HIROSHIMA UNIVERSITY

MASTER THESIS

BonsaiNet: Searching Neural Networks Structure by Pruning and Planting

Author:
Kakeru MITSUNO (M201832)

Supervisor:
Professor Takio KURITA

A thesis submitted in fulfillment of the requirements
for the degree of Master of Engineering

in the

Department of Information Engineering

January 5, 2022

Declaration of Authorship

I, Kakeru MITSUNO (M201832), declare that this thesis titled, "BonsaiNet: Searching Neural Networks Structure by Pruning and Planting" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Kakeru Mitsuno

Date: 2022

HIROSHIMA UNIVERSITY

Abstract

Graduate School of Engineering
Department of Information Engineering

Master of Engineering

BonsaiNet: Searching Neural Networks Structure by Pruning and Planting

by Kakeru MITSUNO (M201832)

In recent years, deep neural networks have become deeper and wider, requiring excessive parameters to achieve superior performance, which increases the computational cost and causes over-fitting. However, these methods utilizing the larger predefined network with handcrafted or existing architecture has an upper bound of the network performance. Searching the optimal network architecture while keeping network performance has been proposed by many researchers. In this paper, we present novel 2 type of neural network architecture search methods. One is decremental training algorithm called pruning and the other is an incremental training algorithm called planting.

The pruning method can remove unnecessary channels using the hierarchical group sparse regularization. It is shown in our previous work that the regularization is effective in obtaining sparse networks in which filters connected to unnecessary channels are automatically close to zero. After training the convolutional neural network with the regularization, the unnecessary filters are selected based on the increase of the classification loss of the randomly selected training samples to obtain a compact network. It is shown that the proposed method can reduce more than 50% parameters of ResNet for CIFAR-10 with only 0.3% decrease in the accuracy of test samples. Also, 34% parameters of ResNet are reduced for TinyImageNet-200 with higher accuracy than the baseline network.

The planting method can search the optimal network architecture for training tasks with smaller parameters by planting channels incrementally to layers of the initial networks while keeping the earlier trained channels fixed for improving the network performances. Also, we propose to use the knowledge distillation method for training the channels planted. By transferring the knowledge of deeper and wider networks, we can grow the networks effectively and efficiently. We evaluate the effectiveness of the proposed method on different datasets such as CIFAR10/100 and STL-10. For the STL-10 dataset, we show that we are able to achieve comparable performance with only 7% parameters compare to the larger network and reduce the overfitting caused by a small amount of the data.

We can find optimal neural networks structure by using the pruning method and the planting method.

Acknowledgements

I would like to express my gratitude to Associate Professor Takio Kuirta, Junichi Miyao, and Hiroaki Aizawa. They provided the good environment for my research, supported my student life and helped my research with many ideas. My research was completed thanks to their knowledge and advice.

I am also grateful to my lab members and my family.

Contents

List of Figures

List of Tables

List of Abbreviations

DNNs	**D**eep **N**eural **N**etworks
CNNs	**C**onvolutional **N**eural **N**etworks
FCNs	**F**ully **C**onvolutional **N**etworks
NAS	**N**eural **A**rchitecture **S**earch

Chapter 1

Introduction

Deep neural networks (DNNs) have been successful with superior performance in computer vision tasks such as image classification [27, 57]. Meanwhile, the network becomes deeper and wider, requiring excessive amount of parameters to achieve excellent performance [15, 21], which increases the computational cost and cause overfitting.

However, these methods utilizing the larger predefined network with handcrafted or existing architecture has an upper bound of the network performance. To search automatically optimal network architecture, Neural Architecture Search (NAS) is introduced. NAS explore the width and depth of networks efficiently for the training task, by using a recurrent neural network as the controller [79], using graph-based algorithm [49, 8], or optimizing search space [62].

To improve performance while reducing the computational cost, various network pruning approaches for compressing the size of the network have been proposed. Network pruning can reduce unnecessary parameters while keeping network performance, by using the Taylor expansion of the loss function [29, 14], enforcing unnecessary parameter to be 0 with sparse regularization [13, 61, 43], evaluating the importance of the parameter based on the norm [31, 17] or using the scaling parameter of batch normalization layers [37].

An approach similar to NAS is incremental training[58]. Incremental training is a dynamic configuration technique for DNNs, that initially train a subset of channels in each layer and gradually add in more channels while keeping the earlier trained channels fixed. By training with this method, we can obtain a flexibility in the network training, which can dynamically adjust the DNNs to reduce the computational cost as long as the accuracy of the classification results is not compromised.

There is another way to compress the size of networks, called knowledge distillation. Knowledge distillation is a technique for transferring the knowledge of a deeper and wider network or ensemble network (teacher networks) to smaller and shallower networks (student networks) by getting close the output of student networks to teacher networks. To transfer the knowledge, the L2 loss or the KL-divergence is used as the loss function of the method [3, 19].

In this paper, we propose a two types of NAS method, one is a filter pruning method, the other is a incremental training method.

First, We propose a filter pruning method with the hierarchical group sparse regularization based on the feature-wise grouping , whose grouping consider as a group the kernels connected to a input channel. By this grouping, we can prune unnecessary output channels in layer $l-1$ (the input channels in layer l). After training with the hierarchical group sparse regularization, we calculate the influence of each channel on the classification loss of the randomly selected training samples, and prune the channels based on the increase of the classification low. After we obtain a compact network by the filter pruning, the parameters of the compact network are retrained from scratch.

Finally, we introduce a novel incremental training method for DNNs called *"planting"*. The existing incremental training method uses the handcrafted network architecture as a base network and divides it into several sub-networks. There is an upper bound of the network performance since the architecture of the sub-networks is fixed. Our planting method can search the optimal network architecture for the training task with smaller parameters by planting channels incrementally to initial networks while keeping the earlier trained channels fixed for improving the network performances. Explore the architecture of the network by planting channels in a layer where the error is reduced by adding channels. For the training of the planted channels, the proposed method utilizes the knowledge transfer method. The parameters in the augmented channels are trained to complement the error of the earlier trained network by imitating the behavior of the teacher network.

Next section, we describe related works to introduce our proposal. First, we describe about Convolutional Neural Network. Second, we describe about sparse regularization and network pruning. Third, we describe about knowledge distillation. Finally, we describe about incremental training.

Third section, we describe our proposal filter pruning techniques with the hierarchical group sparse regularization based on the feature-wise grouping. To confirm the effectiveness of the proposed method, we have performed experiments with different network architectures (VGG and ResNet) on different data sets (CIFAR-10, CIFAR-100, and TinyImageNet-200). The results show that we can obtain the compact networks with about 50% less parameters without decrease of the classification accuracy.

The contributions of this section are summarized as follows:

- We propose a filter pruning method with the hierarchical group sparse regularization based on the feature-wise grouping, the regularization can prune filters more adequately depending on the structure of the network and the number of channels than non-hierarchical sparse regularization.

- The feature-wise grouping can prune the filters connected to unnecessary input channels by removing the channels with low influence on the classification loss.

- The effectiveness of the proposed pruning method is confirmed through experiments with different network architectures (VGG and ResNet) on different data sets (CIFAR-10, CIFAR-100, and TinyImageNet-200).

Forth section, we describe our proposal incremental training techniques for DNNs called *"planting"*. We evaluate the effectiveness of the proposed method on different datasets such as CIFAR-10/100 and STL-10. For the STL-10 dataset, we show that we are able to achieve comparable performance with only 7% parameters compared to the larger network and reduce the overfitting caused by a small amount of the data.

The contributions of this section are summarized as follows:

- We propose a novel incremental training method for DNNs called *planting*, that can train smaller network with excellent performance and find the optimal network architecture automatically.
- We introduce the knowledge transfer to train planted channels.
- We have performed experiments to evaluate the effectiveness of the proposed method on different datasets (CIFAR-10, CIFAR-100, and STL-10).

Chapter 2

Related Works

2.1 Convolutional Neural Network

Convolutional neural network (CNN) is inspired by a biological model [42] and is widely used for image and motion picture recognition. A general CNN has convolution layers, pooling layers, and fully connected layers as shown in Fig. 2.1.

In this part, the image convolution process is represented by a neural network. Calculation of convolution is given as

$$f_{p,q}^{(l)} = h\Big(\sum_{s=0}^{convy-1}\sum_{t=0}^{convx-1} w_{s,t}^{(l)} f_{p+s,q+t}^{(l-1)} + b^{(l)}\Big), \tag{2.1}$$

where $w_{s,t}^{(l)}$ and $b^{(l)}$ are the weight and bias of the convolution layer whose size is $convx \times convy$. Furthermore, h is an activation function and we assume here that it is ReLU and consider only when $f_{p,q}^{(l)} \geq 0$ as

$$f_{p,q}^{(l)} = \sum_{s=0}^{convy-1}\sum_{t=0}^{convx-1} w_{s,t}^{(l)} f_{p+s,q+t}^{(l-1)} + b^{(l)}. \tag{2.2}$$

Next, the calculation formula of the back propagation of the convolution layer is shown. We define an error $\delta_{p,q}^{(l)}$ as

$$\delta_{p,q}^{(l)} \equiv \frac{\partial E_n}{\partial f_{p,q}^{(l)}}. \tag{2.3}$$

Equation (2.3) is transformed as

$$\begin{aligned}\delta_{p,q}^{(l)} = \frac{\partial E_n}{\partial f_{p,q}^{(l)}} &= \sum_{s=0}^{convy-1}\sum_{t=0}^{convx-1} \frac{\partial E_n}{\partial f_{p-s,q-t}^{(l+1)}} \frac{\partial f_{p-s,q-t}^{(l+1)}}{\partial f_{p,q}^{(l)}} \\ &= \sum_{s=0}^{convy-1}\sum_{t=0}^{convx-1} \delta_{p-s,q-t}^{(l+1)} \frac{\partial f_{p-s,q-t}^{(l+1)}}{\partial f_{p,q}^{(l)}}.\end{aligned} \tag{2.4}$$

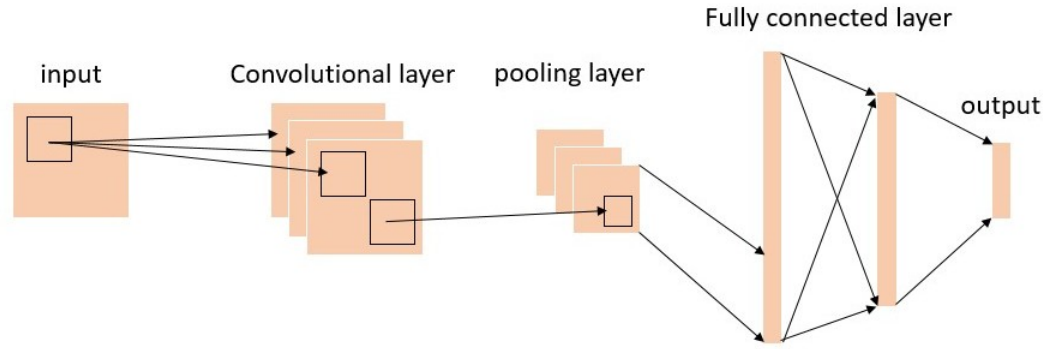


FIGURE 2.1: The structure of general CNN.

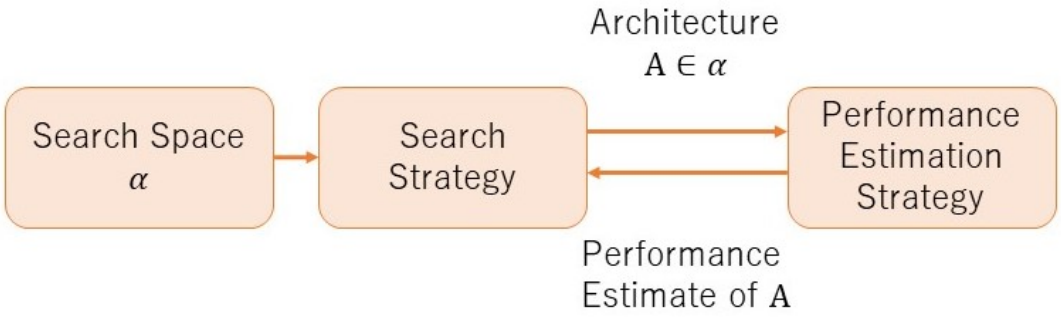


FIGURE 2.2: The illustration of Neural Architecture Search methods.

$\frac{\partial f^{(l+1)}_{p-s,q-t}}{\partial f^{(l)}_{p,q}}$ is calculated as

$$\begin{aligned}\frac{\partial f^{(l+1)}_{p-s,q-t}}{\partial f^{(l)}_{p,q}} &= \frac{\partial}{\partial f^{(l)}_{p,q}} \sum_{s'=0}^{convy-1} \sum_{t'=0}^{convx-1} w^{(l+1)}_{s',t'} f^{(l)}_{p-s+s',q-t+t'} + b^{(l+1)} \\ &= w^{(l+1)}_{s,t}. \end{aligned} \tag{2.5}$$

Substituting into Equation (2.4) results in

$$\delta^{(l)}_{p,q} = \sum_{s=0}^{convy-1} \sum_{t=0}^{convx-1} \delta^{(l+1)}_{p-s,q-t} w^{(l+1)}_{s,t}. \tag{2.6}$$

From the Equation (2.3), the following equation is obtained. Where m and n are the size of the input.

$$\begin{aligned}\frac{\partial E_n}{\partial w_{s,t}^{(l)}} &= \sum_{p=0}^{m-convy} \sum_{q=0}^{n-convx} \delta_{p,q}^{(l)} \frac{\partial f_{p,q}^{(l)}}{\partial w_{s,t}^{(l)}} \\ &= \sum_{p=0}^{m-convy} \sum_{q=0}^{n-convx} \delta_{p,q}^{(l)} f_{p+s,q+t}^{(l-1)} \end{aligned} \tag{2.7}$$

$$\begin{aligned}\frac{\partial E_n}{\partial b^{(l)}} &= \sum_{p=0}^{m-convy} \sum_{q=0}^{n-convx} \delta_{p,q}^{(l)} \frac{\partial f_{p,q}^{(l)}}{\partial b^{(l)}} \\ &= \sum_{p=0}^{m-convy} \sum_{q=0}^{n-convx} \delta_{p,q}^{(l)}. \end{aligned} \tag{2.8}$$

Therefore, error information can propagate to the previous layer and the network can learn a filter that reduces the error. The amount of computation required for learning is small due to weight sharing, and the trained network gives better results than multilayer perceptron.

Pooling layer is placed behind the convolution layer and has the function of reducing the calculation cost and strengthening it to a minute position change. The pooling method has an average pooling that takes the average value within each region and a maximum pooling that takes the maximum value within each region. Currently it is common to use maximum pooling.

Fully connected layer is generally used just before output, and it has the function of obtaining feature variable from feature data extracted by convolution layer and so on. The feature map obtained in the last convolution layer $f_{p,q}^{(r)}$ is input to the fully connected layer in the form of $a_{p*convx+q}$ vector.

$$f_{p,q}^{(r)} = a_{p*convx+q} \tag{2.9}$$

The error $\delta_{p,q}^{(r)}$ of the last convolution layer is expressed as follows using weights $w_{ji}^{(1)}$, $w_{kj}^{(1)}$ of the fully connected layer.

$$\begin{aligned}\delta_{p,q}^{(r)} &= \frac{\partial E_n}{\partial f_{p,q}^{(r)}} = \frac{\partial E_n}{\partial a_{p*convx+q}} \\ &= \sum_{j=1}^{J} \frac{\partial E_n}{\partial z_j} \frac{\partial z_j}{\partial a_{p*convx+q}} \\ &= \sum_{j=1}^{J} \frac{\partial E_n}{\partial y_k} \frac{\partial y_k}{\partial z_j} \frac{\partial z_j}{\partial a_{p*convx+q}} \\ &= \sum_{j=1}^{J} \delta_k w_{kj}^{(2)} w_{p*convx+qj}^{(1)}. \end{aligned} \tag{2.10}$$

The above equation shows that the error obtained from the fully connected layer is propagated to the last convolution layer.

A general CNN has a fully connected layer, but our proposed network does not use the fully connected layer because the output is an image. Such a network is called Fully Convolutional Network (FCN).

2.2 Network Architecture Search

The Deep Neural Networks network becomes deeper, wider, and more complex to achieve excellent performance, where complex neural architectures are designed manually. NAS, the process of automating architecture engineering, automatically finds the optimal neural network structure. NAS methods have outperformed manually designed architectures on some tasks such as image classification.

The illustration of NAS methods is shown in Fig. 2.2. A search strategy selects an architecture A from a predefined search space α. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

NAS methods have been proposed by many researchers. Zoph et al. [79] used a recurrent neural network as the controller to search the optimal neural network architecture in variable-length architecture space. Zoph et al. [80] proposed the NAS algorithm to search for an architectural building block on a small dataset, and then the block was transferred to a larger dataset. This approach is quite flexible as it may be scaled in terms of computational cost and parameters to quickly address a variety of problems. Pham et al. [49] proposed an efficient neural architecture search method by searching for an optimal subgraph within a large computational graph. Also, Cai et al. [5] proposed an efficient architecture search method based on a reinforcement learning agent as the meta-controller. Cai et al. [4] introduced ProxylessNAS that can directly learn neural network architectures on the target task and target hardware without any proxy. Liu et al. [36] and Dong and Yang [8] proposed a gradient-based NAS approach, that represents the search space as a directed acyclic graph. Real et al. [50] introduce the tournament selection evolutionary algorithm. Wu et al. [62] presented a differentiable neural architecture search framework that optimizes over a layer-wise search space and represents the search space by a stochastic supernet.

2.3 Network Pruning

Network pruning can efficiently prune redundant weights or filters to compress deep CNN while maintaining accuracy. Various method have been proposed.

Network pruning consists of the following steps: 1) Train the network with the sparse regularization. 2) Prune the weights with smaller influence on the loss.

There is two methods of network pruning in which unstructured pruning using unstructured sparse regularization and structured pruning using structured sparse regularization.

2.3.1 Sparse Regularization

In this section, we review sparse regularization criteria for pruning of deep neural networks.

We assume that the objective function of the optimization for determining the trainable weights is given by

$$J(W) = \mathbb{L}(f(x, W)|y) + \lambda \sum_{l=1}^{L} R(W^l) \tag{2.11}$$

where (x, y) denotes the pair of the input and target, W is a set of all trainable weights of all the L layers in the CNN, $\mathbb{L}(\cdot)$ is the standard loss for the CNN, and $R(W^l)$ is the regularization term at layer l. The parameter λ is used to balances the loss and the regularization term.

Also, we assume the weight in the layer l as $W^l \in \mathbb{R}^{C_l \times C_{l-1} \times K_l \times K_l}$, where C_l and C_{l-1} are the number of output channels and input channels, K_l is the kernel size of the layer l respectively. In the fully connected layers, $K_l = 1$

Unstructured Sparse Regularization

L2 Regularization The most popular and often used sparse regularization is the L2 regularization defined as

$$R_{L2}(W^l) = \|W^l\|_2^2 = \sum_{i}^{|W^l|} {w_i^l}^2, \tag{2.12}$$

where $|W^l|$ is number of elements of weights in the layer l, which is also called 2-norm and it can reduce variance of the model and suppressed over fitting. This regularization is often used in deep neural networks as weight decay.

L1 Regularization Tibshirami [59] proposed most simple non-structural Sparse Regularization L1 regularization for linear model, which is defined as

$$R_{L1}(W^l) = \|W^l\|_1 = \sum_{i}^{|W^l|} |w_i^l|, \tag{2.13}$$

which is also called 1-norm. L1 regularization works to make unnecessary individual parameters to be zero. In deep neural networks, L1 regularization is known as the primary method of sparse regularization to prevent overfitting by neglecting individual parameters both in the convolution layers and the fully connected layers. This regularization is also known as weight decay. However, it will be difficult to remove subsets of weights such as filters or channels on CNN.

Structured Sparse Regularization

Group Lasso Regularization

Yuan and Lin [72] and Schmidt [53] proposed group lasso regularization. In order to reduce subsets of weights like filters or channels, it is necessary to treat the subsets as groups in the regularization criterion. Yuan and Lin [72] and Schmidt [53] proposed this regularization for a linear model that can treat sets of parameters as a group in the criterion. Group lasso forces subsets of unnecessary parameters to be simultaneously zero. The regularization criterion of group lass is defined as

$$R_{GL}(W^l) = \sum_{g \in G} \|W_g^l\|_2 = \sum_{g \in G} \sqrt{\sum_i {w_{g,i}^l}^2}, \tag{2.14}$$

where $g \in G$ is a group in the set of groups G, W_g^l is the weight matrix or the weight vector for the group g that is a sub matrix or sub vector in W^l and $w_{g,i}^l$ is a weight with index i in the group g. Group lasso introduces sparseness at the group level and can reduce the number of active neurons or active filters. Alvarez et al. [2] proposed an approach to automatically determine the number of neurons in each layer of a DNN during learning, and they showed that group lasso regularization could reduce the number of parameters and even improve network accuracy. Wen et al. [61] proposed a structured sparsity learning (SSL) method to regularize the structures of deep neural networks by group lasso as structured sparse regularization. They introduced several structures of group lasso.

Sparse Group Lasso Regularization

Friedman et al. [11] and Simon et al. [54] proposed sparse group lasso by combining L1 regularization and group lasso, applied to linear regression. Sparse group lasso forces parameters to be zero at both the group and the individual feature level. Scardapane et al. [52] proposed to use sparse group lasso for deep neural networks. The criterion of the sparse group lasso is written as

$$R_{SGL}(W^l) = \alpha \sum_{g \in G} \|W_g^l\|_2 + (1 - \alpha)\|W^l\|_1, \tag{2.15}$$

where α is a balancing parameter to control strength of both group lasso and L1 regularization. By this combination, unnecessary parameters in the network can be pruned at both the group level and the individual feature level.

Exclusive Sparse Regularization

Zhou et al. [77] and Kong et al. [26] proposed exclusive lasso for multi-task feature selection. Exclusive lasso introduces competition among parameters in the same group and can prune neurons in neural networks. It is also called exclusive sparsity and the regularization criterion is defined as

$$R_{ES}(W^l) = \frac{1}{2} \sum_{g \in G} \|W_g^l\|_1^2 = \frac{1}{2} \sum_{g \in G} \left(\sum_i |w_{g,i}^l| \right)^2. \tag{2.16}$$

Combined Group and Exclusive Sparse Regularization

Yoon and Hwang et al. [69] proposed a pruning criterion called combined group and exclusive sparsity (CGES) for deep neural networks, which combines group lasso and exclusive sparse regularization. The authors claim that CGES can make the network sparse and also remove any redundancies among the features to fully utilize the capacity of the network.

Group $L_{1/2}$ Regularization

$L_{1/2}$ regularization, proposed by Xu et al. [65] [66] [73], can make the network to be more sparse than L1 regularization and much simpler than L0 regularization. Fan et al. [63] [9] applied $L_{1/2}$ regularization for pruning the neurons in the hidden layer of feedforward neural networks. Li et al. [30] [1] also applied a group $L_{1/2}$ regularization for feedforward neural networks. $L_{1/2}$ regularization can make not only the redundant hidden nodes to be zero but also the redundant weights of the surviving hidden nodes of the neural networks to be zero. In this paper, we define the criterion of the group $L_{1/2}$ regularization for deep neural network as

$$R_{GL_{1/2}}(W^l) = \sum_{g \in G} \|W_g^l\|_1^{1/2} = \sum_{g \in G} \sqrt{\sum_i |w_{g,i}^l|}. \tag{2.17}$$

Out-In-Channel Sparse Regularization

Li et al. [32] proposed Out-In-Channel Sparse Regularization (OICSR) for compact deep neural networks. In OICSR, the correlations between successive layers are taken into consideration to keep the predictive power of the compact network.

The Way of Grouping

To prune filters that connected unnecessary channels, there are three types of grouping, namely the filter-wise grouping, the neuron-wise grouping and the feature-wise grouping. The way of grouping for convolutional kernels are shown in Fig. 2.3.

In these groupings, structured sparse regularization treats subset of kernels as individual weight in the same group. So, mutual interaction between kernels in the group is not taken into account.

The Hierarchical Group Sparse Regularization

To introduce such interactions in the structured sparse regularization criterion, our previous work [25] propose the concept of the hierarchical group sparse regularization.

There are several possibilities to define the hierarchical interactions between kernels in the group for structured sparse regularization such as group lasso, exclusive sparsity and group $L_{1/2}$. They consider two ways of the integration, namely the square root of the sub-groups and the square of the sub-groups.

The hierarchical group sparse regularization criterion is defined by taking the square root of the sub-groups as

$$r_{SQRT}(W_g^l) = \sqrt{\sum_{k \in K} r(W_{g,k}^l)}, \tag{2.18}$$

where $k \in K$ is a kernel in the set of kernels K and $w^l_{g,k}$ is a kernel in the group g and $r(\cdot)$ is non-hierarchical group sparse regularization such as group lasso, exclusive sparsity and group $L_{1/2}$. In this criterion, the square root of the sub-groups are taken to defined the structured sparse regularization criterion. The hierarchical group sparse regularization criterion is also defined by taking the square of the sub-groups as

$$r_{SQ}(W^l_g) = \left(\sum_{k \in K} r(W^l_{g,k}) \right)^2 . \tag{2.19}$$

They also proposed the hierarchical group sparse regularization criterion combined the L1 regularization criterion, which can prune unnecessary weights at individual level and group level.

2.3.2 Unstructured Pruning

The way of unstructured pruning reduces the individual weight of neural networks [29, 14, 13, 12, 56, 39]. Optimal brain damage [29] and optimal brain surgeon [14] prune unimportant weight from a network to compute the influence of each weight on the training loss based on Hessian matrix. S. Han et al. [13] and C. Louizos et al.[39] utilized unstructured sparse regularizations such as L1 regularization to make a sparse networks, which reduce unnecessarily individual weights by enforcing to be 0. These unstructured pruning method makes network weights sparse, but it can only achieve speedup and compression with dedicated libraries and hardware.

2.3.3 Structured Pruning

Li et al. [31] proposed a method to prune filters with relatively low weight magnitudes based on weight norm. Hu et al. [20] proposed Average Percentage of Zeros to measure the percentage of zero activations of a neuron after the ReLU mapping. Wen et al. [61], Alvarez and Salzmann [2], Lebedev and Lempitsky [28], and Zhou et al. [75] utilized group sparse regularization during training to prune networks. Liu et al. [37] and Ye et al. [68] proposed pruning method with scaling parameter of batch normalization layers. Huang and Wang [23] applied a similar method to scale the outputs of specific structures, such as neurons, groups or residual blocks. He et al. [18] proposed a channels selection step based on lasso regression, and feature map reconstruction step with linear least squares. Luo et al. [41] proposed ThiNet that prune filters using statistics information computed from its next layer. Yu et al. [71] proposed the neuron importance score propagation algorithm to propagate the importance score of final responses to every neurons in the network. Molchanov et al. [46] proposed a criteria based on Taylor expansion that approximates the change in the cost function included by pruning network parameter. Suau et al. [64] proposed principal filter analysis, whose method exploited the intrinsic correlation between filter responses within network layers to recommend a smaller network footprint.

Huang et al. [22] proposed pruning method, that was totally automatic and data-driven and possible to control the tradeoff between network performance and its scale during pruning without involving humans in the loop. Chin et al. [6] proposed the layer-compensated pruning that uses meta-learning to learn a set of latent variables that compensated for the layer-wise approximation error and it was able to improve the performance for various heuristic metrics. He et al. [17] proposed soft filter pruning, that enabled the pruned filters to be updated when training the model after pruning. Lin et al. [34] proposed a novel global and dynamic pruning scheme to prune redundant filters. Lin et al. [33] proposed runtime neural pruning, that conducted pruning according to the input image and current feature maps adaptively. Wang et al. [60] introduced SkipNet, a modified residual network, that used a gating network to selectively skip convolutional blocks based on the activations of the previous layer. Mittal et al. [44] shows that even if randomly prune the filters, its performance after fine-tuning is not much worse than any of the above approaches such as [31] [46] [40] [20]. Zhu and Gupta [78] show that large-sparse (training a large model, but pruned to obtain a sparse model with a small number of nonzero parameters) models outperform comparably-sized small-dense models (with size comparable to the large-sparse model). Frankle and Carbin[10], Zhou et al. [76], and Morcos [47] find that a randomly-initialized, dense neural network contains a subnetwork (winning tickets) that is initialized such that when trained in isolation it can match the test accuracy of the original network after training for at most the same number of iterations. The winning tickets they find have won the initialization lottery: their connections have initial weights that make training particularly effective. Liu et al. [38] finds that fine-tuning a pruned model only gives comparable or worse performance than training that model with randomly initialized weights. They also show that the winning tickets initialization as used in Frankle and Carbin[10] only brings improvement when the learning rate is small (0.01), however such small learning rate leads to a lower accuracy than the widely used large learning rate (0.1). Peng et al. [48] proposed collaborative channel pruning, that quantitatively analyzed the joint influence of pruned/preserved channels to the final loss function, based the secondorder Taylor expansion. Molchanov et al. [45] described two variations of their method using the first and second order Taylor expansions to approximate a filter's contribution. Lin et al. [35] proposed a generative adversarial learning, that was able to jointly prune redundant structures, including filters, branches and blocks to improve the compression and speedup rates. Dong and Yang [7] proposed Transformable Architecture Search, which can search for the width and depth of the networks effectively and efficiently, the parameters of the searched/pruned networks are then learned by knowledge transfer. He et al. [16] proposed Filter Pruning via Geometric Median (FPGM) to prune the most replaceable filters containing redundant information. They also analyzed the norm-based criterion, which prunes the relatively less important filters.

2.4 Knowledge Distillation

Knowledge distillation can transfer the knowledge of DNNs with a large parameter (teacher networks) to smaller shallow networks (student networks). Ba et al. [3] proposed to use L2 loss between the input vectors of the softmax activation function (logits) of the teacher network and the student network. Hinton et al. [19] introduced to use the KL-divergence with a temperature parameter to make the softmax outputs of the teacher network and the softmax outputs (probability) of the student network similar. Romero et al. [51] introduced to map the student hidden layer to the prediction of the teacher hidden layer. Zhang et al. [74] presented a deep mutual learning (DML) strategy where, rather than one-way transfer between a static pre-defined teacher network and a student network, an ensemble of students learn collaboratively and teach each other throughout the training process.

2.5 Incremental Training

Incremental training algorithm is a dynamic configuration technique for DNNs that achieves energy-accuracy trade-offs in runtime by training a network incrementally as sub-networks. Tann et al. [58] proposed an incremental training algorithm in which the subsets of the weights in the network were incrementally trained by keeping the remaining weights trained in earlier steps. Xun et al. [67] proposed a dynamic DNNs using incremental training and group convolution pruning. In the dynamic DNNs, the channels of the convolution layer are divided into groups. At runtime, the following groups can be pruned for inference time/energy reduction or added back for accuracy recovery without model retraining. Istrate et al. [24] proposed an incremental training method that partitions the original network into sub-networks, which are then gradually incorporated in the running network during the training process. Yu et al. [70] introduced slimmable neural networks, that permit instant and adaptive accuracy-efficiency trade-offs at runtime by training divided networks.

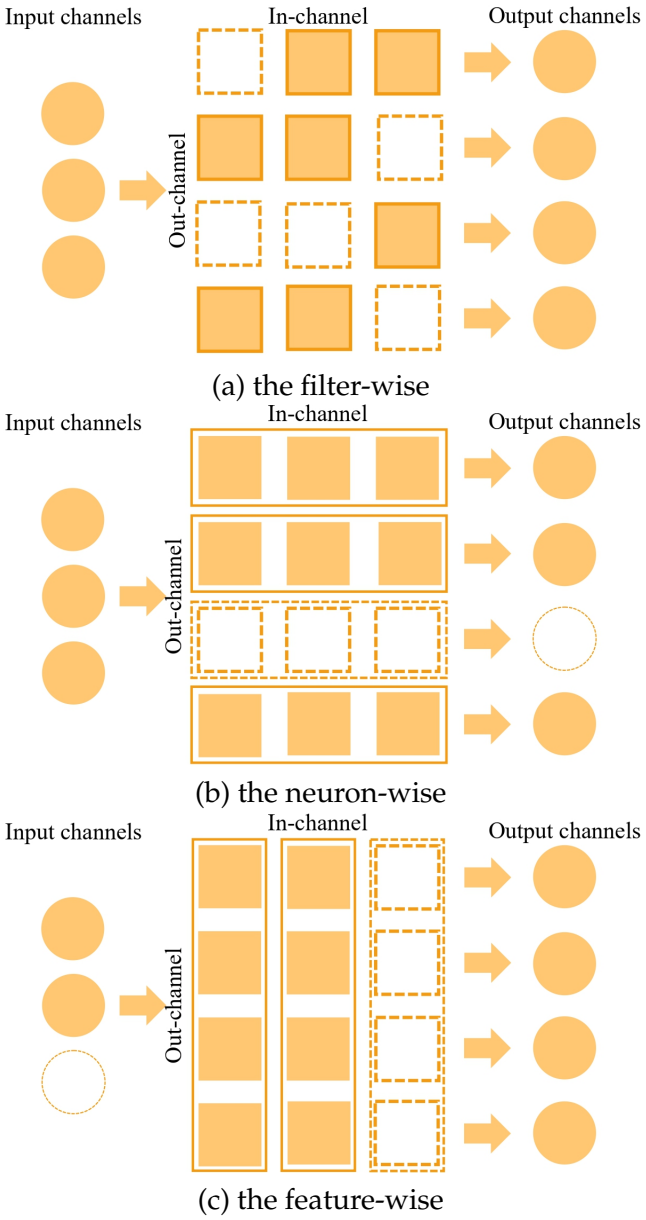


FIGURE 2.3: The way of grouping for convolutional filters. (a) the filter-wise grouping Each filter is considered as a group. We call this grouping the filter-wise grouping. By this grouping, we can prune unnecessary filters. (b) the neuron-wise grouping The weights connected to a output neuron are consider as a group. We call this grouping the neuron-wise grouping. By this grouping, we can prune unnecessary output neurons. (c) the feature-wise grouping The weights connected to a input neuron are considered as a group. We call this grouping the feature-wise grouping. By this grouping, we can prune unnecessary the output channels in $(l-1)^{th}$ layer (the input channels in $(l)^{th}$ layer).

Chapter 3

Filter Pruning using Hierarchical Group Sparse Regularization

3.1 Proposed Method

We propose the feature-wise filter pruning algorithm for deep convolutional neural networks. First, a brief description of the algorithm. Fig. 3.1 shows the flow-chart of the feature-wise filter pruning procedure. The illustration of the proposed the feature-wise filter pruning is shown in Fig. 3.2. It consists of the following steps; 1) Train the large network as the initial network. 2) Train the network with the structured sparse regularization based on the feature-wise grouping to find unnecessary filters connected to input channels by enforcing the weights of unnecessary filters to be zero. 3) Prune the filters with smaller influence on the classification loss. 4) Train the obtained compact network from scratch. 5) Obtain the compact network from scratch. The details of these steps are explained in the next sub-sections.

3.1.1 Training with The Hierarchical Sparse Regularization Based on The Feature-Wise Grouping

First, we train the network as the initial network without sparse regularization.

After obtaining the initial network, we train the network with the structured sparse regularization based on the feature-wise grouping such as group lasso, exclusive sparsity, group $L_{1/2}$ regularization, and the hierarchical group sparse regularization to get the sparse network. In this paper, we propose to use the hierarchical group sparse regularization, which is proposed by Mitsuno et al. [25], and show that the hierarchical group sparse regularization performs better than the non-hierarchical regularization.

The feature-wise group sparse regularization is defined as

$$R(W^l) = \sum_{j=1}^{c_{l-1}} r(W^l_{j,,}). \tag{3.1}$$

The structured sparse regularization criterion based on the feature-wise grouping enforces the filters connected to unnecessary input channels of the convolutional

FIGURE 3.1: Flow-chart of the feature-wise filter pruning procedure.

layer to be zero. As a result, we can remove unnecessary filters connected to output channels in layer $l-1$ by forcing the unnecessary filters connected to the input channels in layer l to be zero. Since the unnecessary input channels connected to the filters don't have influences to the outputs of the layer and final loss of the network, we can prune the filters connected to the output channels of the layer $l-1$ (the unnecessary input channels of the layer l).

In the next section, we explain the method to choose the filters that are pruned.

3.1.2 Filter Pruning

After training with the structured sparse regularization based on the feature-wise grouping, we obtain sparse network in which all weights of the filters connected to the unnecessary input channels are close to zero. Thus we can prune the filters connected to the unnecessary output channels of layer $l-1$ (the unnecessary input channels of the layer l) if the filters have less influence to the classification loss after pruning. We can implement the filter selection method for pruning as shown in Algorithm. 1.

Algorithm 1 Filter Pruning (backward filter selection)

Input: filter of the sparse model $W \in \mathbb{R}^L$, the number of conv layer L^c, the number of all output channel of the conv layer C^c, filter mask $m^l_{c^l} \in M^l$, training sample $s \in x$, loss $\mathbb{L}(\cdot)$, pruning rate P

Output: weight of pruned model $W^* \in \mathbb{R}^L$

$M = 1$
for n in $1 \ldots C^c P$ **do**
  for l in $1 \ldots L^c$ **do**
    for c in $1 \ldots C^c_l$ **do**
      $M^* = M$
      $m^{*l}_c = 0$
      $e_{l,c} = \mathbb{L}(f(s, W \odot M^*)|y)$
    end for
  end for
  $\underset{l^* \in L^c, c^* \in C^c_l}{\arg\min}\ e_{l^*,c^*}$ s.t. $m^{l^*}_{c^*} == 1$
  $m^{l^*}_{c^*} = 0$
end for
pruning $W^* = W[M == 1]$
return W^*

Molchanov et al. [46] proposed to use a criteria based on Taylor expansion for ranking and pruning one filter at a time. Chin et al. [6] also introduced a global ranking approach.

Here we use the global ranking for filter pruning to automatically obtain the pruned network architecture. We introduce a binary mask $m^l_{c^l} \in M^l$ for each output channel of the layer l, $m^l_i \in \{0,1\}$. By using these masks, we can prune i^{th} filter of the layer l by making $m^l_i = 0$. We search the filter which has the minimum loss increase after the filter is pruned. The increase of the classification loss after pruning is represented as

$$\underset{l^* \in L^c, i^* \in C^c_l}{\arg\min} \; |\mathbb{L}(f(s, W)|y) - \mathbb{L}(f(s, W \odot M)|y)| \tag{3.2}$$

where $W \odot M$ denotes the element-wise product. The classification loss $\mathbb{L}$ are evaluated by using randomly selected training samples. By updating the mask and repeating the filter selection algorithm until the number of the masked channels are $C^c P$ channels. After the filters for pruning are selected, we prune the filters from the network, where $m^l_i = 0$. We also prune the batch normalization layers connected to the pruned output channels at the same time. Then we can obtain the compact network architecture with small parameters and less computational operations.

The obtained compact network after pruning can achieve almost the same accuracy with the original large network by fine-tuninig. But Liu et al. [38] examined and showed that the fine-tuning of the pruned model can only give comparable or worse performance than the training of the compact model with randomly initialized weights. So we trained the obtained compact network from scratch. After that, we obtain small network with higher accuracy.

3.2 Experiments and Results

To confirm the effectiveness of the proposed pruning method, we have performed experiments using different data sets (CIFAR-10, CIFAR-100, and TinyImageNet-200) and different network architectures (VGG nets [55] and ResNet [15]). CIFAR-10 contains 60,000 color images of ten different animals and vehicles. They are divided into 50,000 training images and 10,000 testing images. The size of each image is 32×32 pixels. CIFAR-100 also contains 60,000 color images of 100 different categories and 50,000 images are used for training and the remainings are used for test. The size of each image is also 32×32 pixels. TinyImageNet-200 contains 110,000 color images of 200 different categories and 100,000 images are used for training and 10,000 images for test. The size of the image is 64×64 pixels.

We show the images of each dataset in Fig3.3.

In the following experiments, the number of channels of the network at each layer is adjusted to prevent overfitting, depending on each dataset.

3.2.1 Preliminary Experiments

In this section, we observe the difference of the sparse regularization and the difference of the sparsity of the network trained with the sparse regularization for pruning using ResNet18 on CIFAR-10.

Experimental Setting

Networks

For CIFAR-10, we trained ResNet20 (19 convolution(conv) + one fully connected(fc) layers) with batch normalization layers.

Initial Network

All the initial networks are trained from scratch by using SGD optimizer with a momentum of 0.9. We used the weight decay with the strength of $5 * 10^{-4}$ to prevent overfitting. The mini-batch size for CIFAR-10 was set to 128 and the network was trained for 200 epochs. The initial learning rate was set to 0.1 and it was divided by 0.2 after $[60, 120, 160]$ training epochs.

Train with Sparse Regularization

The sparse regularization was applied to the weights except for the bias term in all convolutional layers. All the networks were trained by using SGD optimizer with a momentum of 0.9. For CIFAR-10, the mini-batch size was set to 128 and the network was trained for 100 epochs. The initial learning rate of 0.01 which is divided by 0.1 after $1/3$ and $2/3$ training epochs. The hyper-parameter λ, which balances the cross-entropy loss and the sparse regularization criterion, was experimentally determined by grid search in the range from 10^{-1} to 10^{-7}.

We used hierarchical squared $\mathrm{GL}_{1/2}$ regularization (HSQ-GL12), hierarchical square rooted $\mathrm{GL}_{1/2}$ with L1 regularization (SHSQRT-GL12) and hierarchical squared $\mathrm{GL}_{1/2}$ with L1 regularization (SHSQ-GL12) [25] in the experiments to compare the sparse regularization criteria. We used HSQ-GL12 for sparse regularization in the experiments to compare the sparse regularization criteria. For SHSQRT-$\mathrm{GL}_{1/2}$, SHSQ-$\mathrm{GL}_{1/2}$, we set the parameter α, which balances the L1 regularization and the group sparse regularization criterion, to be 0.5. To evaluate the sparsity of the trained network, the ratio of the zero weights was calculated by assuming that the weights whose absolute value is less than 10^{-3} are zero.

After training the networks with sparse regularization, we selected one best trained network from the trained networks with the various hyper-parameter λ in sparse regularization.

Pruning and Training of the Pruned Networks

Then the channels at the convolutional layers except for the output channel of the final convolutional layer were pruned based on the influence of the pruning of each channel to the classification loss. To evaluate the influence of the pruning, we calculate the increase of the classification loss for the 128 randomly selected samples from the training samples.

The pruning rate P, which is a ratio of the pruned channels in the whole channels of the networks, was changed from 0.1 to 0.9. When the number of channels of a layer becomes 0, the experiments are stopped. After pruning, we can obtain compact networks. The parameters of the obtained compact network were trained from scratch using the same hyper-parameters as the training of the initial networks.

Results

The results on CIFAR-10 are shown in Fig. 3.4.

We calculate the ratio of the active parameters as the ratio of the number of whole parameters of the network and the remaining parameters after pruning.

From Fig. 3.4 (a), the performances of pruned networks with various sparse regularization are almost the same. For all experiments, we train the large networks with HSQ-GL12 for channel selection.

From Fig. 3.4 (b), it is better for pruning to use a sparser model for channel selection even if lower accuracy of the network than a baseline network. So we select a sparser network with the strong hyper-parameter λ for channel selection.

3.2.2 Pruning of VGG Nets

To confirm the effectiveness of the proposed pruning method, we have performed experiments using VGG14 (13-conv + 1-fc layers) with batch normalization layers for different data sets (CIFAR-10, CIFAR-100, and TinyImageNet-200).

All the initial networks are trained from scratch by using SGD optimizer with a momentum of 0.9. We used the weight decay with the strength of $5 * 10^{-4}$ to prevent overfitting. The mini-batch size for CIFAR-10/100 was set to 128 and the network was trained for 200 epochs. For TinyImageNet-200, the mini-batch size was set to 256 and the network was trained for 200 epochs. The initial learning rate was set to 0.1 and it was divided by 0.2 after $[60, 120, 160]$ training epochs.

Then the hierarchical sparse regularization was applied to the weights except for the bias term in all convolutional layers. All the networks were trained by using SGD optimizer with a momentum of 0.9. For CIFAR-10/100, the mini-batch size was set to 128 and the network was trained for 100 epochs. For TinyImageNet-200, the mini-batch size was set to 256 and the network was trained for 100 epochs. The initial learning rate of 0.01 which is divided by 0.1 after $1/3$ and $2/3$ training epochs. The hyper-parameter λ, which balances the cross-entropy loss and the hierarchical

sparse regularization criterion, was experimentally determined in the range from 10^{-1} to 10^{-7}.

We used hierarchical squared $GL_{1/2}$ regularization (HSQ-GL12)[25], which has shown outstanding performance compare to various sparse regularization criteria in our previous work. To evaluate the sparsity of the trained network, the ratio of the zero weights was calculated by assuming that the weights whose absolute value is less than 10^{-3} are zero.

After training the networks with sparse regularization, we selected one best trained network from the trained networks with the various hyper-parameter λ in sparse regularization.

To obtain the compact network, the channels at the convolutional layers except for the output channel of the final convolutional layer were pruned based on the influence of the pruning of each channel to the classification loss. To evaluate the influence of the pruning, we calculate the increase of the classification loss for the 128 randomly selected samples from the training samples.

The pruning rate P, which is a ratio of the pruned channels in the whole channels of the networks, was changed from 0.1 to 0.9. When the number of channels of a layer becomes 0, the experiments are stopped. After pruning, we can obtain compact networks. The parameters of the obtained compact network were trained from scratch using the same parameter settings as the training of the initial networks.

The results for VGG14 are shown in Fig. 3.5. For CIFAR-10, the propose method is scceeded to prune 85% of parameters with only 0.53% drop of the test accuracy. It is impressive that the network in which 41% parameters are pruned achieves better test accuracy than the baseline network. Similarly, we can prune 41% of parameters with only 0.85% drop of the test accuracy for CIFAR-100. For TinyImageNet-200, 37% of parameters can be pruned with only 0.70% drop of the test accuracy.

For all datasets, we observe similar tendency of pruning such that the parameters in the deep layer are pruned much more than the shallow layer.

3.2.3 Pruning of ResNet

We also have performed experiments with ResNet for CIFAR-10, CIFAR-100, and TinyImageNet-200. For CIFAR-10/100, we trained ResNet20 (19 convolution(conv) + one fully connected(fc) layers) and ResNet32 (31-conv + 1-fc layers) with batch normalization layers. For TinyImageNet-200, ResNet18 (17 convolution(conv) + one fully connected(fc) layers) and ResNet34 (33-conv + 1-fc layers) were trained with batch normalization layers. Parameter settings are the same with the experiments for VGG net.

The results with ResNet are shown in Fig. 3.6 and Fig. 3.7. By using the proposed pruning method, we can obtain the compact network which has only 54% of the parameters with the baseline network but can achieve almost same test accuracy with the baseline network (only 0.31% drop) for ResNet20 on CIFAR-10. The pruned network with 16% parameters gives better test accuracy than the baseline network.

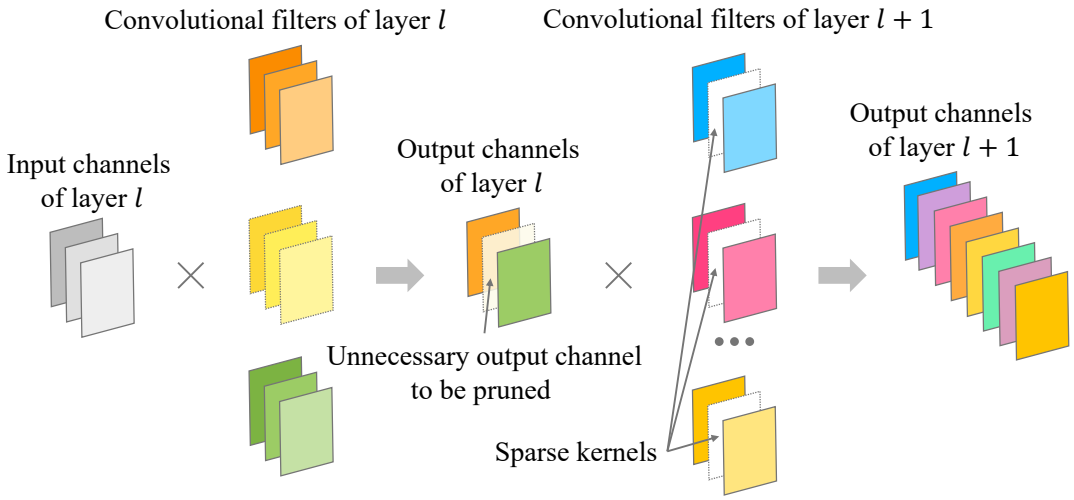
Similaly, for ResNet20 on CIFAR-100, the network with 15% parameters gives almost same test accuracy with the baseline network (only 0.85% drop). The network pruned 34% of the parameters for ResNet18 on TinyImageNet-200 achieved higher test accuracy than the baseline network. The network pruned 54% of the parameters for ResNet32 on CIFAR-10 and the network pruned 13% of the parameters for ResNet32 on CIFAR-100 are also gives the almost same test accuracy (only 0.73% drop and only 1.2% drop). For ResNet32 on TinyImageNet-200, we can prune the network for 16% of the parameters with only 3.8% drop of the test accuracy. From the right figures, it is noticed that the parameters in the deep layer of each residual block are pruned more.

3.2.4 Comparison with The State-of-the-art Method

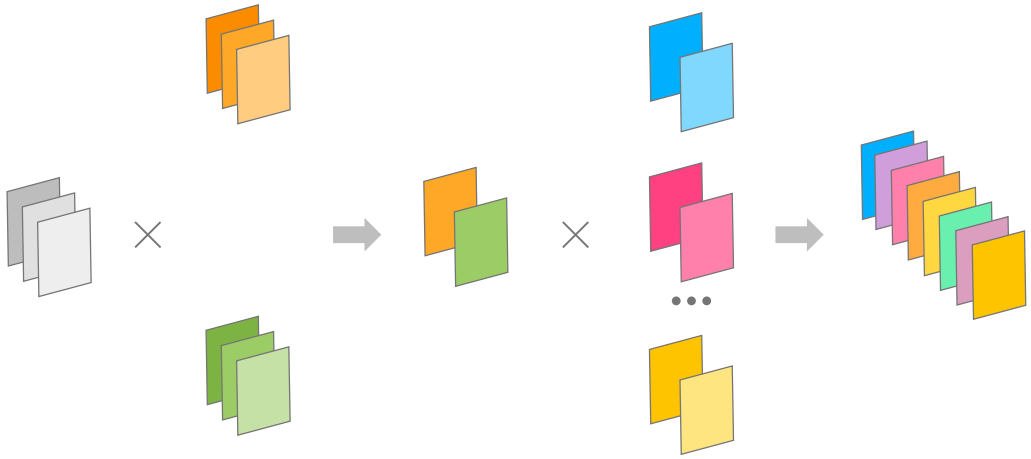
The proposed method was compared with one of the state-of-the-art filter pruning methods FPGM-mix. The filter pruning via geometric median (FPGM) [16] is one of the state-of-the-art method and FPGM-mix is a mixture of FPGM and their previous norm-based method [17]. We have performed experiments with the same pruning rate of P. The ratio of FPGM and the norm-based method is determined according to the paper [17]. Namely, $3/4$ of the filters are selected with FPGM and the remaining $1/4$ filters are selected with the norm-based criterion.

The results of the comparisons are shown in Fig.3.5, Fig.3.6 and Fig.3.7. For VGG nets, ResNet20 and ResNet18, the performance of the pruned network with the proposed method is better than the FPGM-mix for all pruning ratios. For ResNet32 and ResNet34, the test accuracy of the pruned network with the proposed method gives better than the network obtained by FPGM-mix when the pruning rate is larger than 60%. Our proposed method consider the structure of the networks to prune unnnecessary parameters. So that, the proposed method gives better performance than the FPGM-mix when the pruning rate is high.

These results show the effectiveness of the proposed method compares to the state-of-the-art method, especially when the network is pruned more than 50% of the parameters.

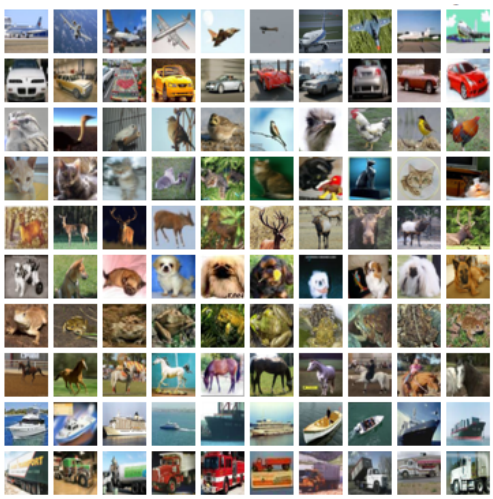


(a) Sparse kernels of layer $l+1$ which are connected to the unnecessary output channel of layer l

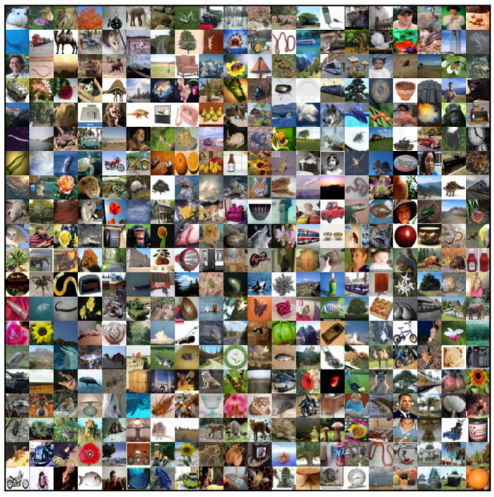


(b) Pruning the filter of layer l and sparse kernels of layer $l+1$, which are connected to the unnecessary output channel of layer l keeping the same output channel of layer $l+1$

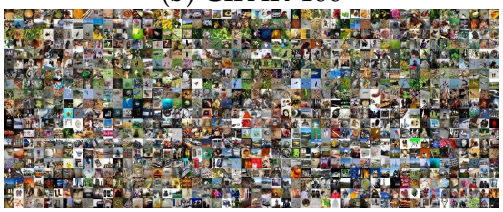
FIGURE 3.2: An illustration of filter pruning via Hierarchical sparse group regularization based on the feature-wise grouping. In convolutional layer, each filter makes one output channel (activation), these colors are the same. For example, the filter of orange of layer l makes orange output channel of layer l. (a) The Hierarchical sparse group regularization based on the feature-wise grouping make the weights of the unnecessary kernels to be almost zero. Since the output of convolution from the input channel connected to the unnecessary kernels will be zero in the layer $l+1$, the output channels are not influenced by the pruning the unnecessary kernel. (b) If the increase of the classification loss of the network after pruning the filters connected to the unnecessary output channel of layer l is very small, we can prune the filters of layer l and kernels of layer $l+1$ connected to the unnecessary output channel of layer l. Then the output channels of layer $l+1$ are almost the same as the output channels before pruning.



(a) CIFAR-10

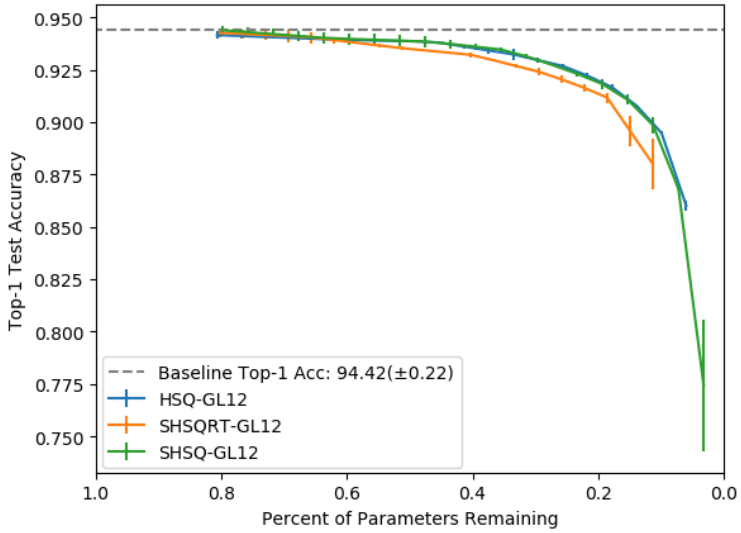


(b) CIFAR-100

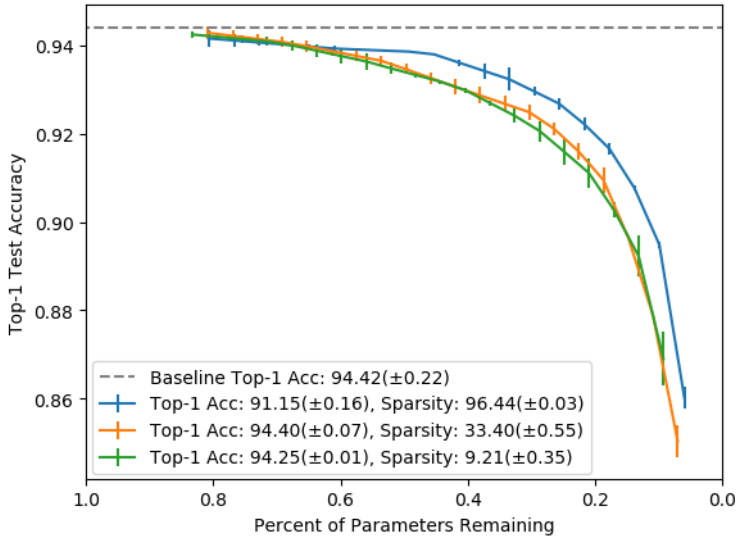


(c) TinyImageNet-200

FIGURE 3.3: This Figure shows images of each dataset CIFAR-10, CIFAR-100, and TinyImageNet-200

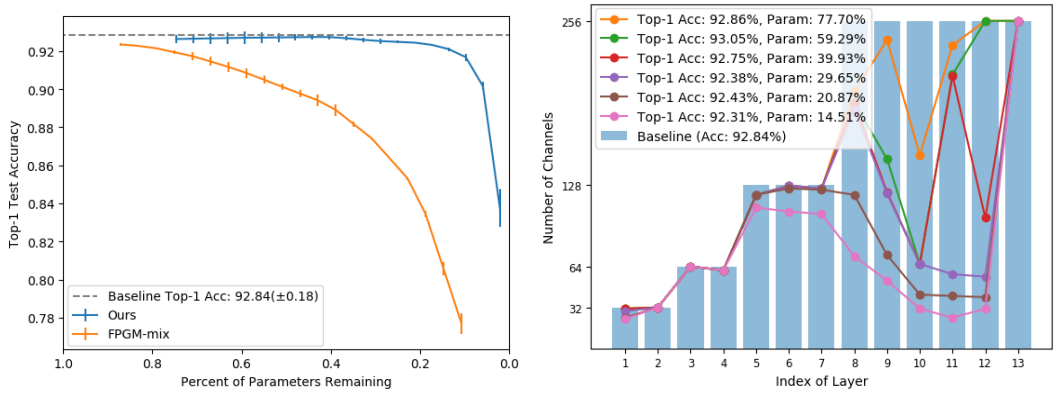


(a) Comparison of pruned networks with various sparse regularizations

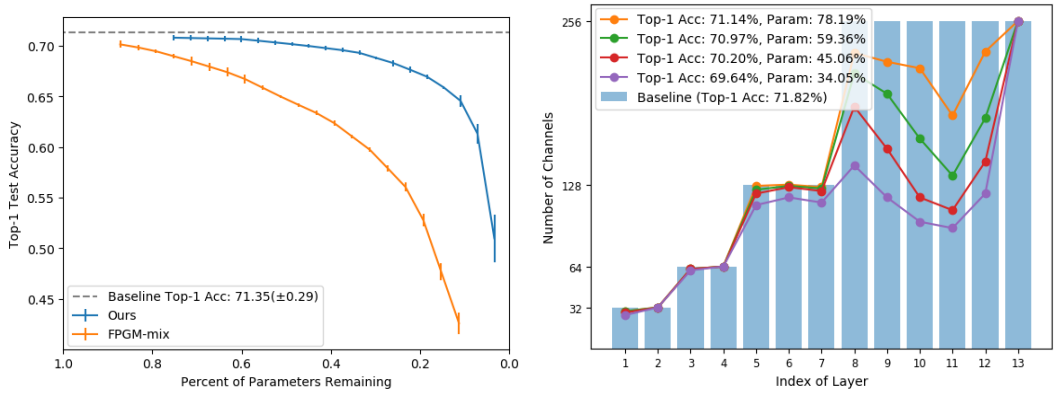


(b) Comparison of pruned networks with various sparsity

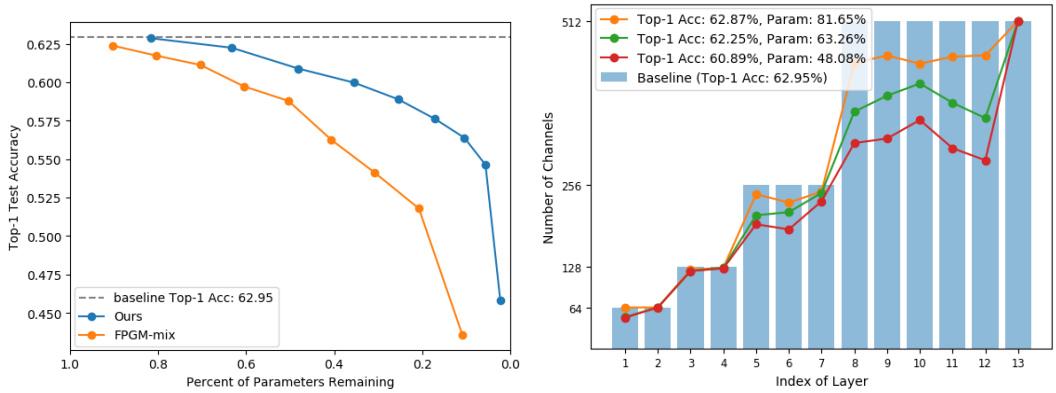
FIGURE 3.4: The results with ResNet20 on CIFAR-10 (a) Comparison of test accuracy of pruned networks with different sparse regularization for training the sparse network (average of three trials). (b) Comparison of test accuracy of pruned networks with different sparsity of the networks that trained with sparse regularization (average of three trials).



(a) VGG14 on CIFAR-10

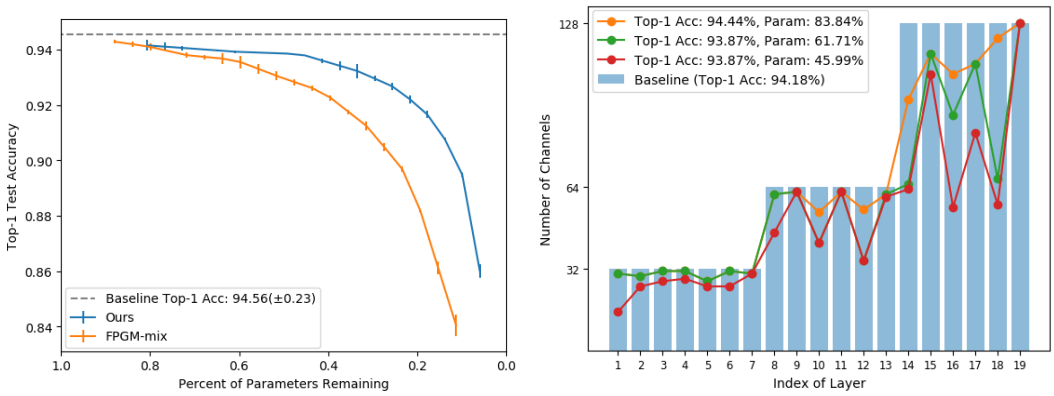


(b) VGG14 on CIFAR-100

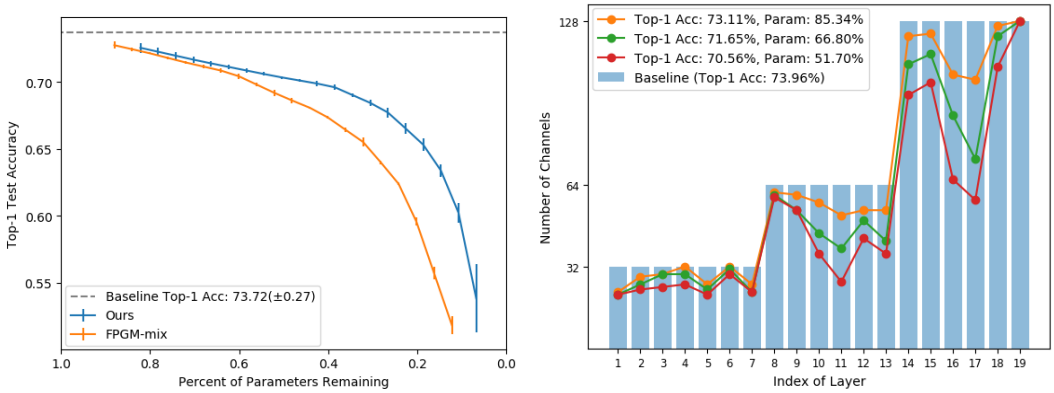


(c) VGG14 on TinyImageNet-200

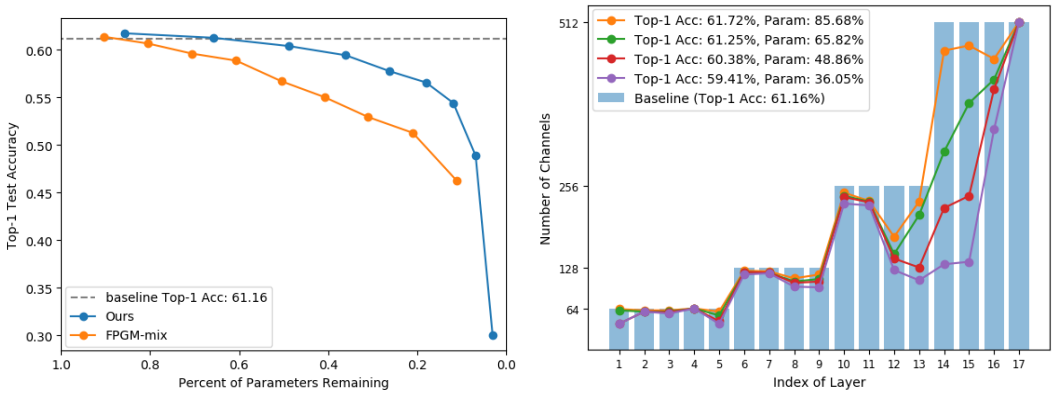
FIGURE 3.5: The results with VGG14 on CIFAR-10/100 and TinyImageNet-200. Comparison of test accuracy of the pruned networks (left). For CIFAR-10/100, the average of three trials are shown. The numbers of the pruned channels in each layer is shown in the right figure. Each line shows the numbers of the pruned channels at each layer. Different colors denote the results with different pruning rates P. Param means ratio of parameter remaining of a pruned networks.



(a) ResNet20 on CIFAR-10

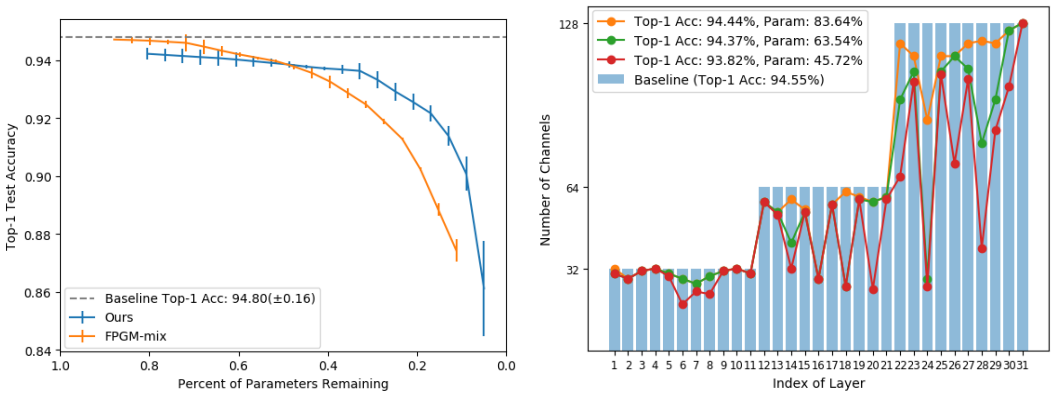


(b) ResNet20 on CIFAR-100

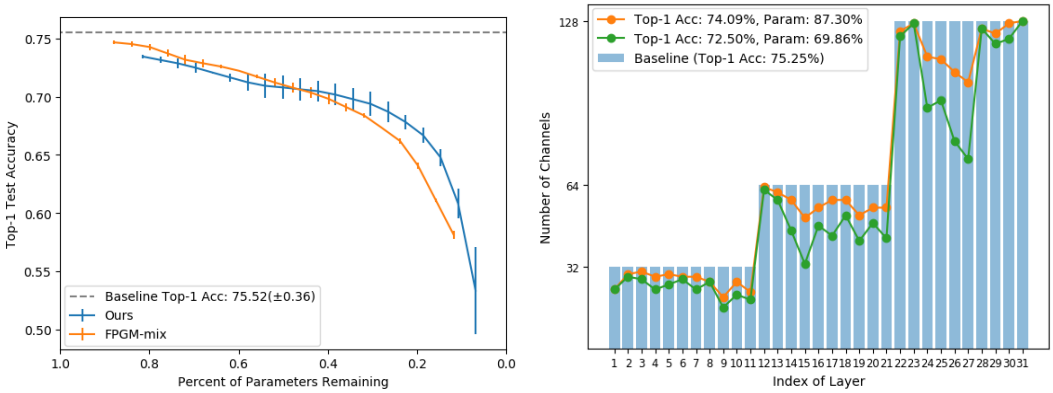


(c) ResNet18 on TinyImageNet-200

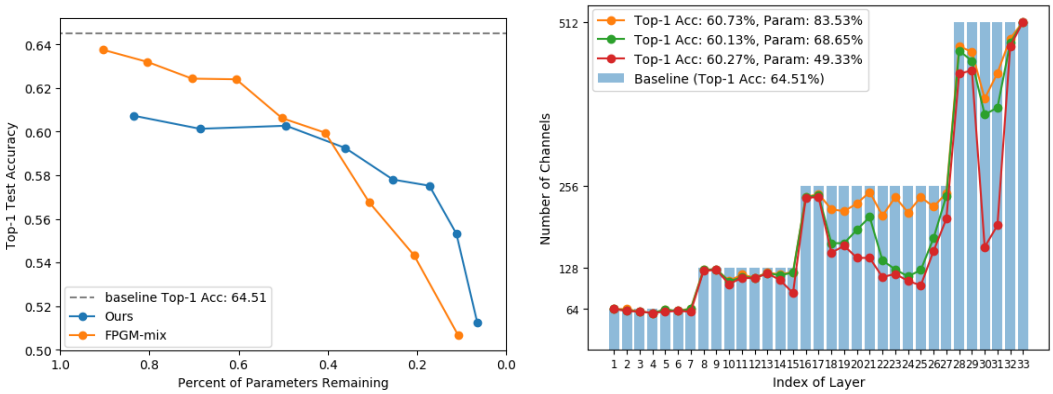
FIGURE 3.6: The results with ResNet20 and ResNet18. Comparison of test accuracy of the pruned networks (left). The average of three trials is shown. The numbers of the pruned channels in each layer of the networks are shown in the right figure.



(a) ResNet20 on CIFAR-10



(b) ResNet20 on CIFAR-100



(c) ResNet18 on TinyImageNet-200

FIGURE 3.7: The results with ResNet32 and ResNet34. Comparison of test accuracy of pruned networks (left). For CIFAR-10/100, the average of three trials is shown. Right figure shows the numbers of pruned channels in each layer of the networks.

Chapter 4

Channel Planting using Knowledge Distillation

4.1 Proposed Method

In this section, we propose a novel incremental training method for DNNs called *planting*. In the proposed incremental training method, channels on a small network are incrementally added to improve classification accuracy. The parameters of the added channel are trained by using the knowledge distillation to imitate the behavior of the large network (the teacher network). The optimal network architecture is searched by incrementally selecting the best channels among the possible candidates of the additions in terms of the classification accuracy (on the validation set). The illustration of the proposed planting procedure on a typical DNN is shown in Fig. 4.1.

In summary, our planting approach consists of the following training processes: (0) training a large network as the teacher network. (1) training a small network with fewer channels of each layer by a standard classification training method. (2) incrementally adding channels on the small network by using a knowledge distillation method with the teacher network.

4.1.1 Planting Approach

Preparation. We assume that the objective function of the optimization for determining the trainable weights is given by

$$J(W) = \mathbb{L}(f(x, W)|y) \tag{4.1}$$

where (x, y) denotes the pair of the input and target, W is a set of all trainable weights of all the L layers in the CNN, $\mathbb{L}(\cdot)$ is the standard loss for the CNN.

Also, we assume the weight in the layer l as $W^l \in \mathbb{R}^{C^l \times C^{l-1} \times K^l \times K^l}$, where C^l and C^{l-1} are the number of output channels and input channels, K^l is the kernel size of the layer l respectively. In the fully connected layers, $K^l = 1$.

Initial network. First, we train a small network with a few channels of each layer by a standard classification training procedure. Also, we train a large network by a

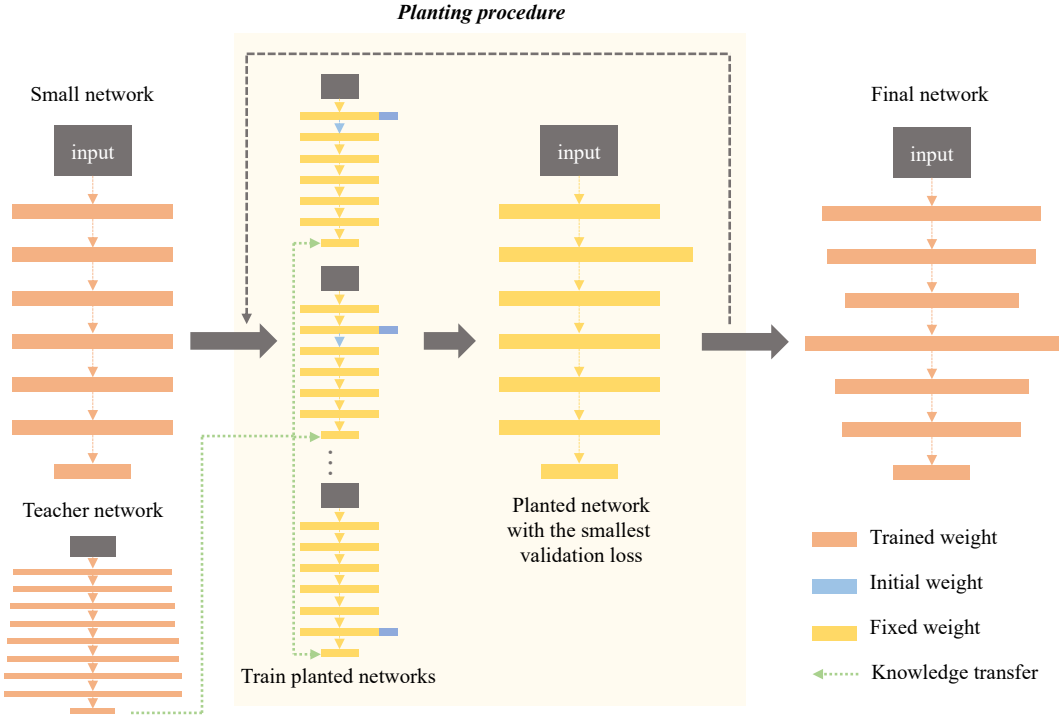


FIGURE 4.1: Illustration of Planting Procedure on a typical DNNs

standard classification training procedure as the teacher network. It is expected that the teacher network has the optimized number of channels with maximum performance in terms of classification accuracy.

Search for the best layer for planting. For considering the impact of planted channels on a network, we divide the layer of the small networks into several groups G, where $1 \leq G \leq L$. When using a very deep network, we plant channels in multiple layers by dividing a small network into groups to increase the impact of the planting. Then, we add n channels on the layer of group g. The number of channels of the added layer is given by

$$C^l + n \;\; s.t. \;\; g * \frac{L}{G} \leq l < (g+1) * \frac{L}{G}. \tag{4.2}$$

The weights of the added channels of the small networks are trained by a knowledge distillation procedure while keeping the remaining weights are fixed. The planted channels are learned to reduce the classification loss of the small network. For example, when adding n channels on the layer l, the weights $W^l_{C^l:C^l+n,:,:,:}$ and $W^{l+1}_{:,C^l:C^l+n,:,:}$ are trained.

We search the best group g which minimizes the loss

$$\arg\min_{g} J_{KL}(W^{S'}, W^L), \tag{4.3}$$

where $W^{S'}$ is the small network with the additional layer of group g and W^L is the large network. The detail definition of the loss $J_{KL}(W^{S'}, W^L)$ is explained in the next

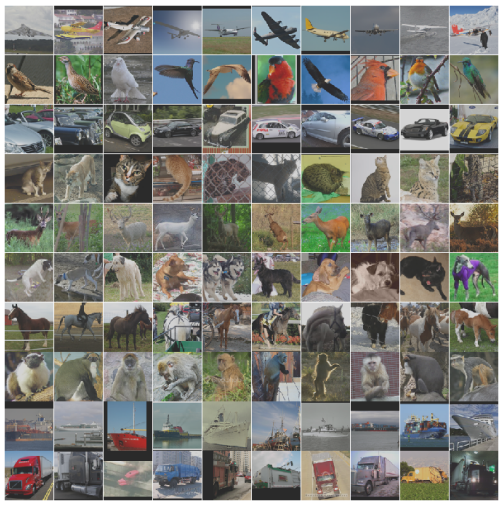


FIGURE 4.2: This Figure shows images of each dataset STL-10

sub section. The best layer to add is searched by using the brute-force search method or the random search method if there are many groups. In the random search, some groups from G are randomly selected to reduce the calculation cost, and the best group is determined from the selected groups. After we determined the best layer to reinforce, fix the planted channels and explore the next channel. By repeating this planting process while reducing the classification loss than the previous network, we can obtained the best network architecture.

After this method, we obtain a small network with fewer channels, which has higher performance than the networks obtained in a standard training procedure and can prevent over-fitting. The network architecture is automatically optimized by the proposed planting procedures. The details of the planting algorithm is shown in Algorithm 2.

4.1.2 Knowledge distillation

Knowledge distillation is an effective method for training the small network. In this study, we employ the Kullback Leibler (KL) Divergence. Suppose the predictions by the small network and the large network are z^S and z^L respectively, the KL divergence from z^S to z^L is given by

$$\mathbb{L}_{\mathbb{KL}}(z^L||z^S) = \sum_i \frac{\exp z_i^L}{\sum_j \exp z_j^L} log\left(\frac{\exp z_i^S}{\sum_j \exp z_j^S}\right). \tag{4.4}$$

The objective function for the proposed planting method is defined as follows

$$\begin{aligned} J_{KL}(W^S, W^L) &= \lambda \mathbb{L}(f(x, W^S)|y) \\ &+(1-\lambda)\mathbb{L}_{\mathbb{KL}}(f(x, W^L)||f(x, W^S)), \end{aligned} \tag{4.5}$$

Algorithm 2 Planting algorithm

Input: W^S : trained small network, W^L : trained teacher network, G : the number of group, n : the number of planted channels, (x_{train}, y_{train}) and (x_{val}, y_{val}) : the training samples and the validation samples obtained by splitting the training set into two disjoint subsets.

while 1 **do**
  for g in $1 \dots G$ **do**
    $W^{S_g} = W^S$
    for l in $1 \dots L$ **do**
      if $g * \frac{L}{G} \leq l < (g+1) * \frac{L}{G}$ **then**
        plant n channels on layer l of W^{S_g}
      end if
    end for
    train W^{S_g} via $J_{KL}(W^{S_g}, W^L)$ on (x_{train}, y_{train})
  end for
  $g_{min} = \arg\min_{g \in G} |J(W^{S_g})|$ on (x_{val}, y_{val})
  if $J(W^{S_{g_{min}}}) \geq J(W^S)$ on (x_{val}, y_{val}) **then**
    break
  end if
  $W^S = W^{S_{g_{min}}}$
end while

$$s.t.\ 0 \leq \lambda \leq 1$$

where λ is used to balance the standard classification loss $\mathbb{L}(f(x, W^S)|y)$ and KL divergence $\mathbb{L}_{\mathbb{KL}}(f(x, W^L)||f(x, W^S))$.

4.2 Experiments and Results

To confirm the effectiveness of the proposed method, we have performed experiments with the image classification task using different datasets (CIFAR-10, CIFAR-100, and STL-10).

4.2.1 Experiments using CIFAR-10

In the experiments on CIFAR-10, we used the 7-layers CNN models with five convolutional layers and two fully connected layers, the structure of the network is shown in Table.4.1. All the experiments, we set the number of channels of the fully connected layers to $[128, 10]$. All the number of channels of convolutional layers were set to 8 for initial network and 128 for the teacher network.

The initial network and the teacher network were trained from scratch by using SGD optimizer with a momentum of 0.9. We used the weight decay with the strength of $5 * 10^{-4}$ to prevent over-fitting. The mini-batch size for CIFAR-10 was set to 128 and the network was trained for 150 epochs. The initial learning rate was set to 0.01 and it was multiplied by 0.2 after $[40, 80, 120]$ training epochs.

TABLE 4.1: The structure of networks

For CIFAR-10/100	For STL-10
ReLU(conv1(kernel=3))	ReLU(conv1(kernel=3))
max pooling(2*2)	max pooling(2*2)
ReLU(conv2(kernel=3))	ReLU(conv2(kernel=3))
max pooling(2*2)	max pooling(2*2)
ReLU(conv3(kernel=3))	ReLU(conv3(kernel=3))
ReLU(conv4(kernel=3))	max pooling(2*2)
ReLU(conv5(kernel=3))	ReLU(conv4(kernel=3))
max pooling(2*2)	ReLU(conv5(kernel=3))
ReLU(fc1())	max pooling(2*2)
output=fc2()	ReLU(fc1())
	output=fc2()

In the planting operation, we used the weight decay with the strength of $5 * 10^{-5}$, the number of the group G was set to 5, and other parameter settings are the same with the training of the initial network. We added 4 channels to the layers at one planting operation. In the training of planted channels, the hyper-parameter λ of KL loss (KLLoss) was set to 0. In the calculation for finding the smallest validation loss, the hyper-parameter λ of KLLoss was set to 1.

For comparing the performance of the proposed method, we trained the baseline networks with cross entropy loss (CELoss) as the standard classification loss, and KLLoss as loss function of knowledge transfer. All the number of channels of the convolutional layer for the baseline networks were set to 8, 16, 32, 64 and 128, and we used the same teacher networks with the planting operation. The hyper-parameter λ of KLLoss was set to 0. Parameter settings are the same with the training of the initial network.

TABLE 4.2: Results on CIFAR-10 dataset. The average of three trials are shown.

Network	Params	Test Err.	Test Acc.	Loss func
Teacher[128]	857.5K	0.5007	88.10%	CELoss
Student[128]		0.3823	88.51%	KLLoss
Initial Network	20.4K	0.8300	71.55%	CELoss
(Student[8])		0.8245	71.69%	KLLoss
Student[16]	43.9K	0.6071	79.42%	CELoss
		0.6108	79.23%	KLLoss
Student[32]	104.8K	0.4898	84.03%	CELoss
		0.4791	84.02%	KLLoss
Student[64]	282.0K	0.4431	86.83%	CELoss
		0.4103	86.80%	KLLoss
Ours	40.6K	0.4825	84.35%	KLLoss

The results for CIFAR-10 are shown in Table 4.2. In this table, the average of three trials are shown. The number of channels of the convolutional layers after planting

operation were $[12, 20, 16, 16, 12]$, $[12, 16, 16, 16, 16]$ and $[12, 16, 16, 16, 16]$. For CIFAR-10, the proposed method is succeeded to train a network with higher classification accuracy, which has only 39% parameters compare to a network where all the convolutional layers are 32 channels.

4.2.2 Experiments using CIFAR-100

In the experiments on CIFAR-100, we used the same network structures with the experiments on CIFAR-10. All the experiments, we set the number of channels of the fully connected layers to $[128, 100]$. All the number of channels of convolutional layers were set to 16 for the initial network and 128 for the teacher network. In the calculation for finding the smallest validation loss, the hyper-parameter λ of KLLoss was set to 0. Other parameter settings are the same as the experiments on CIFAR-10. For comparison of the performance of the proposed method with the standard methods, we trained the baseline networks on the settings of the same experiment with the experiments on CIFAR-10.

TABLE 4.3: Results on CIFAR-100 dataset. The average of three trials are shown.

Network	Params	Test Err.	Test Acc.	Loss func
Teacher[128]	869.1K	2.5010	57.76%	CELoss
Student[128]		1.6232	60.05%	KLLoss
Student[8]	32.0K	2.5280	36.53%	CELoss
		2.5053	36.90%	KLLoss
Initial Network (Student[16])	55.5K	2.1190	45.45%	CELoss
		2.0679	46.66%	KLLoss
Student[32]	116.5K	1.9022	52.15%	CELoss
		1.7805	53.72%	KLLoss
Student[64]	293.6K	1.9510	55.74%	CELoss
		1.6707	57.71%	KLLoss
Ours	78.5K	1.7584	54.31%	KLLoss

The results for CIFAR-100 are shown in Table 4.3. In this table, the average of three trials are shown. The number of channels of the convolutional layers after planting operation were $[20, 24, 20, 24, 24]$, $[20, 24, 20, 24, 24]$ and $[20, 24, 24, 24, 20]$. For CIFAR-100, the proposed method is succeeded to train a network with higher classification accuracy, which has only 67% parameters compare to a network where all the convolutional layers are 32 channels.

4.2.3 Experiments using STL-10

STL-10 contains 13,000 color images of ten animals and vehicles. The size of the image is 96×96 pixels. They are divided into 5,000 training images, 1,000 validation images and 7,000 testing images.

We show the images of STL-10 dataset in Fig4.2.

In the experiments on STL-10, we used the 7-layers CNN models with five convolutional layers and two fully connected layers, the structure of the network is shown in Table.4.1. In all the experiments, we set the number of channels of the fully connected layers to $[128, 10]$. All the number of channels of convolutional layers were set to 8 for initial network and 64 for the teacher network.

The network was trained for 100 epochs, the initial learning rate was set to 0.01 and it was multiplied by 0.1 after every $epoch/3$ training epochs. In the planting operation, we used the weight decay with the strength of $5 * 10^{-4}$. In the calculation for finding the smallest validation loss, the hyper-parameter λ of KLLoss was set to 0. Other parameter settings are the same with with the experiments on CIFAR-10.

For comparing the performance of the proposed method, we trained the baseline networks on the settings of the same experiment with the experiments on CIFAR-10.

TABLE 4.4: Results on STL-10 dataset. The average of three trials are shown.

Network	Params	Test Err.	Test Acc.	Loss func
Teacher[64]	445.8K	1.5360	66.33%	CELoss
Student[64]		1.1807	66.47%	KLLoss
Initial Network	40.8K	1.2776	55.55%	CELoss
(Student[8])		1.2682	54.99%	KLLoss
Student[16]	84.9K	1.2924	59.34%	CELoss
		1.1998	61.10%	KLLoss
Student[32]	186.8K	1.2213	64.57%	CELoss
		1.1712	64.07%	KLLoss
Student[128]	1.2M	1.7612	67.04%	CELoss
		1.1643	67.71%	KLLoss
Ours	82.6K	1.0772	67.12%	KLLoss

The results for STL-10 are shown in Table 4.4. Again the average of three trials are shown in this table. The number of channels of the convolutional layers after planting operation were $[28, 20, 20, 12, 12]$, $[28, 16, 20, 20, 16]$ and $[12, 20, 16, 28, 16]$. For STL-10, the proposed method is succeeded to train a network with higher classification accuracy, which has only 7% parameters compare to a network where all the convolutional layers are 128 channels with CELoss. The test loss of planted network is the smallest than all the comparison networks and also the planting method can train to reduce over-fitting.

Chapter 5

Conclusion

We propose a two types of NAS method, one is a filter pruning method, the other is a incremental training method.

First, we propose a filter pruning method with the hierarchical group sparse regularization based on the feature-wise grouping for pruning filters, which connected to unnecessary input channels, using the influence of the classification loss. At first the network was trained with the hierarchical sparse regularization. We take the strategy of the step-wise pruning of the filters by searching the filter with the minimum loss increase. Then the obtained compact network was retrained from scratch. Experiments using CIFAR-10/100 and TinyImageNet-200 datasets show the outstanding performance than the state-of-the-art pruning method. Especially the performance of the pruned network is better than the state-of-the-art pruning method when more than 50% of the parameters are pruned.

Finally, we proposed a novel incremental training algorithm for deep neural networks called planting. Our planting approach can automatically search the optimal network architecture for training tasks with smaller parameters by planting channels incrementally to layers of the initial networks while keeping the earlier trained channels fixed for improving the network performances. Also, we proposed to use the knowledge distillation method for training the channels planted. By transferring the knowledge of deeper and wider networks, we can grow the networks effectively and efficiently. We evaluated the effectiveness of the proposed method on different datasets. We confirmed that the proposed approach was able to achieve comparable performance with smaller parameters compare to the larger network and reduce the over-fitting caused by a small amount of the data.

Bibliography

[1] Habtamu Zegeye Alemu et al. "Group $L_{1/2}$ regularization for pruning hidden layer nodes of feedforward neural networks". In: *IEEE Access* 7 (2019), pp. 9540–9557.

[2] Jose M Alvarez and Mathieu Salzmann. "Learning the number of neurons in deep networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2270–2278.

[3] Jimmy Ba and Rich Caruana. "Do deep nets really need to be deep?" In: *Advances in neural information processing systems*. 2014, pp. 2654–2662.

[4] Han Cai, Ligeng Zhu, and Song Han. "Proxylessnas: Direct neural architecture search on target task and hardware". In: *arXiv preprint arXiv:1812.00332* (2018).

[5] Han Cai et al. "Efficient architecture search by network transformation". In: *Thirty-Second AAAI conference on artificial intelligence*. 2018.

[6] Ting-Wu Chin, Cha Zhang, and Diana Marculescu. "Layer-compensated pruning for resource-constrained convolutional neural networks". In: *arXiv preprint arXiv:1810.00518* (2018).

[7] Xuanyi Dong and Yi Yang. "Network pruning via transformable architecture search". In: *Advances in Neural Information Processing Systems*. 2019, pp. 759–770.

[8] Xuanyi Dong and Yi Yang. "Searching for a robust neural architecture in four gpu hours". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1761–1770.

[9] Qinwei Fan, Jacek M Zurada, and Wei Wu. "Convergence of online gradient method for feedforward neural networks with smoothing $L_{1/2}$ regularization penalty". In: *Neurocomputing* 131 (2014), pp. 208–216.

[10] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *International Conference on Learning Representations*. 2019.

[11] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. "A note on the group lasso and a sparse group lasso". In: *arXiv preprint arXiv:1001.0736* (2010).

[12] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *arXiv preprint arXiv:1510.00149* (2015).

[13] Song Han et al. "Learning both weights and connections for efficient neural network". In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[14] Babak Hassibi and David G Stork. "Second order derivatives for network pruning: Optimal brain surgeon". In: *Advances in neural information processing systems*. 1993, pp. 164–171.

[15] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[16] Yang He et al. "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.

[17] Yang He et al. "Soft filter pruning for accelerating deep convolutional neural networks". In: *arXiv preprint arXiv:1808.06866* (2018).

[18] Yihui He, Xiangyu Zhang, and Jian Sun. "Channel pruning for accelerating very deep neural networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1389–1397.

[19] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[20] Hengyuan Hu et al. "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures". In: *arXiv preprint arXiv:1607.03250* (2016).

[21] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[22] Qiangui Huang et al. "Learning to prune filters in convolutional neural networks". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 709–718.

[23] Zehao Huang and Naiyan Wang. "Data-driven sparse structure selection for deep neural networks". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 304–320.

[24] Roxana Istrate et al. "Incremental training of deep convolutional neural networks". In: *arXiv preprint arXiv:1803.10232* (2018).

[25] Junichi Miyao Kakeru Mitsuno and Takio Kurita. "Hierarchical Group Sparse Regularization for Deep Convolutional Neural Networks". In: *Proceedings of the international joint conference on neural networks*. 2020.

[26] Deguang Kong et al. "Exclusive Feature Learning on Arbitrary Structures via $\ell_{1,2}$-norm". In: *Advances in Neural Information Processing Systems*. 2014, pp. 1655–1663.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[28] Vadim Lebedev and Victor Lempitsky. "Fast convnets using group-wise brain damage". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2554–2564.

[29] Yann LeCun, John S Denker, and Sara A Solla. "Optimal brain damage". In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[30] Feng Li, Jacek M Zurada, and Wei Wu. "Smooth group $L_{1/2}$ regularization for input layer of feedforward neural networks". In: *Neurocomputing* 314 (2018), pp. 109–119.

[31] Hao Li et al. "Pruning filters for efficient convnets". In: *arXiv preprint arXiv:1608.08710* (2016).

[32] Jiashi Li et al. "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.

[33] Ji Lin et al. "Runtime neural pruning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 2181–2191.

[34] Shaohui Lin et al. "Accelerating Convolutional Networks via Global & Dynamic Filter Pruning." In: *IJCAI*. 2018, pp. 2425–2432.

[35] Shaohui Lin et al. "Towards optimal structured cnn pruning via generative adversarial learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2790–2799.

[36] Hanxiao Liu, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search". In: *arXiv preprint arXiv:1806.09055* (2018).

[37] Zhuang Liu et al. "Learning efficient convolutional networks through network slimming". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2736–2744.

[38] Zhuang Liu et al. "Rethinking the Value of Network Pruning". In: *International Conference on Learning Representations*. 2019.

[39] Christos Louizos, Max Welling, and Diederik P Kingma. "Learning Sparse Neural Networks through L_0 Regularization". In: *arXiv preprint arXiv:1712.01312* (2017).

[40] Jian-Hao Luo and Jianxin Wu. "An entropy-based pruning method for cnn compression". In: *arXiv preprint arXiv:1706.05791* (2017).

[41] Jian-Hao Luo, Jianxin Wu, and Weiyao 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.

[42] Masakazu Matsugu et al. “Subject independent facial expression recognition with robust face detection using a convolutional neural network”. In: *Neural Networks* 16.5-6 (2003), pp. 555–559.

[43] Kakeru Mitsuno and Takio Kurita. “Filter Pruning using Hierarchical Group Sparse Regularization for Deep Convolutional Neural Networks”. submitted. 2020.

[44] Deepak Mittal et al. “Recovering from random pruning: On the plasticity of deep convolutional neural networks”. In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 848–857.

[45] Pavlo Molchanov et al. “Importance estimation for neural network pruning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11264–11272.

[46] Pavlo Molchanov et al. “Pruning convolutional neural networks for resource efficient inference”. In: *arXiv preprint arXiv:1611.06440* (2016).

[47] Ari Morcos et al. “One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 4933–4943.

[48] Hanyu Peng et al. “Collaborative channel pruning for deep networks”. In: *International Conference on Machine Learning*. 2019, pp. 5113–5122.

[49] Hieu Pham et al. “Efficient neural architecture search via parameter sharing”. In: *arXiv preprint arXiv:1802.03268* (2018).

[50] Esteban Real et al. “Regularized evolution for image classifier architecture search”. In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 33. 2019, pp. 4780–4789.

[51] Adriana Romero et al. “Fitnets: Hints for thin deep nets”. In: *arXiv preprint arXiv:1412.6550* (2014).

[52] Simone Scardapane et al. “Group sparse regularization for deep neural networks”. In: *Neurocomputing* 241 (2017), pp. 81–89.

[53] Mark Schmidt. “Graphical model structure learning with l1-regularization”. In: *University of British Columbia* (2010).

[54] Noah Simon et al. “A sparse-group lasso”. In: *Journal of computational and graphical statistics* 22.2 (2013), pp. 231–245.

[55] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[56] Suraj Srinivas and R Venkatesh Babu. “Data-free parameter pruning for deep neural networks”. In: *arXiv preprint arXiv:1507.06149* (2015).

[57] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[58] Hokchhay Tann et al. "Runtime configurable deep neural networks for energy-accuracy trade-off". In: *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE. 2016, pp. 1–10.

[59] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[60] Xin Wang et al. "Skipnet: Learning dynamic routing in convolutional networks". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 409–424.

[61] Wei Wen et al. "Learning structured sparsity in deep neural networks". In: *Advances in neural information processing systems*. 2016, pp. 2074–2082.

[62] Bichen Wu et al. "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10734–10742.

[63] Wei Wu et al. "Batch gradient method with smoothing $L_{1/2}$ regularization for training of feedforward neural networks". In: *Neural Networks* 50 (2014), pp. 72–78.

[64] Vinay Palakkode Xavier Suau Luca Zappella and Nicholas Apostoloff. "Principal Filter Analysis for Guided Network Compression". In: *CoRR* abs/1807.10585 (2018).

[65] Zongben Xu et al. "$L_{1/2}$ regularization". In: *Science China Information Sciences* 53.6 (2010), pp. 1159–1169.

[66] Zongben Xu et al. "$L_{1/2}$ regularization: A thresholding representation theory and a fast solver". In: *IEEE Transactions on neural networks and learning systems* 23.7 (2012), pp. 1013–1027.

[67] Lei Xun et al. "Incremental training and group convolution pruning for runtime dnn performance scaling on heterogeneous embedded platforms". In: *ACM/IEEE Workshop on Machine Learning for CAD 2019 (MLCAD'19)*. 2020, pp. 1–6.

[68] Jianbo Ye et al. "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers". In: *arXiv preprint arXiv:1802.00124* (2018).

[69] Jaehong Yoon and Sung Ju Hwang. "Combined group and exclusive sparsity for deep neural networks". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 3958–3966.

[70] Jiahui Yu et al. "Slimmable Neural Networks". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=H1gMCsAqY7.

[71] Ruichi Yu et al. "Nisp: Pruning networks using neuron importance score propagation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9194–9203.

[72] Ming Yuan and Yi Lin. "Model selection and estimation in regression with grouped variables". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67.

[73] Jinshan Zeng et al. "$L_{1/2}$ regularization: Convergence of iterative half thresholding algorithm". In: *IEEE Transactions on Signal Processing* 62.9 (2014), pp. 2317–2329.

[74] Ying Zhang et al. "Deep mutual learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4320–4328.

[75] Hao Zhou, Jose M Alvarez, and Fatih Porikli. "Less is more: Towards compact cnns". In: *European Conference on Computer Vision*. Springer. 2016, pp. 662–677.

[76] Hattie Zhou et al. "Deconstructing lottery tickets: Zeros, signs, and the supermask". In: *Advances in Neural Information Processing Systems*. 2019, pp. 3592–3602.

[77] Yang Zhou, Rong Jin, and Steven Chu-Hong Hoi. "Exclusive lasso for multi-task feature selection". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 988–995.

[78] Michael Zhu and Suyog Gupta. "To prune, or not to prune: exploring the efficacy of pruning for model compression". In: *arXiv preprint arXiv:1710.01878* (2017).

[79] Barret Zoph and Quoc V Le. "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578* (2016).

[80] Barret Zoph et al. "Learning transferable architectures for scalable image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710.