21)	2.32	3.08E9	39.8
PFAM(ours)	**88.04**(±0.22)	**1.96**	3.08E9	39.8

3.3.7 Ablation Study

3.3.7.1 Impact of Differentiated Pruned FLOPs

In this section, we first conduct experiments on CIFAR-10 using ResNet-56 at various filter pruning ratios (0.1, 0.2, 0.3, 0.4, and 0.5) to explore the effect of different pruning rates on the performance of the model. Since our method belongs to the soft pruning methods, we verified two other soft pruning methods for a fair comparison. Hence, three soft filter pruning methods, SFP [30], FPGM [37], and PFAM (ours), were tested to verify the effectiveness of our method.

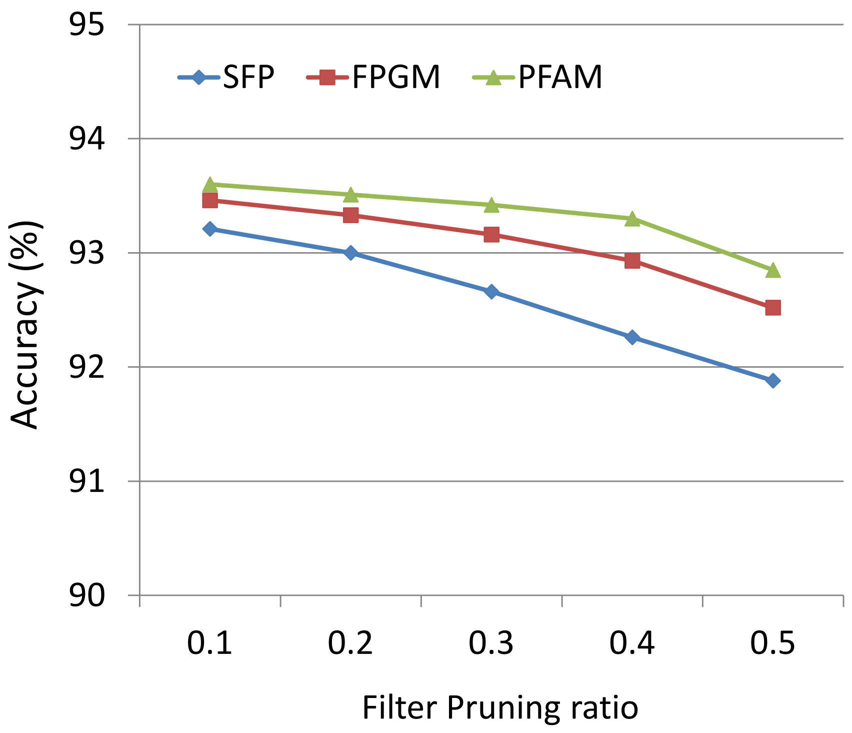


Figure 3.7: Influence of varied pruned FLOPs on pruning.

From Fig. 3.7, it is clear that our method generally yields competitive performance throughout five different evaluations, even if performance is generally trending worse for all three models as the pruning ratios rise. Despite the fact that we all prune filters in a soft manner, it can also demonstrate that our strategy for selecting to-be-pruned filters is superior to the alternatives. The essential advantage of the proposed method is that it takes into account the overall correlation between filters in a single convolutional layer, as opposed to focusing just on the value of each individual filter.

In addition, we conducted extensive experiments on remote sensing image datasets regarding the performance variations by applying different pruning ratios on ResNet-18, 34, 50 and 100. The corresponding results are shown from Figure 3.8 to Figure 3.13. Here we only verified soft pruning methods for a fair comparison. Hence, three soft filter pruning methods: SFP [30], FPGM [37] and PFAM(ours) were tested on NWPU-RESISC45 and RSSCN7 datasets respectively. It is worth mentioning that the performance of SFP [30] becomes far worse than the other two methods with the increasing pruning ratio on both RSSCN7 dataset and NWPU-RESISC45 dataset, where the bottom parts of its curves are omitted in the following figures to analyze in greater detail the performance differences between FPGM [37] and our method within a limited range of accuracy.

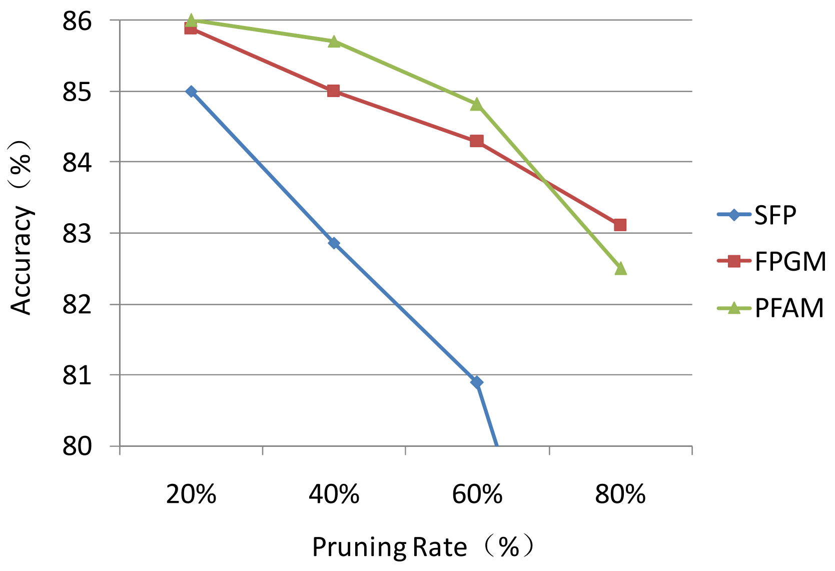


Figure 3.8: The comparisons of SFP(blue), FPGM(red) and PFAM(green) that pruning ResNet-18 on RSSCN7 dataset with pruning ratios 0.2, 0.4, 0.6 and 0.8.

For RSSCN7 dataset, our method achieves the best performances at most pruning ratios(20%, 40% and 60%) in pruning ResNet-18, while the accuracy of FPGM [37] is the highest at the largest compression ratio of 80% in Figure 3.8 and a similar situation happened in the experiment results of pruning ResNet-34, which can be seen in Figure 3.9. It can be illustrated that in addition to an extreme pruning setting, our method can obtain competitive performance at various pruning rates in pruning ResNet-18 and ResNet-34. In Figure 3.10, however, PFAM obtains higher accuracies than those of FPGM [37] at the three different pruning ratios of 20%, 60% and 80% separately while FPGM [37] only gets better performance than ours in one purning rate. Finally, from Figure 3.11 we can see that our method obtains the overall best performance than the other two methods in pruning ResNet-101 and FPGM [37] only gets similar performance as ours in pruning ratios of 40%. Above all, compared to the state-of-the-art baselines, PFAM can achieve superior performance in pruning various models on the remote sensing datasets at different pruning ratios. On NWPU-RESISC45 dataset, SFP [30] continues the poor performance with the increasing pruning ratio. PFAM (green line) outperforms the other two methods throughout the whole pruning ratios, which is shown in Figure 3.12. In Figure 3.13, PFAM achieves the best performance under all settings on ResNet-34, except for FPGM [37] at the pruning rate of 60%.

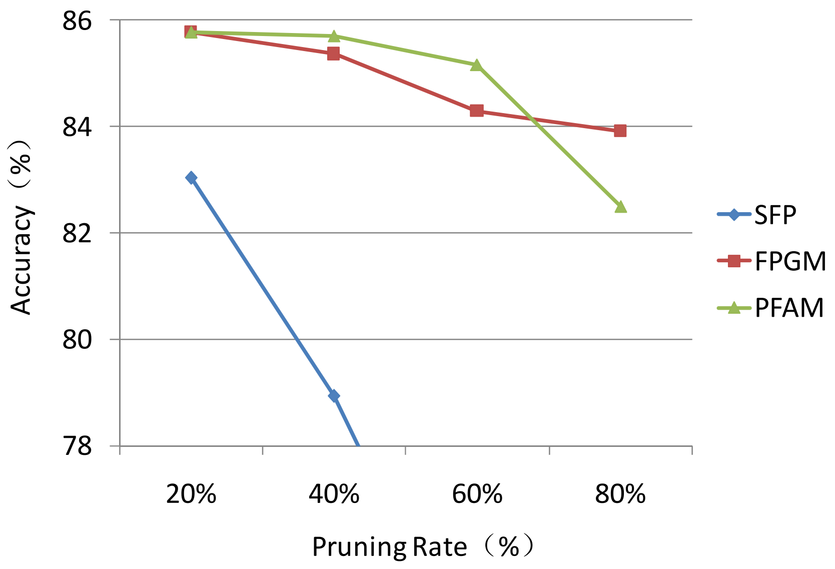


Figure 3.9: The comparisons of three soft pruning methods that pruning ResNet on RSSCN7 dataset with four different pruning ratios. The blue line, red line and green line represent the accuracy of SFP, FPGM and PFAM on pruning ResNet-34 respectively.

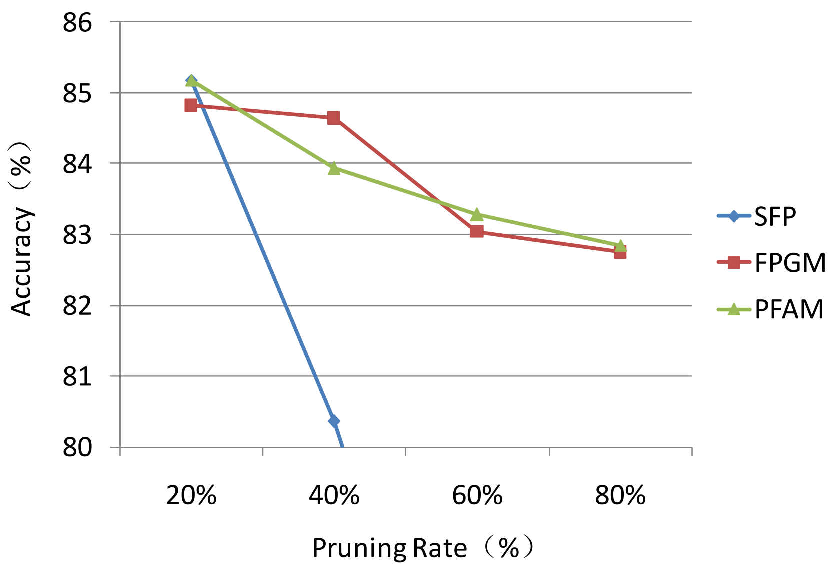


Figure 3.10: The comparisons of SFP, FPGM and PFAM that pruning ResNet on RSSCN7 dataset with four different pruning ratios. Three different color(blue, red and green) lines indicate the accuracy of SFP, FPGM and PFAM on pruning ResNet-50 respectively.

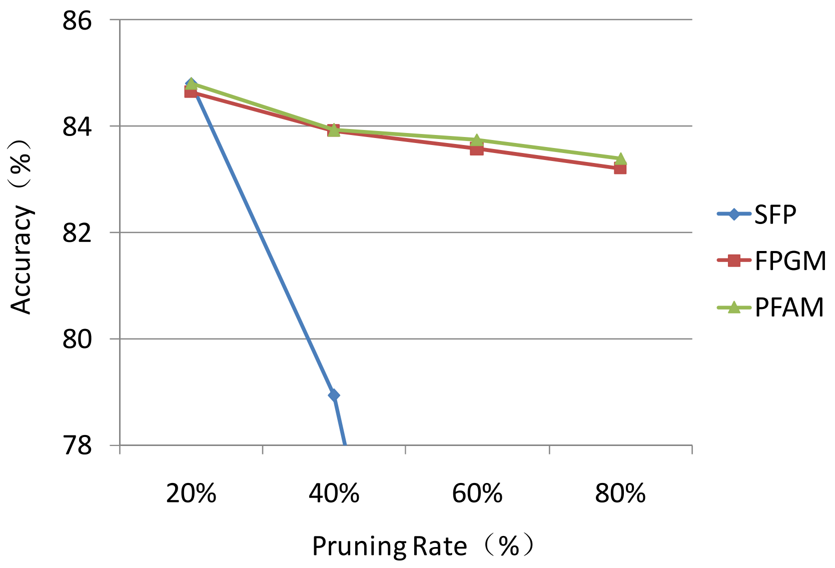


Figure 3.11: The comparisons of three soft pruning methods such as SFP(blue), FPGM(red) and PFAM(gree) that pruning ResNet-101 on RSSCN7 dataset with four different pruning ratios(20%, 40%, 60% and 80%).

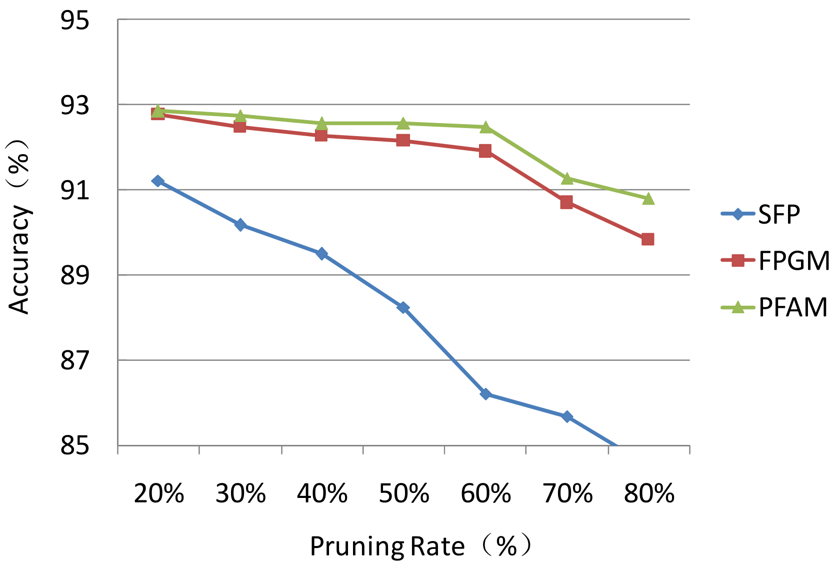


Figure 3.12: The comparisons among three methods on pruning ResNet-18 with 7 different pruning ratios.

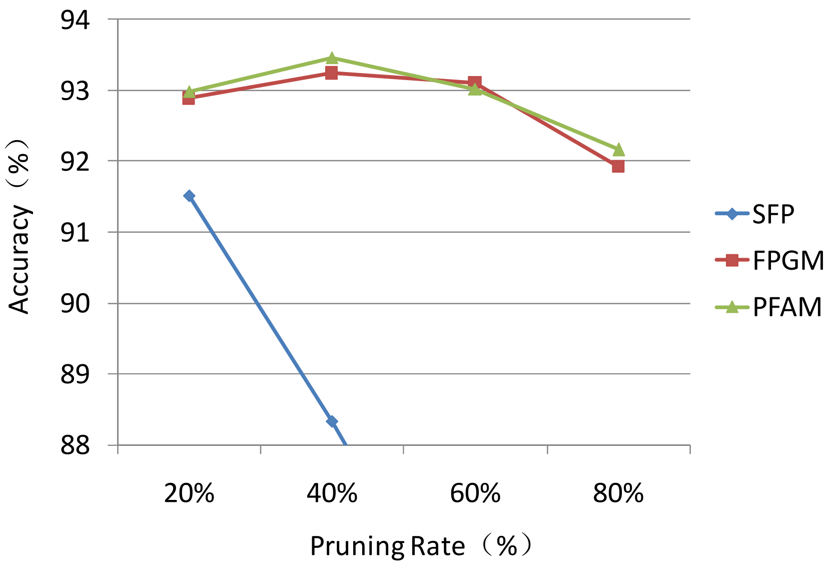


Figure 3.13: The comparisons among three methods on pruning ResNet-34 for 20%, 40%, 60% and 80% pruning rates.

3.3.8 Ablation Study on Remote Sensing Image Datasets

3.3.8.1 Influence of Adding Fine-Tuning Stage

As previously stated, one of the benefits of soft pruning is the ability to eliminate the time-consuming stage of fine-tuning without suffering significant performance losses. Nevertheless, even though the model's performance is excellent after the training and pruning stages, we still wonder how much the performance might be enhanced by adding a fine-tuning step. Therefore, in this part, we employ various models on the CIFAR10 dataset to investigate the impact of adding a fine-tuning stage on model performance. Our experiment is separated into two distinct sections: pruning filters without fine-tuning stage and pruning filters with fine-tuning stage. As seen in Fig. 3.14, all models' performance is enhanced to varying degrees with the addition of the fine-tuning step. This further demonstrates the influence and relevance of the fine-tuning phase. However, the performance achieved with soft pruning is so good that it is a tradeoff whether it is worth spending much extra time on fine-tuning.

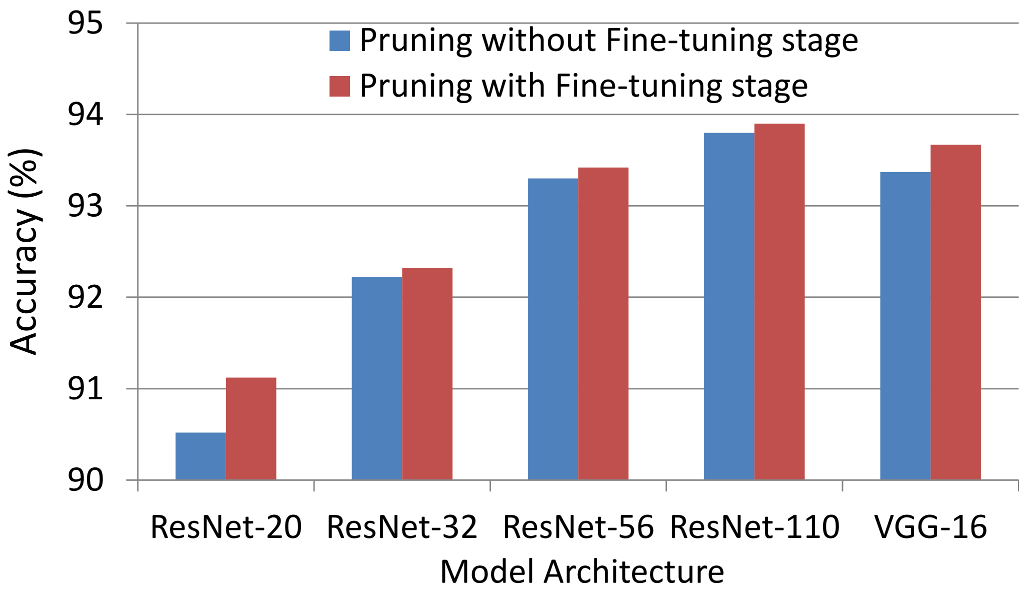


Figure 3.14: Influence of fine-tuning on pruning.

3.4 Chapter Summary

In this chapter, a novel method termed Pruning Filter with Attention Mechanism (PFAM) is proposed for image classification tasks. Specifically, a correlation-based filter pruning criterion is implemented, with the attention mechanism determining the correlation between filters. In contrast to prior pruning approaches, we remove filters with the lowest correlation scores, which has a negligible effect on the total correlation between filters in each layer. These less correlated filters are first pruned during the pruning step of the current training epoch, then recovered and updated during the subsequent training epoch. Consequently, the training data is processed by the original model during the training stage. Therefore, the compressed network model can be created without requiring an additional fine-tuning stage. The proposed method is extensively evaluated on three different-scale public image datasets, and the experimental results demonstrate that our method outperforms the state-of-the-art methods.

In this method, the attention mechanism is introduced to obtain the correlation between the filters. Then the redundant filters in each layer are pruned by setting the global pruning rate. In the next chapter, we focus on measuring the importance of the convolutional layers and exploring which layers are important and need to be preserved. After obtaining the importance information of layers, we then retain the filters of layers with high importance while removing the filters of layers with low importance in the pruning stage. Compared to traditional filter pruning methods, doing so can reduce performance loss under similar model compression. The details of the proposed method will be revealed in the following chapter.

Chapter 4

Data-Dependent Soft Pruning Guided by LSTM

4.1 Introduction

Deep convolutional neural networks (CNNs) have recently demonstrated tremendous success in a variety of computer vision applications, including image classification [8, 9, 86], semantic segmentation [193], image captioning [194–196], object detection [197–199] and recognition [200–202]. However, its success in terms of accuracy has necessitated a large number of model parameters for storage and costly training on GPUs. These factors hinder the application of CNN on resource-restricted devices, such as mobile and embedded devices. To remedy this situation, it is desired that CNN be small and rapid enough while maintaining sufficient precision. Consequently, deep model compression has become a popular research topic.

Currently, the CNN compression techniques can be categorized into four groups. The first category takes advantage of the quantization technique, which often employs a model binarization approach [69]. Although this technique can achieve a high compression rate, binary networks still suffer from a large performance drop and less robustness since it changes the parameter storage types. The second category is network sparsity [24, 25], which is accomplished by pruning irrelevant weights or setting them to zero. However, neither existing software nor hardware libraries can accelerate sparse matrix calculations [30]. The third category makes use of the tensor factorization algorithm [42], which approximates large and complicated tensors by combining small tensors and simple operations. Although it could produce smaller, faster networks, the performance of networks after the operation is dramatically decreased, especially when dealing with classification tasks on large-scale datasets such as ImageNet. The last category is referred to as "filter-wise pruning" [33, 36], which directly removes unnecessary convolutional filters. The filter pruning method effectively maintains the network's structure, leading to smaller, faster models with slight or no performance loss. Therefore, the filter pruning technique has gained the most attention in recent years.

Network compression is essentially a systematic effort, and the pruning choice should be based on a trade-off between all factors from the model. However, it appears that the majority of existing pruning methods [33, 35, 38, 203] simply take into account the information of individual filters and ignore the correlation among layers and filters. Particularly, they concentrate on evaluating the importance of each filter separately at each layer, pruning the filters from top to bottom or bottom to top, layer by layer. In other words, they regard each layer identically, which is antithetical to the phenomenon that the layers differ in terms of importance. If many filters are removed from a critical layer, the performance of the entire system may drop dramatically. In contrast, pruning the number of filters in an unimportant layer might drastically reduce the model's complexity while having no effect on its precision. To illustrate this, we take ResNet-56 as an example. Here, we conduct the filter pruning experiment three times, referred to as three stages, each of which three layers are selected for filter pruning. There is no interaction between stages, meaning each stage's experiments are independent. At each stage, the same number of filters are pruned from each layer, resulting in the same model complexity after pruning. Specifically, we randomly remove the same number of filters from layer 10, layer 12, and layer 18 during our stage two experiment. Then we display the accuracy of the model after pruning at different layers (e.g., L10, L12, or L18 at stage 2) in Fig.4.1, demonstrating that the importance of each layer varies, as the performance difference in accuracy is quite noticeable. Therefore, determining a suitable pruning strategy for each layer is critical. If we select to prune the layers with a more negligible influence on classification accuracy, it will result in a greater reduction in complexity with a smaller performance drop. It turns out that our method achieves a higher pruning rate while maintaining sufficient accuracy.

The aforementioned phenomenon motivates us to investigate the problem that prunes filters in the important layers while removing more filters in some redundant layers. In this work, we propose a novel method for evaluating the importance of each layer and pruning the less significant ones. Specifically, considering that CNNs usually exploit a hierarchical structure that can be represented as a string, we employ long short-term memory (LSTM) [204] as an evaluation metric to generate the pruning decision for each layer. The entire algorithm is carried out in two steps: unimportant

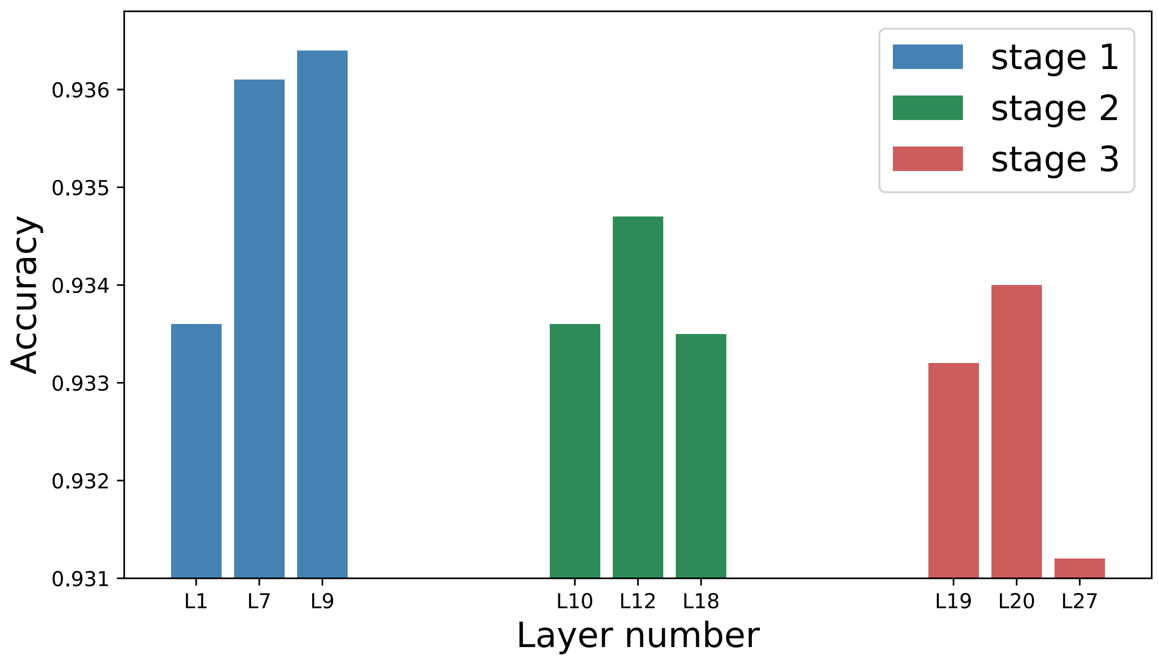


Figure 4.1: The accuracy of ResNet-56 after one layer is pruned. The same number of filters are removed in different layers at the same stage. Lm in x-axis denotes the m^{th} residual block.

layer-finding and unimportant filter-finding. To detect the unimportant layers in a deep CNN, the LSTM is trained using reinforcement learning with model performance and complexity considered in the reward function. In the subsequent unimportant filter-finding step, we employ a channel-based method to go over each filter of a certain neural network layer. Finally, a number of filters with the least importance are pruned. Through several pruning iterations, the slimmer model is generated, which preserves the performance of deep CNN while significantly reducing the complexity of the model. In addition, the new slimmer network usually has a compact structure because the training process of LSTM aids in generating a more efficient architecture.

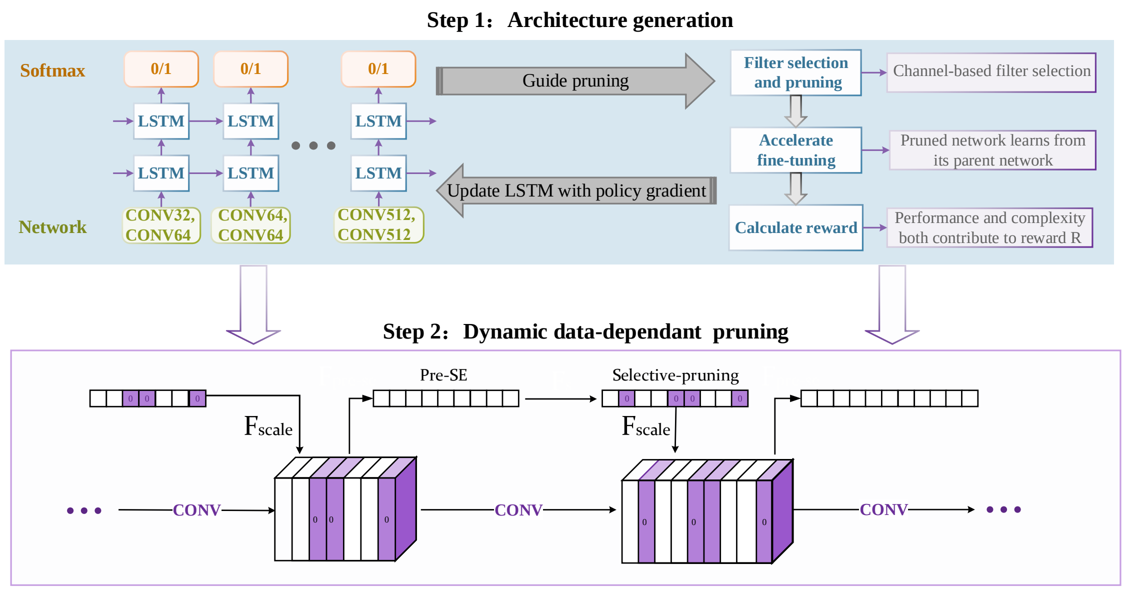
Existing pruning approaches often employ a hard pruning strategy to remove unimportant filters from the network directly, where the importance evaluation of the filter is generally inaccurate due to unclear calculations. Specifically, when a filter is deemed unimportant for the majority of image data, it can be treated as unimportant, even if it plays a relatively important role for a small portion of image data. In this situation, feature extraction and classification performance on these few images are degraded. Therefore applying a hard pruning method will undoubtedly result in a reduction in overall accuracy. On the contrary, a soft pruning strategy allows the pruned filters in the previous epoch to be updated in the next epoch during the training procedure. In this way, no filters are physically removed, and the model capacity can be recovered from the pruned model.

In this work, we propose a novel soft pruning method named Squeeze-Excitation-Pruning (SEP). Referring to the lightweight structure generated by LSTM, the SEP module is applied to rebuild the baseline network. Besides, we use the SEP module to generate the importance scores of all filters for each given image. In our soft pruning, all filters of the baseline network are preserved, but for each input image, only a portion of important ones are involved in the forward and backward calculations. When it comes to different image data, different filters might be softly pruned depending on the selection results of SEP. This data-dependent soft pruning method retains the capacity and knowledge of the baseline model, thus ensuring better performance. Specifically, the entire framework is depicted in Fig.4.2, and our major contributions are summarized as follows:

- We argue that where to prune is actually a critical issue for CNN model compression, which has long been unfortunately neglected. To this end, we propose an end-to-end framework to prune networks in the correct order. Concretely, considering the hierarchical structure of CNN, we employ LSTM as an evaluation model to find the least important layers and thus generate the pruning decision for a given network. LSTM is updated using the policy gradient method with both model performance and complexity as the reward.

- Rather than adopting a hard pruning strategy, we propose the SEP pruning method. SEP is a data-dependent soft pruning method, which preserves all the filter parameters, but for each image data, only some important ones participate in calculating forward and backward propagations. When the given image is changed, different filters may be softly pruned according to the SEP selection.

The rest of this chapter is organized as follows: Section 4.2 introduces the proposed filter pruning method. The experimental results are given and analyzed comprehensively in Section 4.3. The work is concluded with detail summary and introduction of future work in Section 4.4.



Step 1：Architecture generation
Softmax
0/1
0/1
0/1
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
Network
CONV32, CONV64
CONV64, CONV64
CONV512, CONV512
Guide pruning
Update LSTM with policy gradient
Filter selection and pruning
Accelerate fine-tuning
Calculate reward
Channel-based filter selection
Pruned network learns from its parent network
Performance and complexity both contribute to reward R
Step 2：Dynamic data-dependant pruning
Pre-SE
Selective-pruning
Fscale
Fscale
CONV
CONV
CONV

Figure 4.2: The framework of our end-to-end pruning method. The first step is to make pruning decisions based on LSTM evaluation model. After several epochs, a more efficient and slimmer network structure is finally generated. LSTM is updated in the policy gradient method with both model performance and complexity as the reward. In the second step, we rebuild the baseline network by deploying the SEP module for each layer and train it from scratch. The SEP attention module is composed of feature extraction and selection, which generates the weight vector, selects and sets some weights to zero according to the pruning structure generated in the first step. Then, the feature map in the next convolution layer is scaled by this weight vector to achieve dynamic and data-dependent soft pruning.

4.2 Methodology

In this work, we present our end-to-end pruning method. Before going into further details, we briefly explain the basic idea. We first use LSTM to generate the pruning decisions by evaluating the importance of each layer, where the most unimportant layers will be selected to be pruned. Once we have collected such guidance information, the SEP attention mechanism is employed to rebuild the baseline network. Basically, we train a SEP from scratch by deploying the pruning information, e.g., which layers will be pruned and how many filters in each layer need to be pruned, estimated by the preceding LSTM. The SEP module consists of two parts: the pre-SE module includes squeeze and excitation used for feature extraction and weight vector generation; the selective-pruning module selects and sets some weights to zero. Therefore, the SEP module can automatically predict the importance of each feature map and set some weights to zero based on the information of feature extraction. The details of the end-to-end pruning framework in Fig.4.2 are elaborated as follows.

1. **Pruning guidance.** An initial or intermediate network representation is fed into LSTM, and LSTM generates a strategy indicating which layers should be pruned.

2. **Filter selection and fine-tuning.** We evaluate the importance of each filter in the layers chosen by LSTM with a channel-based method, then prune those unimportant filters. Afterwards, we fine-tune the pruned model using the distillation method.

3. **Updating LSTM.** We update LSTM in a reinforcement learning way with both performance and complexity of the pruned model as the reward signal.

4. **Repeat from (1) to (3).**

5. **Data-dependent soft pruning.** Referring to the slimmer architecture generated by LSTM, the baseline network is rebuilt with SEP modules and trained from scratch to achieve data-dependent soft pruning.

4.2.1 Where to Prune

The basic idea can be interpreted as follows: LSTM generates the pruning probability for each layer. The output of each layer is associated with two real values, indicating the probabilities of "Pruning" and "Not Pruning", respectively. Suppose the number of all candidate convolution layers is defined as L, we can get one matrix $P \in \mathbb{R}^{L\times 2}$, corresponding to the pruning probabilities of L conv layers. If the probability of "Not Pruning" is larger than that of "Pruning" in one row, we treat this row as "0", meaning we do nothing for this layer. Otherwise, it is a to-be-pruned layer. In this way, we can obtain a map, indicating which layers in the network need to be pruned.

4.2.1.1 Input and Output of LSTM

A neural network is a hierarchical sequence from input to output connected by operation nodes, which can be convolution, pooling and fully-connected operation. For a common CNN here, the i^{th} node ξ_i is denoted as (m_i, n_i), where the operation type m is in $\{0, 1, 2\}$ corresponding to convolution, pooling, and fully-connected block respectively, operation attribute n equals to filter number, pooling stride or unit number.

Convolution and fully-connected nodes (final classifier layer is not included) are called the primary nodes, while pooling and final classifier are seen as the secondary nodes because they cannot be pruned but supply auxiliary information instead.

Since LSTM is good at time series prediction, we use a 2-layer LSTM in Fig.4.2 to learn the network structure and produce reasonable pruning decisions. At each timestep, the current primary node as well as its next primary or secondary node $[\xi_i, \xi_{i+1}]$ are fed into LSTM equivalent to $[m_i, n_i, m_{i+1}, n_{i+1}]$ and the pruning decision whether to prune the first primary node is generated by a softmax layer. For a network with N primary nodes, LSTM repeats the above step N times and N distinct softmax layers predict whether to prune these nodes or not. Pooling nodes and the final classifier, taken as the secondary nodes, cannot get pruning predicted but play a key role in helping LSTM to understand a complete network structure.

4.2.1.2 Training LSTM with Policy Gradient Method

After LSTM generates pruning decisions, we prune some filters in the chosen layers such that a slimmer model can be obtained. Both performance and complexity of this new model contribute to the reward signal R for assessing the performance of LSTM. The trade-off is shown in Eq. 4.1, where we use the training loss or accuracy on the validation set to measure the $performance$, and use model FLOPs or the number of PARAM to measure the *complexity*. Let λ be a trade-off hyperparameter, whose optimal value can be obtained empirically via experiments:

$$R = performance - \lambda \times \mathit{complexity}. \tag{4.1}$$

We use the policy gradient algorithm [205] (Eq. 4.2) here, enabling LSTM to generate better pruning strategies. Concretely, we define α_t, s_t and R_k as Action, State and Reward at time step t of one trajectory respectively. m is the number of rollouts for a single gradient update. In order to reduce the variance cuased by sampled trajectories, the reward of the current input network is taken as our baseline b:

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \nabla_\theta \log P(\alpha_t|s_t;\theta)(R_k - b). \tag{4.2}$$

4.2.2 Filter Selection and Fine-tuning Strategy

4.2.2.1 Filter Selection

LSTM generates a decision about which layers should be pruned. Given an input network, a convolution node in layer i can be denoted by a triplet $\langle \mathcal{I}_i, \mathcal{W}_i, \mathcal{O}_i \rangle$, where $\mathcal{I}_i \in \mathbb{R}^{x_{i-1} \times h \times w}$ same as $\mathcal{O}_{i-1}$ is the input tensor with channels x_{i-1}, height h_i and width w_i. The filter tensor $\mathcal{W}_i \in \mathbb{R}^{x_i \times x_{i-1} \times k \times k}$ with $k \times k$ filter size convolutes with $\mathcal{I}_i$ and generates an output tensor $\mathcal{O}_i$ with x_i channels. From the perspective of filters, $\mathcal{W}_i$ consists of x_i filters $\mathcal{F}_i \in \mathbb{R}^{x_{i-1} \times k \times k}$, while from the perspective of channels, $\mathcal{W}_i$ consists of x_{i-1} channel sets $\mathcal{C}_i \in \mathbb{R}^{x_i \times k \times k}$.

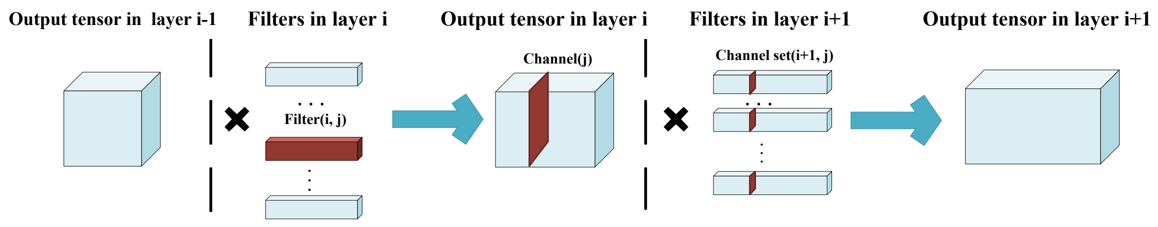


Figure 4.3: Pruning a convolution filter requires removing its corresponding convolution channel set in next layer.

After the j^{th} filter in layer i $\mathcal{F}_{i,j}$ has been pruned, its corresponding j^{th} channel set $\mathcal{C}_{i+1,j}$ becomes useless and should be removed at the same time. Convolution structures in other layers are not affected and remain unchanged as shown in Fig. 4.3. It is the output tensor deviation in layer $i+1$ that transfers errors to the final loss and directly leads to worse performance. Therefore we remove less important filters in layer i and channel sets in layer $i+1$ to minimize the output value deviation $\Delta\mathcal{O}_{i+1}$. Since there often exists an activation, pooling or batchnorm layer between two convolution layers, the channel sets $\mathcal{C}_{i+1}$ affect the output value $\mathcal{O}_{i+1}$ more directly than convolution filters $\mathcal{F}_i$. We follow Eq. 4.3 to measure the importance of each channel set in layer $i+1$ by L2-norm, because L2-norm gives an expectation of the magnitude of the output feature map and reflects the weight diversity. Then $(R_{prune} \times x_i)$ channel sets with smallest score s_j and their responding filters in layer i are selected and removed.

$$s_j = \|(\mathcal{C}_{i+1,j})\|_2, \;\; s.t.\ j \in [1, x_i]. \tag{4.3}$$

4.2.2.2 Accelerated Fine-tuning

In the LSTM training process, there are many intermediate models produced, then they are fine-tuned to calculate reward signals and update LSTM. In order to improve the algorithm efficiency, we use the distillation method [206] to accelerate fine-tuning procedure. Specifically, the input model of LSTM is regarded as a teacher network, and the pruned model based on the teacher is taken as a student network. During fine-tuning, we use the loss function g (Eq. 4.4) to make student's probabilities logit f approximate to teacher's logit z.

$$g(x, z, \theta) = \sum_{x} \parallel f(x, \theta), z \parallel_2^2 . \tag{4.4}$$

4.2.3 Data-dependent Soft Pruning

4.2.3.1 SEP attention module

The slimmer network architecture with the highest reward can be found from the intermediate models generated by LSTM, which becomes the reference of the baseline network rebuilding. The key to the baseline network rebuilding lies in the Squeeze-Excitation-Pruning (SEP) attention mechanism in Fig. 4.4, on which the left part is a normal CNN structure with a few convolution layers while the right part is SEP module. The SEP module consists of two parts: the pre-SE module similar to SENet [137] includes squeeze and excitation used for feature extraction and weight vector generation, the selective-pruning module selects and sets some weights to zero. We apply the SEP operation to the previous conv layer i, then it generates a weight vector to scale the feature map $\mathcal{O}_{i+1}$ in layer $i+1$. There are two differences between SENet [137] and our pre-SE module. Firstly, we perform the SEP operation on the previous conv layer to predict the weight vector in the next conv layer. Secondly, after squeezing and dimensionality-reduction to x_i/r with the reduction ratio r, the dimensionality is increased to x_{i+1} for sake of keeping consistency with the dimensionality of $\mathcal{O}_{i+1}$.

We denote the output of the sigmoid function as $\mathcal{V}_{i+1}$, which is the weight vector in the layer $i+1$, and denote the number of filters to prune as m_{i+1}. The slimmer architecture generated by LSTM determines the value of m_i. If the slimmer module has pruned 36 kernels in the third layer, we set m_3 to 36. Regarding the meaning of the attention mechanism, the larger the weight, the more important it will be. We set some weights with minimum values to zero in Eq. 4.5, because the corresponding feature maps of these weights are of the lowest importance. In Eq. 4.5, the function F_s is a way of using a sorting method to find the m_{i+1}-th minimum weight in $\mathcal{V}_{i+1}$.

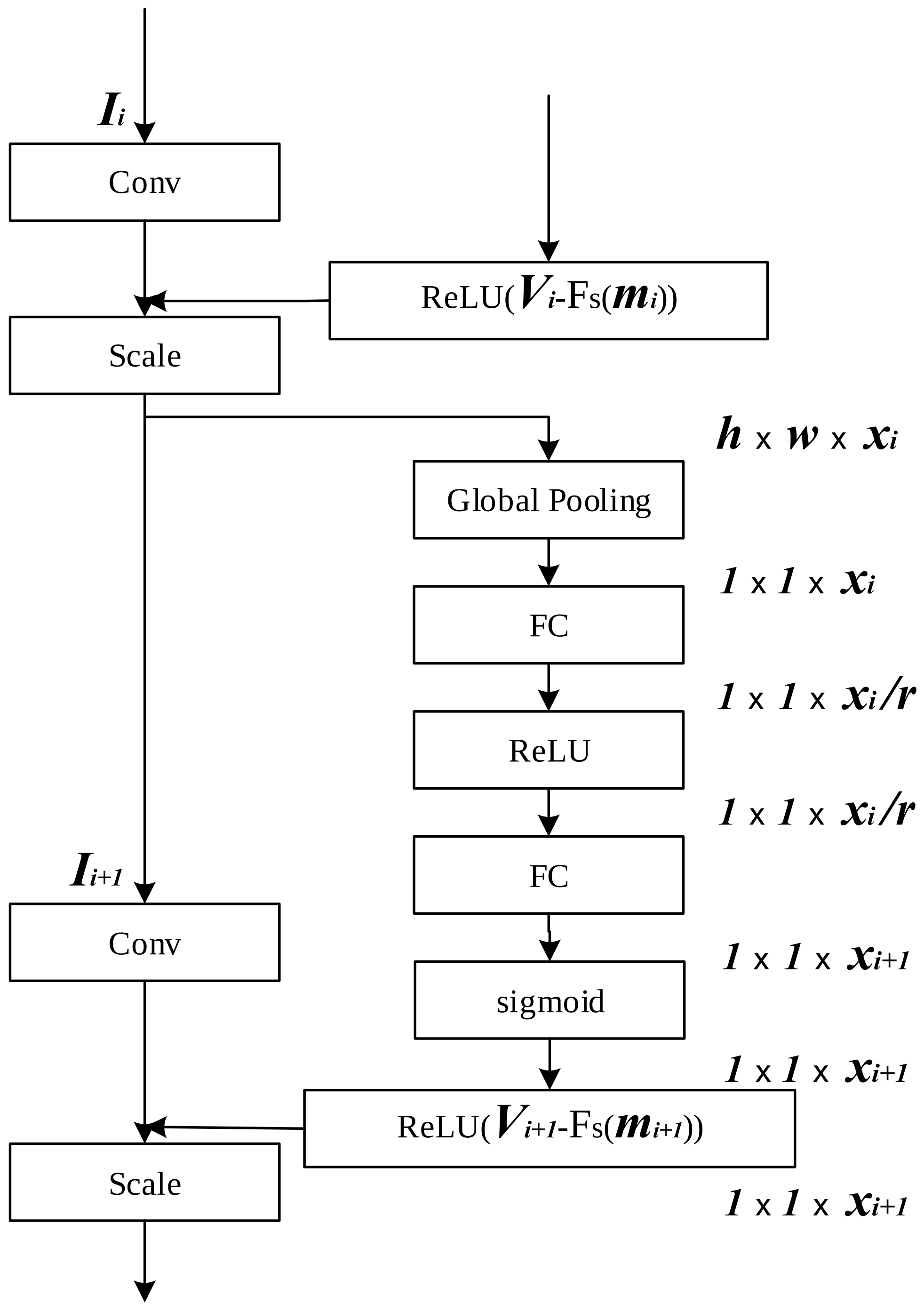


Figure 4.4: The schema of the SEP attention module.

For instance, if the slimmer module decided to prune m_{i+1} kernels in the layer $i + 1$, we sort the values of $\mathcal{V}_{i+1}$ in ascending order. Then we get the m_{i+1}-th smallest value represented by $F_s(m_{i+1})$. The activation function $ReLU$ sets all weights smaller than $F_s(m_{i+1})$ to zero. After the activation operation, we get the sparser vector $\widehat{\mathcal{V}}_{i+1}$, which would scale the feature map x_{i+1}.

$$\widehat{\mathcal{V}}_{i+1} = ReLU(\mathcal{V}_{i+1} - F_s(m_{i+1})). \tag{4.5}$$

4.2.3.2 Architecture Rebuilding

Given the slimmer network architecture generated from LSTM, we deploy the SEP modules on the baseline network to rebuild it. At the core of the SEP module is an attention mechanism, which automatically predicts the importance of each feature map. When the SEP sets some feature maps to zero, it is equivalent to pruning their corresponding kernels. Therefore, when it comes to different image data, different kernels might be utilized according to the SEP selection. All of the kernel parameters are preserved, but only some of which participate in calculating the forward and backward propagations. During the model training, SEP selection strategy is constantly updated, thus indicating this is a dynamic and data-dependent soft pruning procedure. We do not prune the kernels physically, but we preserve all the feature knowledge hierarchy and predict which kernels would be utilized for a specific image data. Although the SEP module introduces extra model complexities, the FLOPs of SEP is negligible because of the fully-connected operation.

In the training process of LSTM, the slimmer model with smaller size and complexity is generated, which can be compared with other hard pruning methods. The SEP algorithm, which is a soft pruning method on the basis of the slimmer architecture generated by LSTM, can also be applied independently, given a predefined slimmer architecture.

4.3 Experiments

We evaluate our method on four benchmark datasets: MNIST [207], CIFAR-10, CIFAR-100 [184] and ImageNet [185]. Two CIFAR datasets contain 50000 training images and 10000 test images. The MNIST contains 60000 and 10000 images for training and testing respectively. In all the datasets, 10% of the images are split from the training set as a validation set used for evaluating new network structures and calculating their reward signals to LSTM. On CIFAR, all images are cropped randomly into 32×32 with four paddings during the training process. Horizontal flip is also adopted. On MNIST, there is no data augmentation preprocessing. In ImageNet, there are over 1.28 million training images and 50k validation images of 1,000 classes.

Three networks: VGGNet [8], ResNet [9] and a 3-layer fully-connected network in [102] are used to validate our method. We employ a 2-layer LSTM with 100 hidden units to make pruning decisions. All the experiments are implemented with PyTorch on one NVIDIA TITAN X GPU.

4.3.1 Implementation Details

We train initial models from scratch and calculate their accuracies as baselines. In the first step for training the LSTM, the pruning rate R_{prune} is set to 0.2, and the teacher instructs the student to fine-tune 30 epochs on CIFAR and 10 epochs on MNIST dataset. LSTM training is terminated when LSTM no longer produces a better network structure within 10 epochs. We retrain the network with the best reward for 250 epochs on CIFAR and 100 epochs on MNIST. For ImageNet, the pruning rate R_{prune} is set to 0.1, and the teacher instructs the student to fine-tune 20 epochs. LSTM training is terminated when LSTM no longer produces a better network structure within 10 epochs. We then retrain the network with the best reward for 40 epochs. Both training and validation datasets are used for retraining the network with the fixed learning rate of 0.001 for ultimate accuracy.

The new parent structure is created based on the combination of current parent structure and the pruning information for each layer generated by LSTM. Then it is added into the list of parent structure for next training epoch. In each epoch of LSTM training, 5 parent structures with the largest rewards in the list are picked up and fed into the LSTM successively for next training epoch. Their rewards are taken as baselines b in policy gradient method. If there are no more than 5 local structures, all local networks are taken as inputs. In the first epoch, the input is the pre-trained network. We use FLOPs to measure the complexity of CNN and PARAM to measure fully-connected networks in order to keep in line with the existing methods.

After getting the slimmer model from LSTM, we rebuild the baseline network to deploy the SEP attention modules. The network redeployed is trained from scratch for 90 epochs on ImageNet and 200 epochs on the rest of datasets to get its final accuracy.

4.3.2 Filter Selection

Our channel-based method is compared with the filter-based methods [35], which evaluate a filter by calculating its absolute weights sum. We prune some filters from a pre-trained VGG-16 on CIFAR-10. Different layers are pruned with the same pruning rate. Then we fine-tune the pruned model for 1 epoch. The experiment is repeated 5 times to eliminate the influence of random disturbance, and we report the averaged accuracy on the test set. Fig. 4.5 shows the pruning results with pruning rate ranging from 0.1 to 0.9 while both methods are set with the same configuration. The results reveal that our channel-based filter selection outperforms the filter-based selection method.

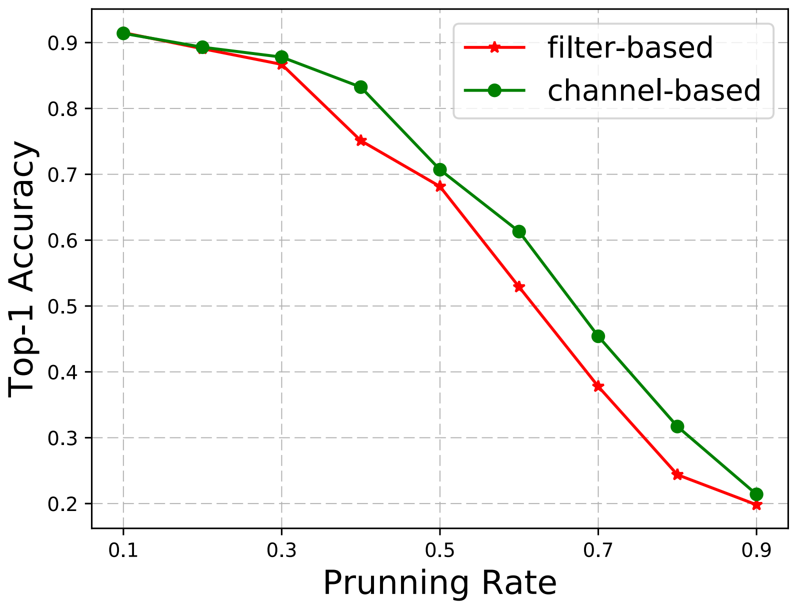


Figure 4.5: Comparison between two methods.

4.3.3 Results

Our layer-selection method based on LSTM is compared to both orderly and global hard filter pruning method. Specifically, on the fully-connected network and VGG, we report the pruning results compared with two global pruning methods [35, 102]. On the ResNet-56, we compare our pruning method with a series of existing pruning methods, including an orderly pruning method [33], CP [109], NISP-56 [208], and FPGM [37]. We also give SEP results to reveal the performance of data-dependent soft pruning.

Table 4.1: Results of VGG-19 on CIFAR-10.

Model	FLOPs	Pruned Rate%	Params	Acc.%
Baseline	3.9×10^8	–	20.04M	93.66
Slimming-4 [35]	8.89×10^7	77.2	3.76M	-0.25
Slimming-5 [35]	4.41×10^7	88.7	2.84M	-1.39
Ours-LSTM	5.98×10^7	84.7	2.92M	-0.36
Ours-scratch	5.98×10^7	84.7	2.92M	-1.74
LSTM-SEP	5.98×10^7	84.7	4.01M	-0.26

"Slimming-N" denotes repeating the slimming method [35] N times.
"Ours-scratch" denotes training the slimmer network generated by LSTM from scratch for 200 epochs.

4.3.3.1 VGG-19 on CIFAR-10

We prune the VGG-19 [8] on the CIFAR-10 dataset. Each convolution layer is followed by a batch normalization layer [209] and we prune its FC layer, which is the last layer before classification .

FLOPs is used as an indicator of model complexity. One multiply-add here is regarded as a floating-point operation unit. We calculate the reward R according to Eq. 4.1 where network's accuracy in validation set represents $performance$, FLOPs represents *complexity* and λ is set to 4×10^{-10}. We summarize the results in Table 4.1 comparing our layer-selective method and SEP method with the global slimming method [35], which selects unimportant filters in all the layers first and then prune all of them simultaneously. "Slimming-N" denotes repeating the slimming method N times. After LSTM is trained for 150 epochs, the optimal structure emerges, whose FLOPs is reduced by 84.7% with only 0.36% accuracy decreased.

We use SEP modules to rebuild the baseline network referring the optimal slimmer structure generated by LSTM, which gets the better performance with only 0.26% accuracy declined. We also train the slimmer network from scratch for 200 epochs, which is equivalent to a hard filter pruning model, and compare it with the SEP rebuilding model. The results show that our soft pruning can maintain higher precision than the hard pruning.

It is worth noting that [35] takes one multiply-add as two floating-point operations, so their calculated FLOPs is two times as much as ours. For a fair and clear comparison, we convert their FLOPs such that it can be in line with ours. It can be observed that our pruned model is more accurate than the pruned model generated by [35](-0.26% vs -1.39%) when the FLOPs are comparable (84.7% vs 88.7%).

Moreover, the number of parameters (Params) is an indication of the memory costs for storing a trained deep model, which is a widely used criterion for evaluating model pruning algorithms. Since extra FC layers are added into the proposed SEP modules, additional parameters are inevitably produced. As a result, though our LSTM-SEP obtains the best accuracy, the model parameters are increased a little bit, compared to Ours-LSTM method without SEP modules. However, from Table I, we still can see that the accuracy of Ours-LSTM is over 1% higher than that of Slimming-5 [35] when the numbers of the parameters of these two trimmed models are similar (2.92M vs 2.84M).

For further investigation, we plot the sensitivity of each layer in the pre-trained VGG-19 in Fig. 4.6. Specifically, at each time we prune one layer while keeping the other layers unchanged, then calculate the accuracy. The results depict that the overall sensitivity distribution keeps the same under different pruning rates and the most sensitive four layers are layer 2, 3, 4, 5. Fig. 4.7, Fig. 4.8 and Fig. 4.9 represent the practical pruning rates of each layer for the optimal network after training LSTM for 50, 150 and 250 epochs respectively. With more training, the real pruning rate from layer 2 to 5 becomes lower and the other layers are pruned more, which is consistent with the observation from Fig. 4.6. The results demonstrate that our method could make reasonable pruning decisions and learn the network sensitivity effectively.

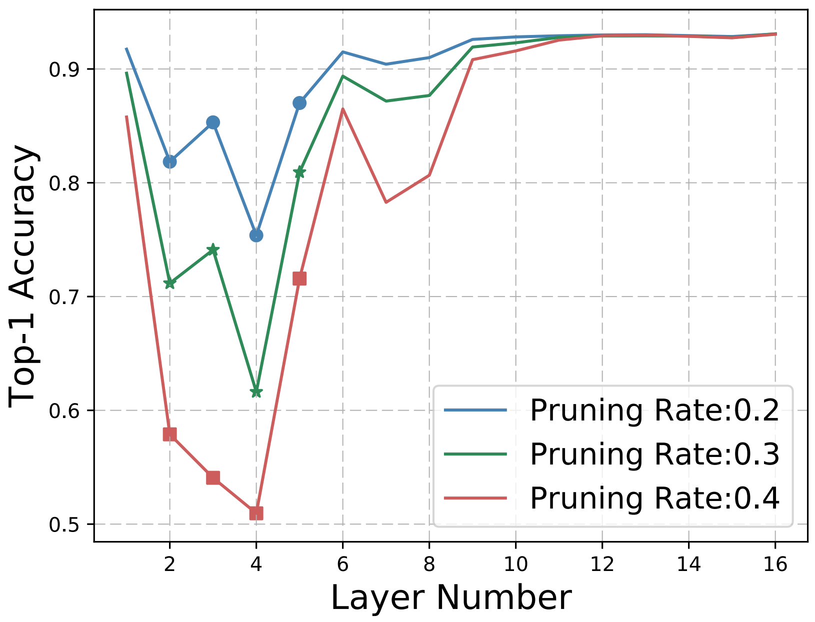


Figure 4.6: Sensitivity of VGG-19 for layers.

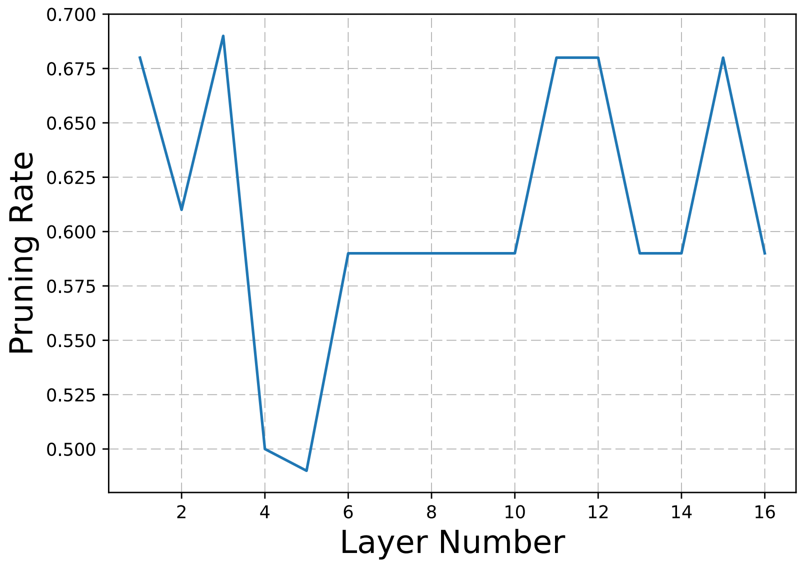


Figure 4.7: Pruning rate within 50 epochs.

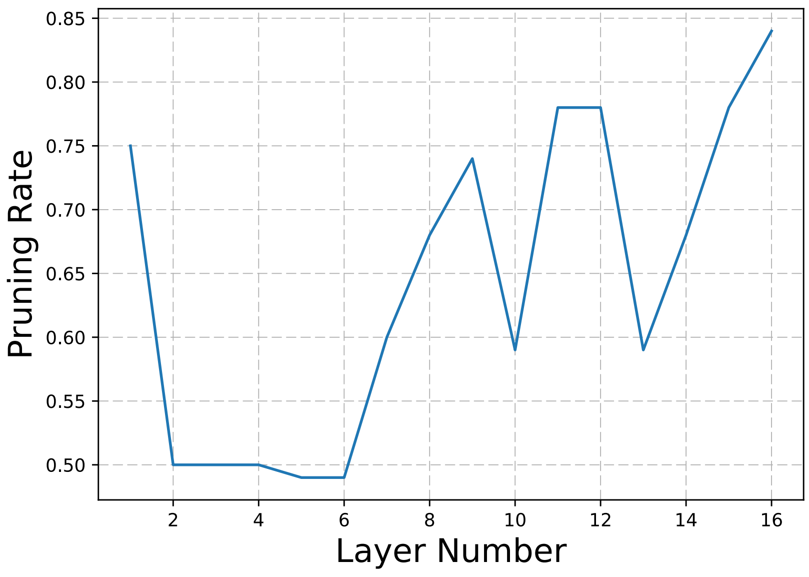


Figure 4.8: Pruning rate within 150 epochs.

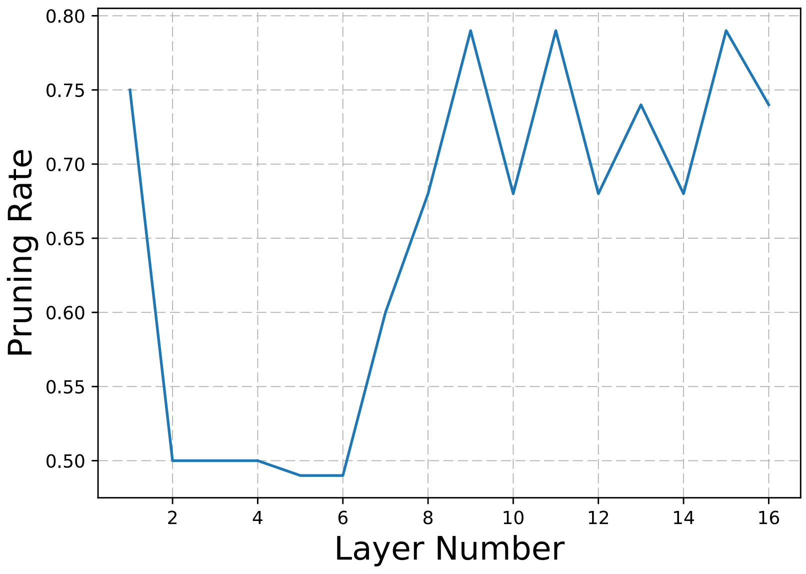


Figure 4.9: Pruning rate within 250 epochs.

4.3.3.2 VGG-19 on CIFAR-100

Table 4.2: Results of VGG-19 on CIFAR-100.

Model	FLOPs	Pruned Rate%	Params	Acc.%
Baseline	3.9×10^8	–	20.04M	73.26
Slimming-3 [35]	1.27×10^8	67.3	5.48M	-2.34
Slimming-4 [35]	6.63×10^7	83	3.27M	-3.85
Ours-LSTM	1.17×10^8	70.1	4.86M	+0.0
Ours-scratch	1.17×10^8	70.1	4.86M	-3.5
LSTM-SEP	1.17×10^8	70.1	5.95M	+0.31

We use the same VGG-19 network to evaluate our method on CIFAR-100. Due to more categories, CIFAR-100 is much more difficult to train than CIFAR-10. Thus, the training and validation set are both used to fine-tune the pruned model. Here we use the training loss to evaluate $performance$, and set λ to 2×10^{-11} in Eq. 4.1. After training LSTM for 123 epochs, we get the best network whose FLOPs is reduced by 70.1% with no accuracy drop. The SEP rebuilding model even improves the accuracy by 0.31%, which indicates the superiority of the SEP module. The attention mechanism improves the model performance. In the meantime, the soft pruning manner retains the capacity of the baseline network, thus maintaining the accuracy to the fullest. As can be seen from Table 4.2, our method outperforms the slimming method significantly (above 3%), even though the number of parameters is a little bit higher than that of Slimming-4 [35].

4.3.3.3 ResNet-56 on CIFAR-10

In this section, we verify the feasibility of our method on ResNet-56 [9]. Due to the particularity of ResNet structure, we only prune the first convolution layer of each ResNet block and keep the second convolution layer unchanged. The parameter configuration of Eq. 4.1 is the same as the VGG-19 experiment on CIFAR-10.

Table. 4.3 reports our results compared to [33, 37, 109, 208], which analyze the sensitivity of each ResNet block first, then prune filters referring to the analysis results. Note that all the results of existing algorithms are collected from their original publications. We do not make any analysis in advance because our method is capable of automatically learning the network sensitivity. After LSTM is trained for 32 epochs, the best network emerges with 47.5% FLOPs reduction and comparable accuracy. Compared to [33], more filters are pruned with acceptable accuracy decrease of 0.11% so that we get the model with the minimum number of parameters, which is only 0.49M. For further comparison, we select the second-best structure with less FLOPs reduction, and it achieves a notable 0.56% accuracy promotion. The SEP models based on these two slimmer architectures present more significant performance. Even if compared to a recent method [37], we still obtain the promising results - the accuracy of our best slimmer model exceeds that of their slimmer model by 1.47% when the FLOPs of both models are equivalent.

In Fig.4.10, we draw the maps of soft-pruned filters distribution in the last and second-to-last residual block respectively to show the data dependence of the SEP method. We run the CIFAR-10 testset on LSTM-SEP-2 model to check which filters are discarded for different image data in a specific layer. The x-axis represents the filter index, while the y-axis represents the number of images on which this filter is discarded. The last and the second-to-last residual block both consist of 64 filters. The CIFAR-10 testset concludes 10000 images. As can be seen in Fig.4.10, some filters(red bar) are always discarded on all the images but some (green bar) are discarded on specific images. Some filters are always utilized because they play a significant role for all data. SEP selects different filters for different images, which results reveal that the SEP has data-dependent nature, and has the ability to select different filters for different images.

Table 4.3: Results of ResNet-56 on CIFAR-10.

Model	FLOPs	Pruned Rate%	Params	Acc.%
Baseline	1.25×10^8	–	0.86M	93.04
PFEC-A [33]	1.12×10^8	10.4	0.77M	+0.06
PFEC-B [33]	9.09×10^7	27.6	0.61M	+0.02
CP [109]	–	50.6	–	-1.00
NISP-56 [208]	–	43.6	–	-0.03
SFP [30]	5.94×10^7	52.6	0.51M	-1.33
FPGM [37]	5.94×10^7	52.6	0.51M	-0.66
Ours-LSTM-1	8.24×10^7	34.1	0.56M	+0.56
Ours-scratch-1	8.24×10^7	34.1	0.56M	-0.85
LSTM-SEP-1	8.24×10^7	34.1	0.62M	+1.01
Ours-LSTM-2	6.56×10^7	47.5	0.49M	-0.11
Ours-scratch-2	6.56×10^7	47.5	0.49M	-0.87
LSTM-SEP-2	6.56×10^7	47.5	0.62M	+0.81

4.3.3.4 A Fully-connected Network on MNIST

We further validate the effect of our method on multi-layer perceptrons. We prune a 3-layer fully-connected network compared with two global pruning methods [35, 102]

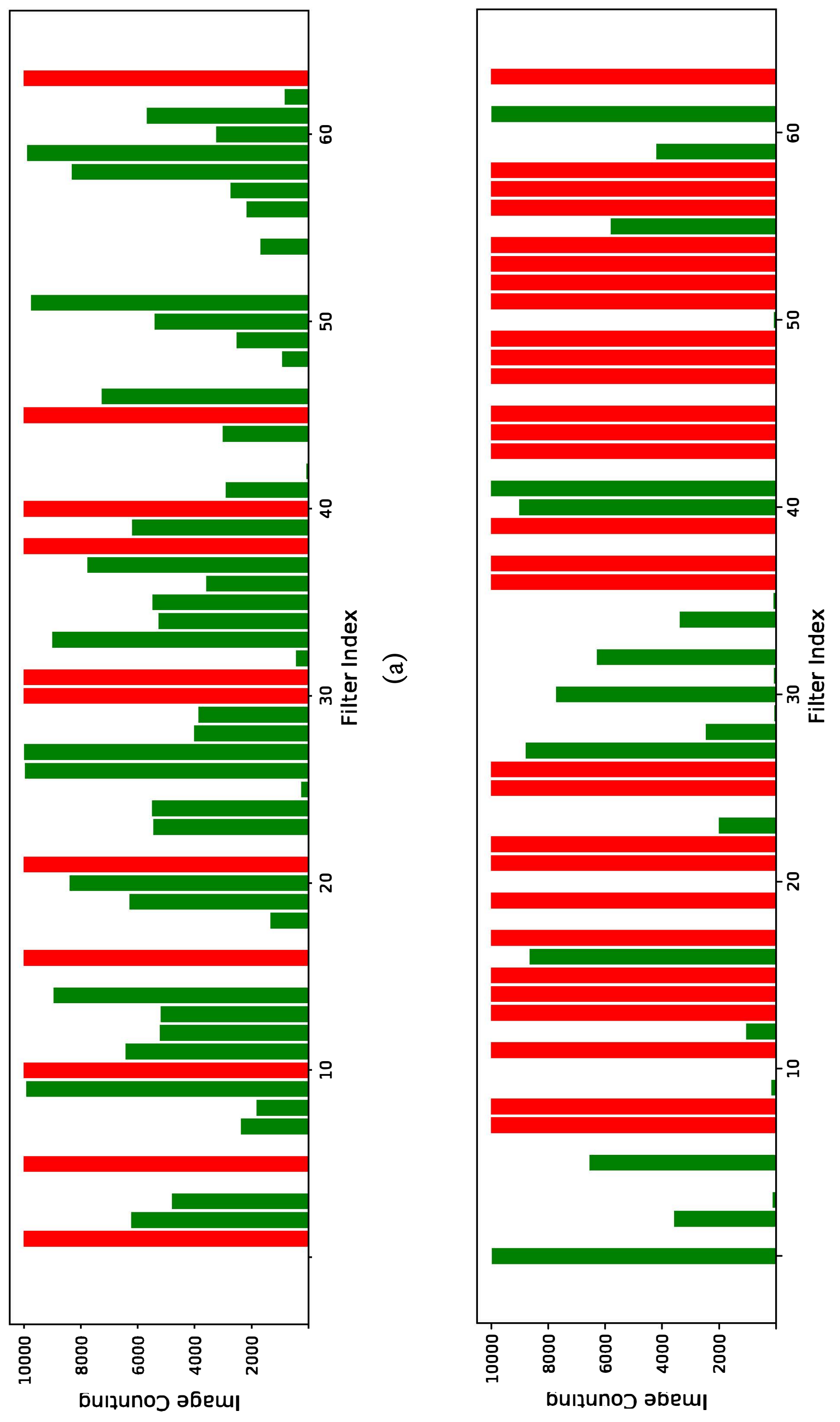


Figure 4.10: The soft-pruned filters distribution of the last (a) and the second-to-last (b) residual block in LSTM-SEP-2 model based on ResNet-56 on CIFAR-10 testset. Some filters(red bar) are always discarded on all the images but some (green bar) are discarded on specific images.

as shown in Table 4.4. Similar to CNN, the evaluation of neurons in the current FC layer depends on its next FC layer. Here we use the accuracy on the validation set to measure $performance$. We set λ to 1×10^{-7}. After 20 epochs, the optimal network structure emerges with 87% neurons pruned and 0.03% accuracy drop.

Table 4.4: Results of a Fully-connected Network on MNIST.

Model	Pruned%	Acc.%	#Neurons
Baseline	–	98.57	784-500-300-10
Structured sparsity [102]	83.5	-0.11	434-174-78-10
Slimming-1 [35]	84.4	-0.06	784-100-60-10
Ours-LSTM	87.26	-0.03	784-83-48-10

4.3.3.5 ResNet-50 on Imagenet

Table 4.5: Results of ResNet-50 on ImageNet.

Model	baseline Top-1 Acc.%	baseline Top-5 Acc.%	Pruned Rate%	Params	Top-1 Acc.%	Top-5 Acc. %
SFP [30]	76.15	92.87	41.8	16.96M	-1.54	-0.81
FPGM [37]	76.15	92.87	42.2	16.96M	-1.12	-0.47
CFP [210]	75.30	92.20	49.6	–	-1.90	-0.80
CP [109]	–	92.20	50.0	–	–	-1.40
GDP [21]	75.13	92.30	51.3	–	-3.24	-1.59
Ours-LSTM	76.12	93.00	43.0	15.96M	-1.12	-0.33
LSTM-SEP	76.12	93.00	43.0	17.18M	-0.90	-0.27

In this section, we verify the performance of our method on ImageNet, which is a large-scale dataset. Since ResNet-50 [9] structure is commonly used by many pruning methods such as SFP [30], CP [109], GDP [21], we choose it to conduct the pruning experiments for a fair comparison. Specifically, we prune the first and second convolution layers of each ResNet block and keep the third convolution layer unchanged. The parameter configuration of Eq. 4.1 is the same as the VGG-19 and ResNet-56 experiments on CIFAR-10.

Table. 4.5 shows that our methods can achieve competitive performance, compared to state-of-the-art methods including SFP [30], FPGM [37], CFP [210], CP [109], GDP [21]. Note that we collect their results from the original publications, and no pre-trained models are used. Seen from the results, the performance of LSTM-SEP is better than that of Ours-LSTM, which shows that our soft pruning can maintain higher accuracy than the hard pruning. Although the pruning rate of our methods is slightly lower than that of CFP [210], CP [109] and GDP [21], the accuracies of Ours-LSTM and LSTM-SEP are much higher than them. Particularly, the accuracy of LSTM-SEP exceeds CFP [210] and GDP [21] models by 1.00% and 2.34% respectively and the Top5 accuracies of our two methods are 1.00% higher than that of CP [109] and GDP [21]. Besides, in comparison to SFP [30] and FPGM [37] that also prune filters in a soft manner, LSTM-SEP reduces more FLOPs of the model with even less accuracy drops. Overall, it is clear that our method outperforms the state-of-the-art soft pruning methods.

4.4 Chapter Summary

In this chapter, a novel filter pruning framework is presented to evaluate the importance of each network layer and thus generate the specific pruning decision for each one. Besides, we propose the SEP pruning method rather than adopting a hard pruning strategy to prune redundant filters accurately. SEP is a data-dependent soft pruning method, which preserves all the filter parameters, but for each image data, only some important ones participate in calculating forward and backward propagations. Different filters may be softly pruned when the given image is changed according to the SEP selection. Experiment results show that our pruning method is capable of compressing a variety of network structures with comparable accuracy and works well on both convolutional and fully-connected networks. It also reveals that our method learns the sensitivity of each network layer.

In the next chapter, we transfer our view from the filters to the feature maps. Feature maps contain more comprehensive information than filters since they are generated after the convolution operation by filters and input data. We further transform the encoded features into the frequency domain by DCT to efficiently extract valuable and concentrated information from the feature maps. This way, the proposed method can effectively select redundant filters to reduce network complexity while maintaining its performance to the largest extent. The details of the proposed method will be revealed in the following chapter.

Chapter 5

Filter Pruning with Uniqueness Mechanism in the Frequency Domain

5.1 Introduction

The rapid progress in Convolutional Neural Networks (CNNs) has revolutionalized various computer vision tasks, e.g., image classification [86,211], object detection [212–214], and segmentation [193, 215, 216]. To pursue better performance, most methods resort to complex network architectures. However, such implementations require heavy computations and memory footprints, limiting their applications in resource-limited systems (e.g., embedded or mobile devices). Therefore, how to compress complex networks while preserving competitive performance has recently drawn much attention.

The existing techniques for network compression can be mainly grouped into four categories: knowledge distillation [54, 217], parameter binarization [69, 71], compact network design [177, 218], and network pruning [118, 131]. Unlike the former three techniques, which build lightweight networks from scratch or change the parameter storage types, network pruning methods achieve compactness by finding and removing redundant parameters from the off-the-shelf networks. By doing so, the pruned networks can retain the most knowledge from the original networks while reducing the requirements for memory and computation resources. In general, there are two techniques to achieve network pruning: weight pruning [88, 92, 95] and filter pruning [30, 33, 41]. Weight pruning methods reduce network parameters by seeking a sparse weight matrix. However, the network slimmed down by those methods is unstructured as each parameter is likely to be removed, which disorders the integrity

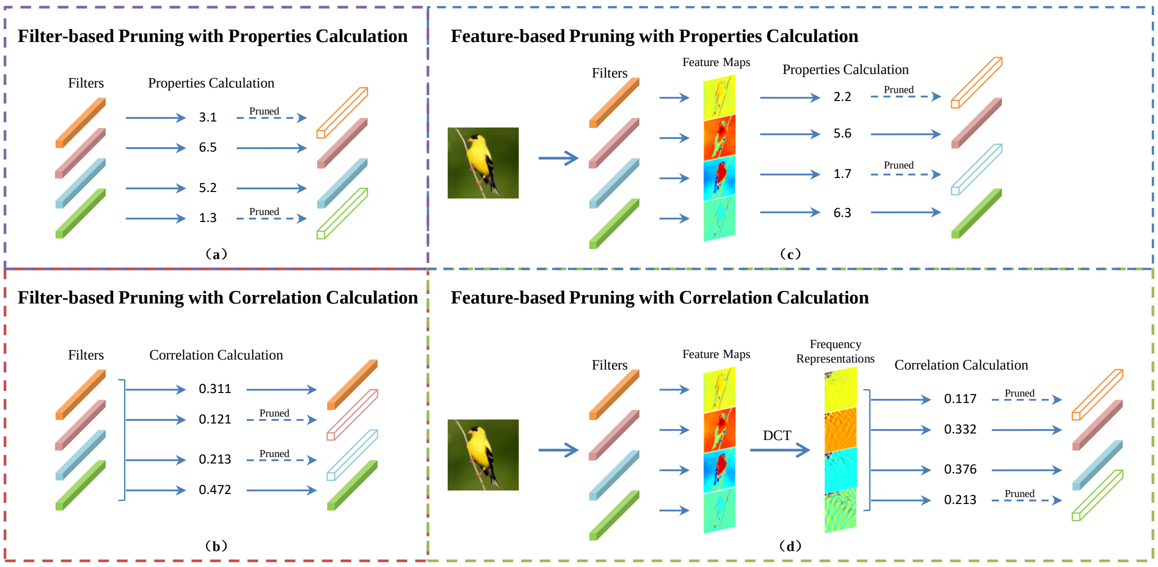


Figure 5.1: Four different pruning methods.

of the network components and makes it unable to achieve optimal efficiency via the Basic Linear Algebra Subprograms (BLAS) library [30]. By contrast, the latter methods prune all the related parameters once one filter is unimportant. Therefore, the pruned network still consists of integral components and can fully leverage the BLAS library for accelerated training and inference.

Due to its excellent efficiency, filter pruning has become prevalent in network compression. One intuitive attempt is to prune filters based on their importance measured by their intrinsic properties. For example, the method shown in Figure 5.1 (a) only considers the norm value to determine its importance. Specifically, it removes the filters whose norms are below a threshold. However, since the correlation between filters is ignored, one type of redundancy still exists in the pruned network: the filters that are prone to be replaced by others, even though their norm values are high enough. To address this issue, some methods [37, 41] take the correlation into account, as shown in Figure 5.1 (b). Despite achieving better performance, these methods mainly focus on filter weights but ignore more comprehensive information generated by the filters: feature maps.

Unlike the filter-based pruning methods, Lin *et al.* [39] considered the rank of each feature map to measure the corresponding filter's importance. As shown in Figure 5.1 (c), the method performs pruning with the properties derived from feature maps. With more comprehensive guidance, the feature-based methods outperform the filter-based ones. However, there is still room to improve them further: (1)

For each feature map, current methods only compute the rank based on its intrinsic properties, ignoring the correlation between maps; (2) For visual tasks, low-frequency channels are generally more informative than high-frequency ones [162], and almost all the energy in spatial features is concentrated in the low-frequency spectrum after DCT. Therefore, in contrast to the spatial domain, it seems easier and more efficient to find unimportant filters in the frequency domain.

Inspired by the above observations, we measure the filter importance based on the frequency-domain correlation between features. Fig. 5.1 (d) illustrates the basic idea of our method. For each feature map, we compute the *uniqueness* to indicate whether it consists of unique enough information which is not easily replaced by others. Since the *uniqueness* comes from the interaction between feature maps, our method is more robust against the interference from intrinsic properties, e.g., norm values. With the uniqueness and the effective convergence of DCT, our proposed method can prune various networks with complex architectures. Specifically, we first transform the feature maps into the frequency domain by DCT. Then, we compute the uniqueness for each feature map and remove the filters corresponding to the low-uniqueness maps. The remaining filters form the final network with less complexity and performance preserved. Fig. 5.2 illustrates the framework of our proposed method, and our main contributions are summarized as follows:

1) We propose uniqueness, a novel criterion for filter pruning. Unlike intrinsic properties of filters, uniqueness is measured from the correlation between feature maps. It implicitly indicates how much and how unique a feature map embeds the critical information. Therefore, a more comprehensive pruning strategy can be achieved.

2) We propose to determine to-be-pruned filters in the frequency domain. With the advantages of the frequency-domain operations, our proposed method can find and prune unimportant filters more efficiently, without much interference as in the spatial domain.

3) The extensive experiments involving various network architectures and two different scales of image datasets demonstrate that our proposed method outperforms the state-of-the-art in accuracy and model compression.

We elaborate the proposed FPUM in Section 5.2. Experimental results along with ablation study are provided in Section 5.3. Finally, the proposed method is concluded in Section 5.4.

5.2 Methodology

5.2.1 Preliminaries

Given a CNN model with L layers, let $\boldsymbol{W}^l \in \mathbb{R}^{c^l \times c^{l-1} \times k \times k}$ be the filter tensor of the l-th convolutional layer, where k, c^l, and c^{l-1} represent the kernel size, output channel, and input channel of the filter, respectively. The filter-based pruning methods generally formulate the objective function as follows:

$$\begin{aligned} \min_{k_l} = \sum_{l=1}^{L}\sum_{j=1}^{c^l} k_l E(\boldsymbol{W}_j^l) \\ \text{s.t. } 0 < \|k_l\|_0 \leq (1-\alpha^l)c^l, \end{aligned} \quad , \tag{5.1}$$

where k_l is a list of indices, indicating the filters to prune in the l-th layer. The amount of the pruned filters $\|k_l\|_0$ is limited by both the compression ratio α^l and the initial number c^l. $E(\cdot)$ estimates the intrinsic property of each filter $\boldsymbol{W}_j^l$, which implicitly measures the importance of the filter and, therefore, can serve as a criterion for pruning. In more recent pruning methods, however, feature maps have gradually dominated the criterion measurement since they embed more comprehensive information regarding filters and input data. In this case, the above objective function can be rewritten as the following form:

$$\begin{aligned} \min_{k_l} = \sum_{l=1}^{L}\sum_{j=1}^{c^l} k_l E(\boldsymbol{I}^l * \boldsymbol{W}_j^l) \\ \text{s.t. } 0 < \|k_l\|_0 \leq (1-\alpha^l)c^l, \end{aligned} \quad , \tag{5.2}$$

where $\boldsymbol{I}^l$ is the input tensor to the l-th layer and $*$ denotes the convolution operation. So far, several prior works [30, 37, 39] have been implemented to prune unimportant filters through the above optimization processes. The main focus of these methods usually lies in designing the function $E(\cdot)$ for importance measurement.

5.2.2 Uniqueness Calculation in the Image Domain

Unlike the previous works, which focus on spatial operations and prune unimportant filters using their intrinsic properties or feature maps, we perform pruning from a novel view: the frequency domain. Specifically, we design an effective mechanism to measure the uniqueness of each filter, based on the correlation between its corresponding feature map and others in the frequency domain. With the uniqueness, we can

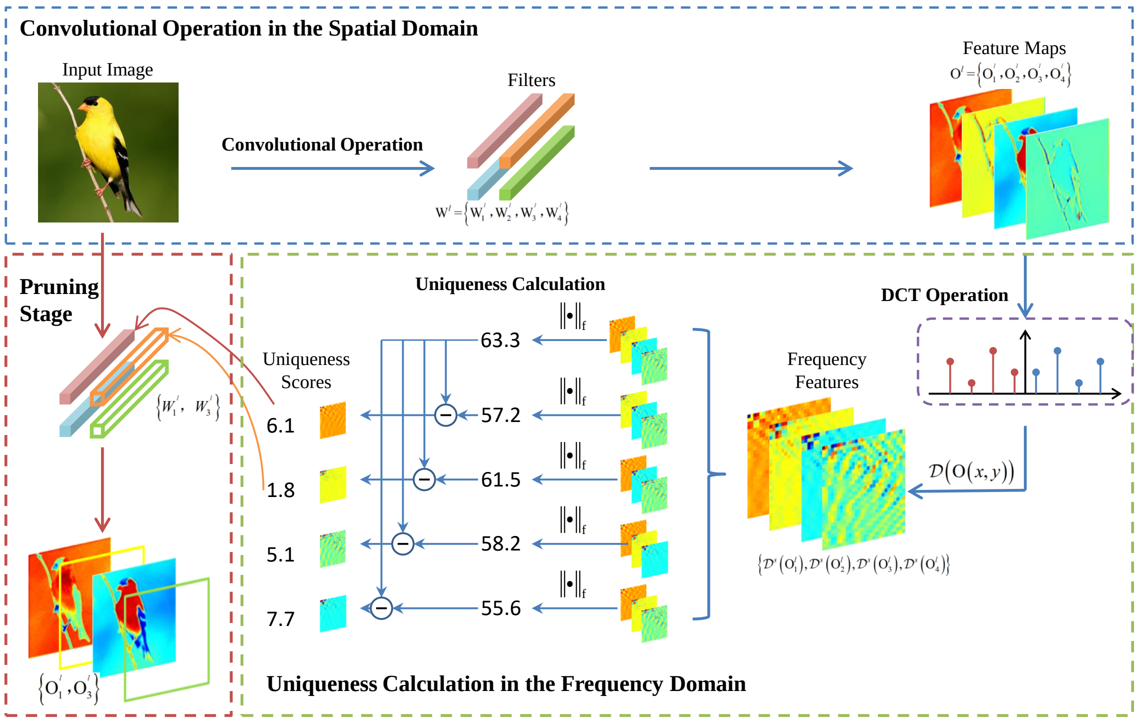

Convolutional Operation in the Spatial Domain
Input Image
Convolutional Operation
Filters
$W^l=\{W_1^l,W_2^l,W_3^l,W_4^l\}$
Feature Maps
$O^l=\{O_1^l,O_2^l,O_3^l,O_4^l\}$
Pruning Stage
$\{W_1^l, W_3^l\}$
$\{O_1^l,O_3^l\}$
Uniqueness Calculation
Uniqueness Scores
6.1
1.8
5.1
7.7
63.3
57.2
61.5
58.2
55.6
$\|\cdot\|_f$
DCT Operation
Frequency Features
$\mathcal{D}(O(x,y))$
$\{\mathcal{D}^l(O_1^l),\mathcal{D}^l(O_2^l),\mathcal{D}^l(O_3^l),\mathcal{D}^l(O_4^l)\}$
Uniqueness Calculation in the Frequency Domain

Figure 5.2: The framework of our proposed method. At first, we encode features in the spatial domain, where each feature map comes from the convolution between the input image and one filter (shown in the blue box, the top region). Then, these features are transferred to the frequency domain by DCT, which are in turn used to compute the uniqueness score for each feature map (shown in the green box, the bottom-right region). Finally, the filters corresponding to the low-uniqueness feature maps will be pruned and will not participate in the subsequent computation (shown in the red box, the bottom-left region). Best viewed in color.

infer if one filter can be replaced by others. Compared with intrinsic properties (e.g., norm values), it better reflects the filter redundancy. Therefore, the over-pruning or under-pruning due to inaccurate redundancy measurement can be effectively alleviated.

We first introduce the uniqueness in the spatial domain. For simplicity, we assume $\boldsymbol{O}^l = \boldsymbol{I}^l * \boldsymbol{W}^l \in \mathbb{R}^{c^l\times h^l\times w^l}$ as the feature maps generated by the l-th convolutional layer, ignoring the activations and biases. c^l, h^l, and w^l indicate the channel, height, and width dimensions, respectively. For each feature map $\boldsymbol{O}^l_j \in \mathbb{R}^{h^l\times w^l}$, its uniqueness can be defined as:

$$E(\boldsymbol{O}^l_j) = \|\boldsymbol{O}^l\|_f - \|\boldsymbol{O}^l_{j*}\|_f, \tag{5.3}$$

where $\|\cdot\|_f$ computes the Frobenius norm. We reshape feature maps $\boldsymbol{O}^l$ and $\boldsymbol{O}^l_{j*}$ to

$c^l \times h^l w^l$ to meet the requirement for the input dimensions. $\boldsymbol{O}^l_{j*}$ is initialized from $\boldsymbol{O}^l$ and all its values in the j-th row are set to zero after dimension transformation.

5.2.3 Uniqueness Calculation in the Frequency Domain

Given the efficient energy convergence of DCT, it is easier to determine the uniqueness of each feature map in the frequency domain than in the spatial domain since almost all the essential energy in spatial features is concentrated in the low-frequency spectrum after DCT. Doing so can eliminate noise interference in the space domain and extract more useful information from the original feature map. Besides, we only need to process a small-size of feature map rather than the entire one to obtain an accurate pruning strategy, thus reducing the computational complexity. Therefore, we transform the feature maps into the frequency domain through DCT for uniqueness computation, which is defined by reformulating Eq. 5.3 as the following form:

$$E(\mathcal{D}(\boldsymbol{O}^l_j)) = \|\mathcal{D}(\boldsymbol{O}^l)\|_f - \|\mathcal{D}(\boldsymbol{O}^l_{j*})\|_f, \tag{5.4}$$

where $\mathcal{D}(\cdot)$ denotes the DCT operation. For each position (x, y) in the j-th feature map $\boldsymbol{O}^l_j$, its counterpart in the frequency domain is obtained as below:

$$\mathcal{D}(\boldsymbol{O}^l_j(x,y)) = \frac{s}{\sqrt{h^l w^l}} \sum_{x=0}^{h^l-1} \sum_{y=0}^{v-1} \boldsymbol{O}^l_j(x,y) \cos(\frac{\pi}{h^l} u(x+\frac{1}{2})) \cos(\frac{\pi}{w^l} v(y+\frac{1}{2})), \tag{5.5}$$

where (u, v) indicates the location in the frequency domain. s is a scaling factor and conditioned on (u, v):

$$\begin{cases} s = 1, \text{ if } (u,v) = (0,0) \\ s = 2, \text{ if } (u,v) \neq (0,0). \end{cases} \tag{5.6}$$

After DCT, most low-frequency information will be concentrated in a local region in the frequency domain, which comprises the most informative knowledge of the input data. Therefore, such a local area can support robust and more efficient data processing compared with the whole region. We further reformulate Eq. 5.4 to measure the uniqueness of the j-th filter in the l-th layer:

$$\mathrm{E}(\mathcal{D}^s(\boldsymbol{O}^l_j)) = \|\mathcal{D}^s(\boldsymbol{O}^l)\|_f - \|\mathcal{D}^s(\boldsymbol{O}^l_{j*})\|_f, \tag{5.7}$$

where $\mathcal{D}^s(\cdot)$ only focuses on the frequency-domain features from the concentrated local region after DCT. For the j-th filter in the l-th layer, its importance is evaluated by $E(\mathcal{D}^s(\boldsymbol{O}^l_j))$, the correlation of its corresponding feature map with others in the frequency domain. In this way, we can prune filters more accurately and efficiently if we replace the original criterion ($E(\boldsymbol{W}_j^l)$) in Eq. 5.1 with $E(\mathcal{D}^s(\boldsymbol{O}_j^l))$. After obtaining the uniqueness score of each filter, the number of removable filters is determined based on the pruning rate for each layer. Therefore, we prune the filter according to its uniqueness score, from small to large, until the number of pruning requirements is reached.

5.3 Experiments

In this section, we provide extensive experimental results and analysis to illustrate the superior performance of our algorithm on two different scales of datasets: CIFAR-10 [184] and ImageNet [219].

5.3.1 Experimental Settings

5.3.1.1 Baselines and Datasets

To evaluate our proposed method comprehensively, we apply it to various CNN architectures and compare the pruning performance with state-of-the-art methods on different datasets. Specifically, we first consider two ResNet architectures [9] (ResNet-56 and ResNet-110) and VGG-16 [8] on a small benchmark dataset: CIFAR-10 [184]. Then, we conduct experiments with ResNet-50 on a larger dataset (ImageNet [219]) for further analysis.

5.3.1.2 Evaluation Metrics

Similar to existing methods, we utilize the number of floating-point operations (FLOPs) and parameters (Params) to measure the complexity and size of the pruned CNN models, respectively. As for the accuracy evaluation, we compute models' Top-1 accuracy on CIFAR-10 and Top-1 and Top-5 accuracy on ImageNet. It is worth noting that ΔTop-1(%) and ΔTop-5(%) represent the difference in Top-1 and Top-5 accuracies before and after pruning. Params(%)$\downarrow$ and FLOPs(%)$\downarrow$ indicate the drops (percentage) in parameters and FLOPs between the pruned and baseline models. The results of competitors shown in Tables 5.1, 5.2, 5.3, and 5.4 are reported from their original publications.

5.3.1.3 Implementation Details

During the fine-tuning stage on CIFAR-10, we set the number of epochs as 300 and batch size as 128, as employed by the competitors, to make a fair comparison. We also consider Stochastic Gradient Descent (SGD) as the optimizer with an initial learning rate, weight decay, and momentum of 0.01, 0.05, and 0.9, respectively. As for the experiment setting on ImageNet, we retrain the pruned network for 180 epochs with a batch size of 256. The initial learning rate, weight decay, and momentum are set as 0.1, 0.0001, and 0.9, respectively. All the experiments are implemented with PyTorch on 4 NVIDIA GTX1080Ti GPUs.

5.3.2 Results and Analysis

5.3.2.1 Results on CIFAR-10

Table 5.1: Comparison of pruned ResNet-56 on CIFAR-10.

Method	**Top1(%)**	**ΔTop1(%)**	**Params(%)$\downarrow$**	**FLOPs(%)$\downarrow$**
DNAL [220]	94.15 → 93.76	-0.39	22.3	25.1
DNAL [220]	94.15 → 93.75	-0.40	33.8	30.6
NISP [208]	94.15 → 93.01	-0.14	42.4	35.5
GAL [192]	93.26 → 93.38	+0.12	11.8	37.6
Ours	**93.26 → 93.88**	**+0.62**	**42.8**	**47.4**
LSTM [221]	93.04 → 92.93	-0.11	N/A	47.5
CP [109]	92.80 → 90.90	-1.90	N/A	50.0
FPSST [222]	93.57 → 93.28	-0.29	N/A	51.1
SFP [30]	93.59 → 92.26	-1.33	N/A	52.6
FPGM [37]	93.59 → 92.93	-0.66	N/A	52.6
GAL [192]	93.26 → 91.58	-1.68	65.9	60.2
DNAL [220]	94.15 → 93.20	-0.95	70.5	70.5
Hrank [39]	93.26 → 90.72	-2.54	68.1	74.1
Ours	**93.26 → 92.48**	**-0.78**	**71.8**	72.3

In this section, we firstly deploy ResNet-56 to verify the performance of our proposed method on CIFAR-10. As shown in Table 5.1, our method obtains the best performance compared to state-of-the-art methods on both moderate and deep compression. Specifically, the Top-1 accuracy of our method is 0.49% higher than LSTM [221] with almost the same FLOPs reduction. Moreover, our results are even better than the baseline model (93.88% vs. 93.26%). Compared with NISP [208], DNAL [220], and GAL [192], we obtain a better performance in Top-1 accuracy, parameter reduction, and FLOPs reduction. Although CP [109], SEP [30], and FPGM [37] reduce more FLOPs than ours, their Top-1 accuracy is much lower than ours (-2.52%, -1.95%, -1.28%). For deep compression, our method achieves the best Top-1 accuracy while obtaining the maximum parameter reduction and FLOPs reduction compared with GAL [192] and DNAL [220]. Besides, our method achieves better quantitative results (92.48% vs. 90.72%) even though we share a similar model compression with Hrank [39].

Table 5.2: Comparison of pruned ResNet-110 on CIFAR-10.

Method	**Top1(%)**	**ΔTop1(%)**	**Params(%)↓**	**FLOPs(%)↓**
SFP [30]	93.68 → 93.38	-0.30	N/A	40.8
Rethink [191]	93.77 → 93.70	-0.07	N/A	40.8
Hrank [39]	93.50 → 94.23	+0.73	39.4	41.2
GAL [192]	93.50 → 92.55	-0.95	44.8	48.5
FPSST [222]	93.70 → 93.62	-0.08	N/A	50.6
Ours	**93.50 → 94.51**	**+1.01**	**48.3**	52.1
FPGM [37]	93.68 → 93.74	+0.06	N/A	52.3
FalCon [223]	93.68 → 93.79	+0.11	N/A	60.3
FalCon [223]	93.68 → 93.63	-0.05	N/A	62.3
CCPrune [224]	94.11 → 93.36	-0.75	N/A	68.0
Hrank [39]	93.50 → 92.65	-0.85	68.7	68.6
Ours	**93.50 → 93.73**	**+0.23**	68.3	**71.6**

Analogous to ResNet-56, our method achieves excellent performance on ResNet-110. From Table 5.2, it can be seen that we obtain the best accuracy improvement with the most significant parameter and FLOPs reductions, compared to SEP [30], Rethink [191], HRank [39] and GAL [192]. Although FPGM [37] obtains a similar FLOPs reduction as ours, it refines the accuracy by +0.06% only, which is much lower than ours (+1.01%). In addition to moderate compression, we observe that our method with deep compression outperforms the existing competitors in both Top-1 accuracy and FLOPs reduction, further illustrating the consistent superiority of our proposed method.

Finally, we employ VGG-16 to verify the performance of our proposed method on CIFAR-10, as shown in Table 5.3. Not surprisingly, our method achieves better performance with deep compression. Compared to PEFC [33], FPMG [37], SSS [105], GAL [192], and Hrank [39], our method has apparent advantages in Top-1 accuracy and FLOPs compression ratio. To be specific, our method achieves up to 73.7% FLOPs reduction with only a 0.2% accuracy loss (from 93.96% to 93.76%). Although RL-MCTS [225] and FalCon [223] further improve the Top-1 accuracy, their FLOPs reductions (45.5% and 50.0%) are much lower than ours (73.7%). Moreover, our method achieves fewer Top-1 accuracy loss than HRank [39](-0.35% vs.-2.73%) even though we yield approximately the same FLOPs reduction. These results illustrate the effectiveness of our proposed method.

Table 5.3: Comparison of pruned VGG-16 on CIFAR-10.

Method	**Top1(%)**	**ΔTop1(%)**	**Params(%)↓**	**FLOPs(%)↓**
PFEC [33]	93.58 → 93.28	-0.30	N/A	34.2
FPGM [37]	93.58 → 93.23	-0.35	N/A	35.9
SSS [105]	93.96 → 93.02	-0.94	73.8	41.6
GAL [192]	93.96 → 93.42	-0.54	82.2	45.2
RL-MCTS [225]	93.51 → 93.90	+0.39	N/A	45.5
FalCon [223]	93.32 → 93.63	+0.31	N/A	50.0
Hrank [39]	93.96 → 93.43	-0.53	82.9	53.5
FalCon [223]	93.32 → 91.92	-1.40	N/A	67.3
Ours	**93.96 → 93.76**	**-0.20**	80.8	73.7
Hrank [39]	93.96 → 91.23	-2.73	92.0	76.5
Ours	**93.96 → 93.61**	**-0.35**	**84.8**	**76.7**

5.3.2.2 Results on ImageNet

In this section, we verify the performance of our method on a large-scale dataset (ImageNet). Table 5.4 shows that our method achieves competitive performance compared to state-of-the-art methods. For moderate compression, our approach outperforms HRank [39], LSTM [221], BNFI [226], and SFP [30] in parameter reduction, FLOPs reduction, and Top-1 accuracy. Besides, our method shares similar FLOPs reduction with Hinge [124] and RL-MCTS [225] but obtains less accuracy degradation (both in Top-1 and Top-5). Although the pruning rate of our method is slightly lower than CFP [210], DSA [121], Autopruner [227], GDP [21], BNFI [226], FalCon [223], and FPGM [37], it can retain much more performance of the original model than them.

For deep compression, the Top-1 accuracy of our method exceeds Hrank [39] and FalCon [223] by 2.82% and 0.93% respectively under similar FLOPs reduction. Notably, compared to Hrank [39], which reduces the FLOPs up to 76%, our method achieves slightly more parameters and FLOPs drop but with much less accuracy degradation (-2.97% vs.-7.05%). From the quantitative results, our pruning method can not only outperform the state-of-the-art methods in moderate compression but also maintain higher accuracy even in deep compression.

Table 5.4: Comparison of pruned ResNet-50 on ImageNet.

Method	**Top1(%)**	**ΔTop1(%)**	**Top5(%)**	**ΔTop5(%)**	**Params(%)↓**	**FLOPs(%)↓**
SFP [30]	76.15 → 74.61	-1.54	92.87 → 92.06	-0.81	N/A	41.8
BNFI [226]	76.33 → 75.47	-0.86	N/A	N/A	N/A	42.8
LSTM [221]	76.12 → 75.00	-1.12	93.00 → 92.67	-0.33	N/A	43.0
HRank [39]	76.15 → 74.98	-1.17	92.87 → 92.33	-0.54	36.6	43.7
CPS [228]	76.15 → 75.59	-0.46	N/A	N/A	N/A	44.3
RL-MCTS [225]	77.34 → 76.80	-0.54	93.27 → 93.00	-0.27	N/A	46.1
Hinge [124]	76.10 → 74.70	-1.40	N/A	N/A	N/A	46.5
Ours	**76.15 → 75.83**	**-0.32**	**92.87 → 92.76**	**-0.11**	**44.2**	48.7
CFP [210]	75.30 → 73.40	-1.90	92.20 → 91.40	-0.80	N/A	49.6
DSA [121]	76.02 → 74.69	-1.33	N/A	-0.80	N/A	50.0
Autopruner [227]	76.15 → 74.76	-1.39	92.87 → 92.15	-0.72	N/A	51.2
GDP [21]	75.13 → 71.89	-3.24	92.30 → 90.71	-1.59	N/A	51.3
BNFI [226]	76.33 → 75.02	-1.29	N/A	N/A	N/A	52.6
FalCon [223]	75.83 → 74.59	-1.24	92.78 → 92.51	-0.27	N/A	53.5
FPGM [37]	76.15 → 74.83	-1.32	92.87 → 92.32	-0.55	N/A	53.5
ABCPruner [117]	76.01 → 73.86	-2.15	92.96 → 91.69	-1.27	N/A	54.3
RL-MCTS [225]	77.34 → 76.46	-0.88	93.27 → 92.83	-0.44	N/A	55.0
Hrank [39]	76.15 → 71.98	-2.73	92.87 → 92.68	-1.86	46.0	62.1
Ours	**76.15 → 74.80**	**-1.35**	**92.87 → 92.39**	**-0.48**	**56.7**	62.8
FalCon [223]	75.83 → 73.55	-2.28	92.78 → 91.99	-0.79	N/A	63.4
Hrank [39]	76.15 → 69.10	-7.05	92.87 → 89.58	-3.29	67.5	76.0
Ours	**76.15 → 73.18**	**-2.97**	**92.87 → 91.32**	**-1.55**	**68.6**	**76.7**

5.3.3 Ablation Study

5.3.3.1 The Comparison of High and Low Representations

Frequency information is able to measure the pixel value change in images. Specifically, low-frequency and high-frequency signals reflect the image regions with smooth and sharp differences, respectively. In general, low-frequency features are more informative than high-frequency ones in visual tasks [162]. To analyze the impact of different frequencies of information in pruning, we conduct experiments with various networks and compare the results on CIFAR-10. Fig. 5.3 shows the Top-1 accuracy achieved by different networks and different frequencies of features. Under the same pruning rate, it is observed that pruning with low-frequency features is better than with high-frequency ones, which means the low-frequency components can effectively measure the uniqueness of feature maps. In addition, we follow Eq. 5.3 to prune filters based on their spatial uniqueness scores, whose Top-1 accuracy is in-between the methods with low-frequency and high-frequency features. The comparison results show that the low-frequency components in our method indeed consist of more valuable information than others, validating the effectiveness of our frequency-based strategy.

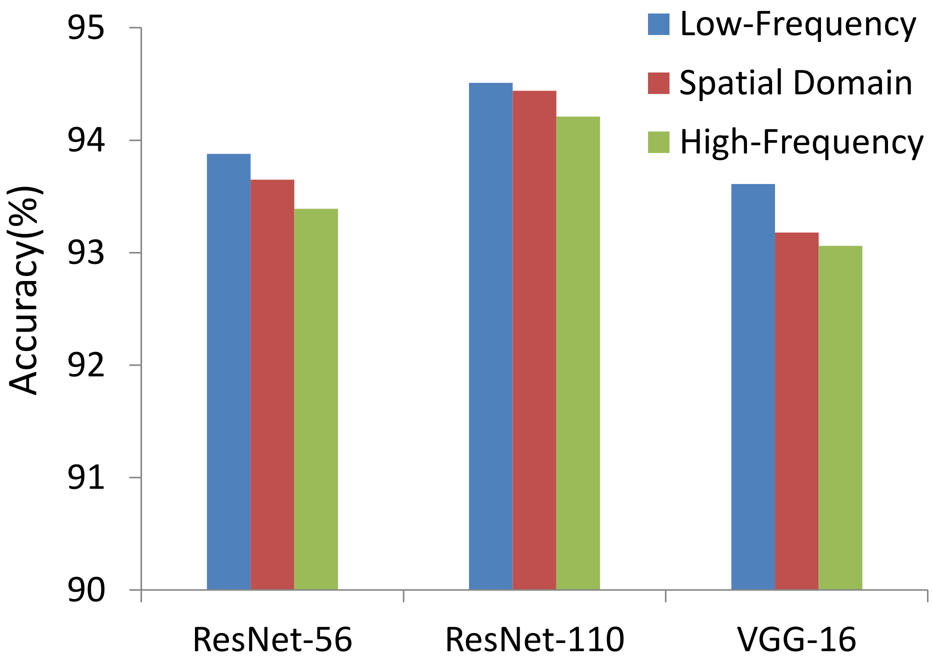


Figure 5.3: Top-1 accuracy of various models pruned with different frequencies/domains of features.

5.3.3.2 The Effectiveness between Various Pruning Methods

To further evaluate the effectiveness of our proposed method, we study the accuracy-pruning rate trade-off curve for the ResNet-50 model on ImageNet dataset. The results are shown in Figure 5.4:

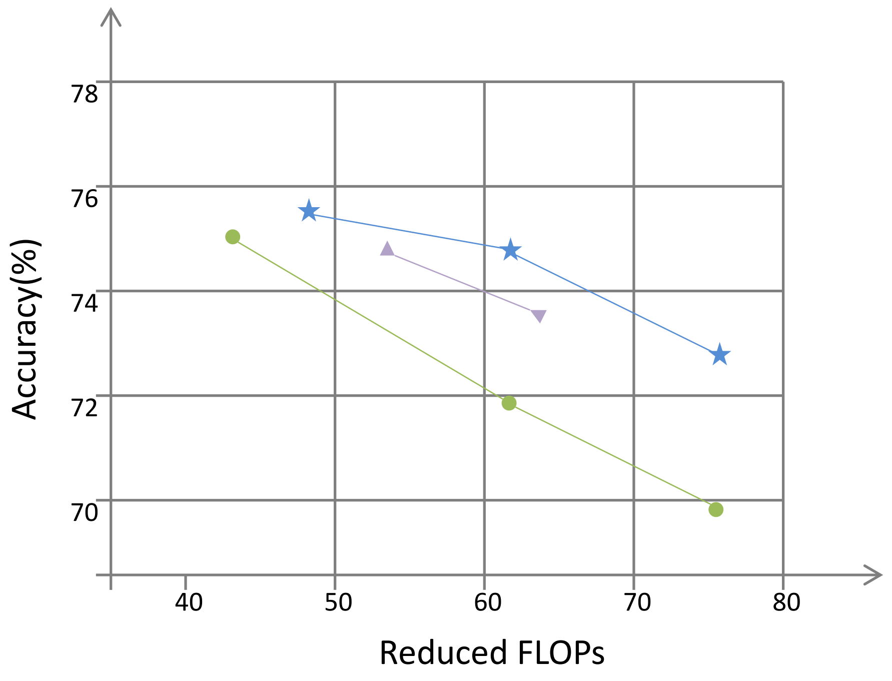


Figure 5.4: The comparison of the accuracy-pruning rate trade-off curve for various pruning methods using the ResNet-50 model on the ImageNet dataset. Note that the blue stars indicate the results of our method while the green dots and pruple curves show the results of Hrank [39] and FalCon [223],respectively.

In this figure, the blue star represents the accuracy of our method under the specific FLOPs reduction, whereas the green circle and purple triangle indicate the accuracy of HRank and FalCon with given FLOPs reduction, respectively. As we can see that our proposed method achieves the best performance throughout various pruning rates.

5.3.3.3 Feature Map Visualization

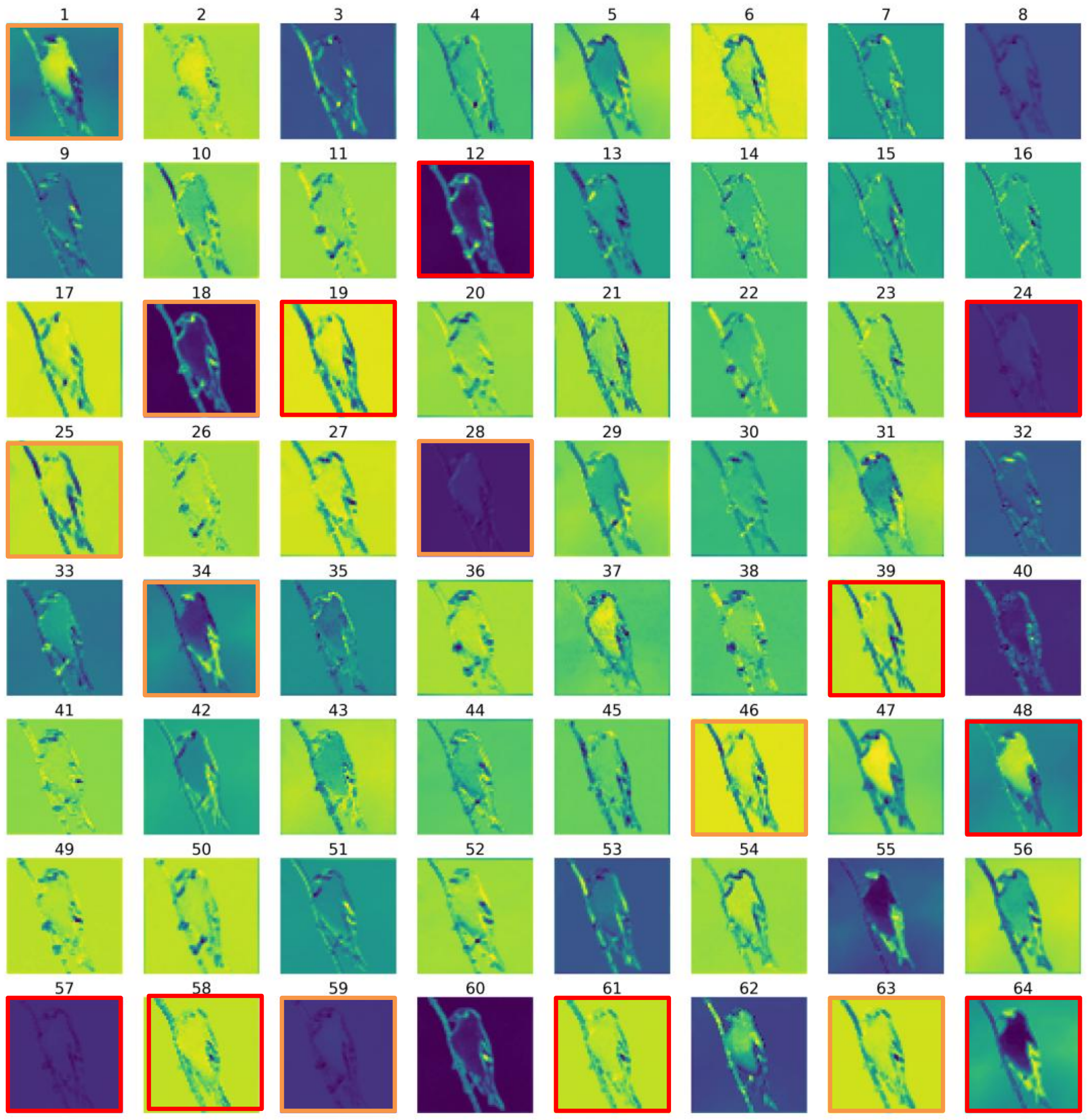


Figure 5.5: The visualization of feature maps, generated by ResNet-50 (block1, conv1), where red boxes highlight the pruned feature maps by our method, orange boxes indicate the feature maps (retained) which can replace the pruned ones.

To explore the effectiveness of our uniqueness computation, we visualize one layer of feature maps before and after pruning, as shown in Fig. 5.5. Specifically, we select the features generated by the first convolution layer in the first block of ResNet-50, trained on ImageNet. Under the pruning rate of 0.15, our method removes the feature maps with indices 12, 19, 24, 39, 48, 57, 58, 61, and 64 (marked by red boxes) due to their low uniqueness scores. Although the removed maps cannot contribute to the subsequent computation, it will not degrade the final results considerably since the remaining maps can easily replace them. From Fig. 5.5, we can find the substitute for each removed map and show them as a pair with the format (removed map index - substitute map index with **bold** text): (12-**18**), (19-**46**), (24-**28**), (39-**25**), (48-**1**), (57-**59**), (58-**63**), (61-**63**), and (64-**34**). These substitutes are marked by orange boxes in Fig. 5.5 to highlight their relationships to the removed ones. To sum up, the visualized results show that the feature maps pruned by our method are indeed replaceable and unimportant, further validating our uniqueness-based strategy.

5.4 Chapter Summary

This work presented a novel pruning method, which operates mainly in the frequency domain and computes uniqueness as the critical criteria for removing filters. Unlike the previous spatial methods, we further transform the encoded features into the frequency domain by DCT to mine more valuable and concentrated information from the input data. After this, we compute uniqueness scores from each feature map, considering both the properties within and across maps. The network pruning is achieved by removing the filters corresponding to the low-uniqueness maps, which can be easily replaced by others. This way, the proposed method can effectively reduce the network complexity while maintaining its performance to the largest extent. We evaluated our method with various network architectures on two different scales of datasets. The experimental results showed that our method achieves superior performance compared to the state-of-the-art approaches.

The proposed method selects redundant filters based on the correlations of feature maps rather than that of filters. In the next chapter, we will summarize this thesis and discuss several possible research directions in the future.

Chapter 6

Conclusions and Future Work

6.1 Thesis Summary

This thesis mainly focuses on reducing the memory usage and computation cost while maintaining the network performance in network pruning. Particularly, three different challenging tasks are explored: filter pruning with an attention mechanism (Chapter 3), data-dependent filter pruning guided by LSTM (Chapter 4) and filter pruning with uniqueness mechanism in the frequency domain (Chapter 5). In each chapter, a novel learning framework is proposed to handle the target challenge in network pruning and the superior performance has been achieved against state-of-the-art baselines. Each chapter is briefly concluded and discussed in the following subsections.

6.1.1 Filter Pruning with an Attention Mechanism

To speed up the classical CNNs, a new filter pruning method named Pruning Filter with an Attention Mechanism (PFAM) is presented. Different from the previous pruning methods that evaluate the importance of filters by their intrinsic properties, a novel correlation-based filter pruning criterion that explores the long-range dependencies among filters via an attention module is employed to select and remove redundant filters. Our proposed method is evaluated on three general public datasets to demonstrate its superiority compared to state-of-the-art baselines. Moreover, it is also verified on three public remote sensing image datasets, and the experimental results show it can also be applied to particular tasks in computer vision.

We apply the attention mechanism to evaluate the redundancy of filters by calculating their correlation values among each other. The filters with the least correlation values can be deemed redundant and pruned with little degradation in performance. However, in our method, we select and prune the redundant filters in each layer yet treat all layers equally during the pruning stage, which may still have a non-negligible negative impact on overall performance if it happens to prune many filters from a critical layer. Hence, we are aware of this potential issue and plan to design a new pruning strategy in the following work that could determine the importance of layers before pruning redundant filters in each layer. Doing so can effectively differentiate the importance of different layers so as to preserve more filters in some important layers that have a positive impact on the final performance while removing more redundant filters in some less important layers. As a consequence, it can achieve the same model reduction while losing less performance than the previous method. The details of this subsequent work are presented in Chapter 4.

6.1.2 Data-Dependent Filter Pruning Guided by LSTM

In this chapter, we present a novel filter pruning framework to evaluate the importance of each network layer and thus generate the specific pruning decision for each one. Considering the hierarchical structure of CNN, we employ LSTM as an evaluation model to generate pruning decisions. Besides, based on the slimmer architecture generated from LSTM, we further propose the SEP attention mechanism to rebuild the baseline network, which realizes the data-dependent soft pruning. Experimental results show the superiority of our methods compared to both orderly and global pruning methods and reveal the ability to learn the sensitivity of each network layer.

Although LSTM can generate the pruning decision for a given network, training the LSTM to approximate the slimmer architectures usually takes a long time, which might be a bottleneck for our algorithm. Concretely, it requires several epochs to train each student model before the reward gets maximized. The computational cost of this step can be high, given a large-scale dataset such as ImageNet. In addition, both the previous pruning method in Chapter 3 and the proposed method in this Chapter still make the filter pruning decision under the filter view. In other words, we choose and prune redundant filters by calculating the intrinsic properties or correlation of filters. However, feature maps can be shown to have more merit than filters in the pruning stage since they embed more comprehensive information regarding filters and input data. Therefore, designing a novel pruning method that can extract information from feature maps rather than filters to make the filter pruning decision could select and prune redundant filters more accurately. Chapter 5 will illustrate the details of the follow-up improvement work.

6.1.3 Filter Pruning with Uniqueness Mechanism in the Frequency Domain

In this chapter, a novel filter pruning method is presented, which operates mainly in the frequency domain and computes uniqueness as the critical criterion for removing filters. We further transform the encoded features into the frequency domain by DCT to mine more valuable and concentrated information from the input data. After this, we compute uniqueness scores from each feature map, considering the properties within and across maps. The network pruning is achieved by removing the filters corresponding to the low-uniqueness maps, which can be easily replaced by others. This way, the proposed method can effectively reduce network complexity while maintaining its performance to the largest extent. We evaluated our method with various network architectures on two different scales of datasets. The experimental results showed that our method achieves superior performance compared to the state-of-the-art approaches.

Nevertheless, filter pruning only maintains accuracy under moderate sparsity rates because it reduces network complexity at a coarse-grained level by removing all weights in a filter. Moreover, since the granularity of filter pruning is much larger than that of weight pruning, with the degree of compression increasing, filter pruning suffers more significant performance degradation than weight pruning. Therefore, in our future work, we will propose an intermediate granular level for network pruning, which is coarser as compared to the fine-grained weight but finer as compared to the coarse-grained filter. By doing this, the issue of keeping the model accurate while achieving general CPU speedups can be solved, which enables practical model deployment on CPU-based platforms.

6.2 Future Research Topics

In this section, we briefly discuss several possible research directions in the future, which are strictly related to the research topic in this thesis.

Currently, some novel branches of network pruning methods [229–231] have been proposed to achieve deep model compression while maintaining excellent performance. For example, [229, 230] explore the possibility of hybrid pruning granularity for the pruning method. Generally, weight pruning is named the fine-grained level network pruning since it removes a single weight, while filter pruning, which prunes all weights in a filter can be deemed the coarse-grained level network pruning. Unlike the previous network pruning methods that prune unimportant weights based on the fine-grained or coarse-grained levels, [229, 230] remove unimportant weights using an intermediate granular level. The experimental results show that they can obtain the deep compression ratio as weight pruning while achieving the same performance as filter pruning.

Another interesting and promising network pruning method [231] uses the structural re-parameterization technique [232, 233]. To be specific, a set of structural parameters with powerful performance is trained during the training stage. Then, they convert multiple well-trained parameter modules into the parameters of a single module for the inference stage by means of an equivalent transformation. Doing so can use the parameters of a single module to achieve the same performance as the parameters of multiple modules. These two types of network pruning methods are different from ours, and combining them with the following two new applications would be a good direction for the following research stage.

6.2.1 Model Compression for Efficient Object Detection of Autonomous Vehicles

Autonomous driving technology is gradually changing people's lives in fundamental ways. As one of the key technologies of autonomous navigation, object detection has received much attention and developed rapidly in computer vision. Specially, 2D object detection receives two-dimensional image information from cameras to detect multiple environmental objects, such as pedestrians and cars. In addition, 3D object detection uses the 3D LiDAR signal (point clouds) to identify objects.

Although some powerful models, such as YOLO [234] and PointPillars [235] are designed to improve the accuracy of objects significantly, deploying and running these models requires huge memory and computing resources. Therefore, only the expensive and powerful GPUs rather than the embedded chips can fully release their performance, which cannot be employed on cars due to their space and cost. Consequently, how to detect objects accurately and efficiently under the constraints of memory and computation resources has become a hot research topic.

6.2.2 Network Pruning for Transformers

Inspired by the attention mechanism [146], [236, 237] were proposed by applying it in the field of natural language processing (NLP), which achieved astounding performance for machine translation. Afterward, some attention-based works [137,238–241] were proposed to achieve excellent performance in various tasks in computer vision compared to the traditional convolutional neural networks (CNNs). Unlike the previous traditional CNNs, transformers [57,65,242–244] that adopt the attention-based structure also showed their powerful performance on various tasks in both natural speech processing and computer vision fields.

However, similarly to the limitations of traditional CNNs, such powerful models still require large memory spaces and high computing resources, which means they can only be employed on large servers with expensive and energy-consuming GPUs. As one of the main methods of model compression, network pruning can effectively reduce the parameters of models while maintaining their high performance. Therefore, how to apply the new pruning techniques for the transformers that can be employed in the embedded systems has attracted much attention for researchers.

Bibliography

[1] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with noisy student improves imagenet classification," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 687–10 698.

[2] J. Cao, H. Cholakkal, R. M. Anwer, F. S. Khan, Y. Pang, and L. Shao, "D2det: Towards high quality object detection and instance segmentation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 485–11 494.

[3] L. Porzi, M. Hofinger, I. Ruiz, J. Serrat, S. R. Bulo, and P. Kontschieder, "Learning multi-object tracking and segmentation from automatic annotations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6846–6855.

[4] M. Yousef and T. E. Bishop, "Origaminet: weakly-supervised, segmentation-free, one-step, full page text recognition by learning to unfold," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 710–14 719.

[5] W. Zhang, P. Tang, and L. Zhao, "Remote sensing image scene classification using cnn-capsnet," *Remote Sensing*, vol. 11, no. 5, p. 494, 2019.

[6] F. Zhu, L. Zhu, and Y. Yang, "Sim-real joint reinforcement transfer for 3d indoor navigation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 388–11 397.

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

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

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[10] S. Ge, S. Zhao, C. Li, and J. Li, "Low-resolution face recognition in the wild via selective knowledge distillation," *IEEE Transactions on Image Processing*, vol. 28, no. 4, pp. 2051–2062, 2018.

[11] Z. Xu, Y.-C. Hsu, and J. Huang, "Training shallow and thin networks for acceleration via knowledge distillation with conditional adversarial networks," *arXiv preprint arXiv:1709.00513*, 2017.

[12] F. Ruffy and K. Chahal, "The state of knowledge distillation for classification," *arXiv preprint arXiv:1912.10850*, 2019.

[13] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," *Advances in neural information processing systems*, vol. 30, 2017.

[14] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[15] Q. Hu, P. Wang, and J. Cheng, "From hashing to cnns: Training binary weight networks via hashing," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[16] Q. Zhao, L. Zhang, and A. Cichocki, "Bayesian cp factorization of incomplete tensors with automatic rank determination," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1751–1763, 2015.

[17] A.-H. Phan, P. Tichavskỳ, and A. Cichocki, "Partitioned hierarchical alternating least squares algorithm for cp tensor decomposition," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 2542–2546.

[18] M. Ashraphijuo, V. Aggarwal, and X. Wang, "A characterization of sampling patterns for low-tucker-rank tensor completion problem," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 531–535.

[19] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the regularity of sparse structure in convolutional neural networks," *arXiv preprint arXiv:1705.08922*, 2017.

[20] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[21] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang, "Accelerating convolutional networks via global & dynamic filter pruning." in *IJCAI*, vol. 2, no. 7. Stockholm, 2018, p. 8.

[22] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[23] S. Hanson and L. Pratt, "Comparing biases for minimal network construction with back-propagation," *Advances in neural information processing systems*, vol. 1, 1988.

[24] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989.

[25] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 1993, pp. 164–171.

[26] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, 1990, pp. 598–605.

[27] M. A. Carreira-Perpinán and Y. Idelbayev, ""learning-compression" algorithms for neural net pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8532–8541.

[28] F. Tung and G. Mori, "Clip-q: Deep network compression learning by in-parallel pruning-quantization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7873–7882.

[29] 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," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 184–199.

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

[31] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "Nvidia a100 tensor core gpu: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[32] L. Berrada, A. Zisserman, and M. P. Kumar, "Smooth loss functions for deep top-k classification," *arXiv preprint arXiv:1802.07595*, 2018.

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

[34] T.-W. Chin, R. Ding, C. Zhang, and D. Marculescu, "Towards efficient model compression via learned global ranking," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1518–1528.

[35] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

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

[37] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4340–4349.

[38] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[39] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1529–1538.

[40] S. Gao, F. Huang, J. Pei, and H. Huang, "Discrete model compression with resource constraint for deep neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1899–1908.

[41] S. Zhang, G. Wu, J. Gu, and J. Han, "Pruning convolutional neural networks with an attention mechanism for remote sensing image classification," *Electronics*, vol. 9, no. 8, p. 1209, 2020.

[42] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," *Advances in neural information processing systems*, vol. 27, 2014.

[43] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[44] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[45] C. Tai, T. Xiao, Y. Zhang, X. Wang *et al.*, "Convolutional neural networks with low-rank regularization," *arXiv preprint arXiv:1511.06067*, 2015.

[46] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and pattern Recognition*, 2015, pp. 1984–1992.

[47] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 1943–1955, 2015.

[48] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.

[49] M. Astrid and S.-I. Lee, "Cp-decomposition with tensor power method for convolutional neural networks compression," in *2017 IEEE International Conference on Big Data and Smart Computing (BigComp).* IEEE, 2017, pp. 115–118.

[50] A.-H. Phan, K. Sobolev, K. Sozykin, D. Ermilov, J. Gusak, P. Tichavskỳ, V. Glukhov, I. Oseledets, and A. Cichocki, “Stable low-rank tensor decomposition for compression of convolutional neural network,” in *European Conference on Computer Vision.* Springer, 2020, pp. 522–539.

[51] M. Yin, Y. Sui, S. Liao, and B. Yuan, “Towards efficient tensor decomposition-based dnn model compression with optimization framework,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 674–10 683.

[52] G. Hinton, O. Vinyals, J. Dean *et al.*, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[53] T. Fukuda, M. Suzuki, G. Kurata, S. Thomas, J. Cui, and B. Ramabhadran, “Efficient knowledge distillation from an ensemble of teachers.” in *Interspeech*, 2017, pp. 3697–3701.

[54] A. Polino, R. Pascanu, and D. Alistarh, “Model compression via distillation and quantization,” *arXiv preprint arXiv:1802.05668*, 2018.

[55] G. K. Nayak, K. R. Mopuri, V. Shaj, V. B. Radhakrishnan, and A. Chakraborty, “Zero-shot knowledge distillation in deep networks,” in *International Conference on Machine Learning.* PMLR, 2019, pp. 4743–4751.

[56] R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova, and J. Lin, “Distilling task-specific knowledge from bert into simple neural networks,” *arXiv preprint arXiv:1903.12136*, 2019.

[57] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter,” *arXiv preprint arXiv:1910.01108*, 2019.

[58] X. Jin, B. Peng, Y. Wu, Y. Liu, J. Liu, D. Liang, J. Yan, and X. Hu, “Knowledge distillation via route constrained optimization,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1345–1354.

[59] Q. Guo, X. Wang, Y. Wu, Z. Yu, D. Liang, X. Hu, and P. Luo, “Online knowledge distillation via collaborative learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 020–11 029.

[60] D. Walawalkar, Z. Shen, and M. Savvides, “Online ensemble model compression using knowledge distillation,” in *European Conference on Computer Vision.* Springer, 2020, pp. 18–35.

[61] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, “Deep mutual learning,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4320–4328.

[62] J. H. Cho and B. Hariharan, “On the efficacy of knowledge distillation,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 4794–4802.

[63] C. Yang, L. Xie, S. Qiao, and A. L. Yuille, “Training deep neural networks in generations: A more tolerant teacher educates better students,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 5628–5635.

[64] M. Phuong and C. H. Lampert, “Distillation-based training for multi-exit architectures,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1355–1364.

[65] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, “Fitnets: Hints for thin deep nets,” *arXiv preprint arXiv:1412.6550*, 2014.

[66] S. Zagoruyko and N. Komodakis, “Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer,” *arXiv preprint arXiv:1612.03928*, 2016.

[67] M. Courbariaux, Y. Bengio, and J.-P. David, “Binaryconnect: Training deep neural networks with binary weights during propagations,” in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.

[68] M. Kim and P. Smaragdis, “Bitwise neural networks,” *arXiv preprint arXiv:1601.06071*, 2016.

[69] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *European Conference on Computer Vision.* Springer, 2016, pp. 525–542.

[70] Z. Li, B. Ni, W. Zhang, X. Yang, and W. Gao, "Performance guaranteed network acceleration via high-order residual quantization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2584–2592.

[71] X. Wang, B. Zhang, C. Li, R. Ji, J. Han, X. Cao, and J. Liu, "Modulated convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 840–848.

[72] P. Wang, Q. Hu, Y. Zhang, C. Zhang, Y. Liu, and J. Cheng, "Two-step quantization for low-bit neural networks," in *Proceedings of the IEEE Conference on computer vision and pattern recognition*, 2018, pp. 4376–4384.

[73] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "Wrpn: Wide reduced-precision networks," *arXiv preprint arXiv:1709.01134*, 2017.

[74] A. Bulat and G. Tzimiropoulos, "Xnor-net++: Improved binary neural networks," *arXiv preprint arXiv:1909.13863*, 2019.

[75] J. Faraone, N. Fraser, M. Blott, and P. H. Leong, "Syq: Learning symmetric quantization for efficient deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4300–4309.

[76] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[77] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," *arXiv preprint arXiv:1711.05852*, 2017.

[78] Y. Xu, X. Dong, Y. Li, and H. Su, "A main/subsidiary network framework for simplifying binary neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7154–7162.

[79] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," 2011.

[80] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[81] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4820–4828.

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

[83] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back-propagation," in *Advances in Neural Information Processing Systems*, 1989, pp. 177–185.

[84] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[85] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems*, vol. 2, 1989.

[86] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[87] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *arXiv preprint arXiv:1507.06149*, 2015.

[88] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[89] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, 2016, pp. 1379–1387.

[90] 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," *arXiv preprint arXiv:1608.01409*, 2016.

[91] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5687–5695.

[92] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[93] X. Xiao, Z. Wang, and S. Rajasekaran, "Autoprune: Automatic network pruning by regularizing auxiliary parameters," *Advances in neural information processing systems*, vol. 32, 2019.

[94] X. Ding, X. Zhou, Y. Guo, J. Han, J. Liu *et al.*, "Global sparse momentum sgd for pruning very deep neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[95] V. Sanh, T. Wolf, and A. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 20 378–20 389, 2020.

[96] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin, "Layer-adaptive sparsity for the magnitude-based pruning," *arXiv preprint arXiv:2010.07611*, 2020.

[97] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, "Net-trim: Convex pruning of deep neural networks with performance guarantee," *Advances in neural information processing systems*, vol. 30, 2017.

[98] C. Chen, F. Tung, N. Vedula, and G. Mori, "Constraint-aware deep neural network compression," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 400–415.

[99] Z. Liu, J. Xu, X. Peng, and R. Xiong, "Frequency-domain dynamic pruning for convolutional neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[100] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International conference on machine learning*. PMLR, 2015, pp. 2285–2294.

[101] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," *arXiv preprint arXiv:1702.04008*, 2017.

[102] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[103] V. Roth and B. Fischer, “The group-lasso for generalized linear models: uniqueness of solutions and efficient algorithms,” in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 848–855.

[104] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani, “A sparse-group lasso,” *Journal of computational and graphical statistics*, vol. 22, no. 2, pp. 231–245, 2013.

[105] Z. Huang and N. Wang, “Data-driven sparse structure selection for deep neural networks,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 304–320.

[106] E. Tartaglione, S. Lepsøy, A. Fiandrotti, and G. Francini, “Learning sparse neural networks via sensitivity-driven regularization,” *Advances in neural information processing systems*, vol. 31, 2018.

[107] X. Xu, M. S. Park, and C. Brick, “Hybrid pruning: Thinner sparse networks for fast inference on edge devices,” *arXiv preprint arXiv:1811.00482*, 2018.

[108] X. Ding, G. Ding, J. Han, and S. Tang, “Auto-balanced filter pruning for efficient convolutional neural networks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[109] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1389–1397.

[110] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7046–7055.

[111] T. Zhuang, Z. Zhang, Y. Huang, X. Zeng, K. Shuang, and X. Li, “Neuron-level structured pruning using polarization regularizer,” *Advances in neural information processing systems*, vol. 33, pp. 9865–9877, 2020.

[112] H. Wang, C. Qin, Y. Zhang, and Y. Fu, “Neural pruning via growing regularization,” *arXiv preprint arXiv:2012.09243*, 2020.

[113] J. Lin, Y. Rao, J. Lu, and J. Zhou, “Runtime neural pruning,” *Advances in neural information processing systems*, vol. 30, 2017.

[114] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal sgd for pruning very deep convolutional networks with complicated structure," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4943–4953.

[115] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun, "Metapruning: Meta learning for automatic neural network channel pruning," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 3296–3305.

[116] X. Dong and Y. Yang, "Network pruning via transformable architecture search," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[117] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," *arXiv preprint arXiv:2001.08565*, 2020.

[118] S. Guo, Y. Wang, Q. Li, and J. Yan, "Dmcp: Differentiable markov channel pruning for neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1539–1547.

[119] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 4876–4883.

[120] Z. Zhan, Y. Gong, P. Zhao, G. Yuan, W. Niu, Y. Wu, T. Zhang, M. Jayaweera, D. Kaeli, B. Ren *et al.*, "Achieving on-mobile real-time super-resolution with neural architecture and pruning search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 4821–4831.

[121] X. Ning, T. Zhao, W. Li, P. Lei, Y. Wang, and H. Yang, "Dsa: More efficient budgeted pruning via differentiable sparsity allocation," in *European Conference on Computer Vision.* Springer, 2020, pp. 592–607.

[122] Y. Li, S. Gu, K. Zhang, L. V. Gool, and R. Timofte, "Dhp: Differentiable meta pruning via hypernetworks," in *European Conference on Computer Vision.* Springer, 2020, pp. 608–624.

[123] T. Li, J. Li, Z. Liu, and C. Zhang, "Few sample knowledge distillation for efficient network compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 639–14 647.

[124] 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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 8018–8027.

[125] Y. Li, S. Lin, J. Liu, Q. Ye, M. Wang, F. Chao, F. Yang, J. Ma, Q. Tian, and R. Ji, “Towards compact cnns via collaborative compression,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 6438–6447.

[126] A. Dubey, M. Chatterjee, and N. Ahuja, “Coreset-based neural network compression,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 454–470.

[127] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani, “N2n learning: Network to network compression via policy gradient reinforcement learning,” *arXiv preprint arXiv:1709.06030*, 2017.

[128] K. Yamamoto and K. Maeno, “Pcas: Pruning channels with attention statistics for deep network compression,” *arXiv preprint arXiv:1806.05382*, 2018.

[129] Y. Tang, Y. Wang, Y. Xu, D. Tao, C. Xu, C. Xu, and C. Xu, “Scop: Scientific control for reliable neural network pruning,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 936–10 947, 2020.

[130] T. Chen, B. Ji, T. Ding, B. Fang, G. Wang, Z. Zhu, L. Liang, Y. Shi, S. Yi, and X. Tu, “Only train once: A one-shot neural network training and pruning framework,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 19 637–19 651, 2021.

[131] B. Li, B. Wu, J. Su, and G. Wang, “Eagleeye: Fast sub-net evaluation for efficient neural network pruning,” in *European conference on computer vision*. Springer, 2020, pp. 639–654.

[132] R. Pahwa, M. G. Arivazhagan, A. Garg, S. Krishnamoorthy, R. Saxena, and S. Choudhary, “Data-driven compression of convolutional neural networks,” *arXiv preprint arXiv:1911.12740*, 2019.

[133] J. Chen, S. Chen, and S. J. Pan, “Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 14 747–14 758, 2020.

[134] 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 (ECCV)*, 2018, pp. 784–800.

[135] 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," *Advances in neural information processing systems*, vol. 31, 2018.

[136] X. Ding, G. Ding, Y. Guo, J. Han, and C. Yan, "Approximated oracle filter pruning for destructive cnn width optimization," in *International Conference on Machine Learning*. PMLR, 2019, pp. 1607–1616.

[137] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[138] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-z. Xu, "Dynamic channel pruning: Feature boosting and suppression," *arXiv preprint arXiv:1810.05331*, 2018.

[139] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 264–11 272.

[140] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[141] L. Liebenwein, C. Baykal, H. Lang, D. Feldman, and D. Rus, "Provable filter pruning for efficient neural networks," *arXiv preprint arXiv:1911.07412*, 2019.

[142] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang, "Learning filter pruning criteria for deep convolutional neural networks acceleration," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2009–2018.

[143] C. M. J. Tan and M. Motani, "Dropnet: Reducing neural network complexity via iterative pruning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9356–9366.

[144] Y. Tang, Y. Wang, Y. Xu, Y. Deng, C. Xu, D. Tao, and C. Xu, "Manifold regularized dynamic network pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5018–5028.

[145] Z. Wang, C. Li, and X. Wang, "Convolutional neural network pruning with structural redundancy reduction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 913–14 922.

[146] V. Mnih, N. Heess, A. Graves, and k. kavukcuoglu, "Recurrent models of visual attention," in *Advances in Neural Information Processing Systems*, 2014, pp. 2204–2212.

[147] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," *arXiv preprint arXiv:1412.7755*, 2014.

[148] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2017–2025.

[149] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3156–3164.

[150] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model for natural language inference," *arXiv preprint arXiv:1606.01933*, 2016.

[151] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[152] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7794–7803.

[153] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," *arXiv preprint arXiv:1805.08318*, 2018.

[154] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[155] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[156] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the genetic and evolutionary computation conference*, 2017, pp. 497–504.

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

[158] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[159] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[160] L. Gueguen, A. Sergeev, B. Kadlec, R. Liu, and J. Yosinski, "Faster neural networks straight from jpeg," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[161] M. Ehrlich and L. S. Davis, "Deep residual learning in the jpeg transform domain," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3484–3493.

[162] K. Xu, M. Qin, F. Sun, Y. Wang, Y.-K. Chen, and F. Ren, "Learning in the frequency domain," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1740–1749.

[163] Z. Qin, P. Zhang, F. Wu, and X. Li, "Fcanet: Frequency channel attention networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 783–792.

[164] L. Jiang, B. Dai, W. Wu, and C. C. Loy, "Focal frequency loss for image reconstruction and synthesis," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 13 919–13 929.

[165] M. Cai, H. Zhang, H. Huang, Q. Geng, Y. Li, and G. Huang, "Frequency domain image translation: More photo-realistic, better identity-preserving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 13 930–13 940.

[166] W. Chen, J. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing convolutional neural networks in the frequency domain," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1475–1484.

[167] C. Liang-Chieh, G. Papandreou, I. Kokkinos, K. Murphy *et al.*, "Semantic image segmentation with deep convolutional nets and fully connected crfs," in *International conference on learning representations*, 2015.

[168] K. Yanai, R. Tanno, and K. Okamoto, "Efficient mobile implementation of a cnn-based object recognition system," in *Proceedings of the 24th ACM international conference on Multimedia*, 2016, pp. 362–366.

[169] R. J. Wang, X. Li, and C. X. Ling, "Pelee: A real-time object detection system on mobile devices," *Advances in neural information processing systems*, vol. 31, 2018.

[170] K. Yang, T. Xing, Y. Liu, Z. Li, X. Gong, X. Chen, and D. Fang, "cdeeparch: A compact deep neural network architecture for mobile sensing," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 2043–2055, 2019.

[171] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Convolutional neural networks for large-scale remote-sensing image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 2, pp. 645–657, 2016.

[172] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," in *Advances in Neural Information Processing Systems*, 2014, pp. 963–971.

[173] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, 2015, pp. 1117–1125.

[174] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[175] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[176] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[177] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[178] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks.* IEEE, 1993, pp. 293–299.

[179] H. Van Nguyen, K. Zhou, and R. Vemulapalli, "Cross-domain synthesis of medical images using efficient location-sensitive deep network," in *International Conference on Medical Image Computing and Computer-Assisted Intervention.* Springer, 2015, pp. 677–684.

[180] S. Tang and J. Han, "A pruning based method to learn both weights and connections for lstm," in *Advances In Neural Information Processing Systems.* NIPS, 2015.

[181] Y. Hu, S. Sun, J. Li, X. Wang, and Q. Gu, "A novel channel pruning method for deep neural network compression," *arXiv preprint arXiv:1805.11394*, 2018.

[182] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," *arXiv preprint arXiv:1802.00124*, 2018.

[183] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," *arXiv preprint arXiv:1601.06733*, 2016.

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

[185] 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," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[186] G. Cheng, J. Han, and X. Lu, “Remote sensing image scene classification: Benchmark and state of the art,” *Proceedings of the IEEE*, vol. 105, no. 10, pp. 1865–1883, 2017.

[187] G.-S. Xia, J. Hu, F. Hu, B. Shi, X. Bai, Y. Zhong, L. Zhang, and X. Lu, “Aid: A benchmark data set for performance evaluation of aerial scene classification,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 7, pp. 3965–3981, 2017.

[188] Q. Zou, L. Ni, T. Zhang, and Q. Wang, “Deep learning based feature selection for remote sensing scene classification,” *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 11, pp. 2321–2325, 2015.

[189] 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,” 2017.

[190] X. Dong, J. Huang, Y. Yang, and S. Yan, “More is less: A more complicated network with less inference complexity,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5840–5848.

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

[192] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2790–2799.

[193] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[194] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, “Show and tell: A neural image caption generator,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3156–3164.

[195] S. Ye, J. Han, and N. Liu, “Attentive linear transformation for image captioning,” *IEEE Transactions on Image Processing*, vol. 27, no. 11, pp. 5514–5524, 2018.

[196] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning.* PMLR, 2015, pp. 2048–2057.

[197] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[198] Y. Zhu, C. Zhao, H. Guo, J. Wang, X. Zhao, and H. Lu, "Attention couplenet: Fully convolutional attention coupling network for object detection," *IEEE Transactions on Image Processing*, vol. 28, no. 1, pp. 113–126, 2018.

[199] G. Cheng, J. Han, P. Zhou, and D. Xu, "Learning rotation-invariant and fisher discriminative convolutional neural networks for object detection," *IEEE Transactions on Image Processing*, vol. 28, no. 1, pp. 265–278, 2018.

[200] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[201] X. Zhang, H. Luo, X. Fan, W. Xiang, Y. Sun, Q. Xiao, W. Jiang, C. Zhang, and J. Sun, "Alignedreid: Surpassing human-level performance in person reidentification," *arXiv preprint arXiv:1711.08184*, 2017.

[202] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[203] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[204] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[205] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[206] J. Ba and R. Caruana, "Do deep nets really need to be deep?" *Advances in neural information processing systems*, vol. 27, 2014.

[207] Y. LeCun, “The mnist database of handwritten digits,” *http://yann. lecun. com/exdb/mnist/*, 1998.

[208] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9194–9203.

[209] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[210] P. Singh, V. K. Verma, P. Rai, and V. Namboodiri, “Leveraging filter correlations for deep model compression,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 835–844.

[211] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[212] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[213] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *Advances in neural information processing systems*, vol. 28, 2015.

[214] B. Singh, M. Najibi, and L. S. Davis, “Sniper: Efficient multi-scale training,” *Advances in neural information processing systems*, vol. 31, 2018.

[215] Z. Zhu, M. Xu, S. Bai, T. Huang, and X. Bai, “Asymmetric non-local neural networks for semantic segmentation,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 593–602.

[216] M. Gao, F. Zheng, J. J. Yu, C. Shan, G. Ding, and J. Han, “Deep learning for video object segmentation: a review,” *Artificial Intelligence Review*, pp. 1–75, 2022.

[217] N. Aghli and E. Ribeiro, "Combining weight pruning and knowledge distillation for cnn compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3191–3198.

[218] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[219] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[220] Q. Guo, X.-J. Wu, J. Kittler, and Z. Feng, "Differentiable neural architecture learning for efficient neural networks," *Pattern Recognition*, p. 108448, 2022.

[221] G. Ding, S. Zhang, Z. Jia, J. Zhong, and J. Han, "Where to prune: Using lstm to guide data-dependent soft pruning," *IEEE Transactions on Image Processing*, vol. 30, pp. 293–304, 2020.

[222] Y. Lian, P. Peng, and W. Xu, "Filter pruning via separation of sparsity search and model training," *Neurocomputing*, vol. 462, pp. 185–194, 2021.

[223] Z. Xu, F. Yu, C. Liu, Z. Wu, H. Wang, and X. Chen, "Falcon: Fine-grained feature map sparsity computing with decomposed convolutions for inference optimization," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 350–360.

[224] Y. Chen, X. Wen, Y. Zhang, and W. Shi, "Ccprune: Collaborative channel pruning for learning compact convolutional networks," *Neurocomputing*, vol. 451, pp. 35–45, 2021.

[225] Z. Wang and C. Li, "Channel pruning via lookahead search guided reinforcement learning," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 2029–2040.

[226] J. Oh, H. Kim, S. Baik, C. Hong, and K. M. Lee, "Batch normalization tells you which filter is important," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 2645–2654.

[227] J.-H. Luo and J. Wu, "Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference," *Pattern Recognition*, vol. 107, p. 107461, 2020.

[228] C. Yang and H. Liu, "Channel pruning based on convolutional neural network sensitivity," *Neurocomputing*, vol. 507, pp. 97–106, 2022.

[229] M. Lin, Y. Zhang, Y. Li, B. Chen, F. Chao, M. Wang, S. Li, Y. Tian, and R. Ji, "1xn pattern for pruning convolutional neural networks," *arXiv preprint arXiv:2105.14713*, 2021.

[230] F. Meng, H. Cheng, K. Li, H. Luo, X. Guo, G. Lu, and X. Sun, "Pruning filter in filter," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17 629–17 640, 2020.

[231] X. Ding, T. Hao, J. Tan, J. Liu, J. Han, Y. Guo, and G. Ding, "Resrep: Lossless cnn pruning via decoupling remembering and forgetting," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 4510–4520.

[232] X. Ding, Y. Guo, G. Ding, and J. Han, "Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1911–1920.

[233] X. Ding, X. Zhang, J. Han, and G. Ding, "Diverse branch block: Building a convolution as an inception-like unit," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 886–10 895.

[234] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[235] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 12 697–12 705.

[236] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[237] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[238] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 764–773.

[239] J. Fu, J. Liu, H. Tian, Y. Li, Y. Bao, Z. Fang, and H. Lu, "Dual attention network for scene segmentation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 3146–3154.

[240] B. Chen, W. Deng, and J. Hu, "Mixed high-order attention network for person re-identification," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 371–381.

[241] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7794–7803.

[242] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.

[243] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[244] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.