New Efficient Pruning Algorithms for Compressing and Accelerating Convolutional Neural Networks.

by

Shangqian Gao

M.S., Northeastern University, 2017

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2024

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINNERING

This dissertation was presented

by

Shangqian Gao

It was presented on

November 29th 2023

and approved by

Wei Gao, PhD, Associate Professor, Department of Electrical and Computer Engineering

Wei Chen, PhD, Professor, Department of Pediatrics, School of Medicine

Liang Zhan, PhD, Associate Professor, Department of Electrical and Computer Engineering

Zhi-Hong Mao, PhD, Professor, Department of Electrical and Computer Engineering

Dissertation Director: Heng Huang, PhD, Brendan Iribe Endowed Professor, Department of Computer Science, University of Maryland College Park

New Efficient Pruning Algorithms for Compressing and Accelerating Convolutional Neural Networks.

Shangqian Gao

University of Pittsburgh, 2024

Recently, Convolutional Neural Networks (CNNs) are continuously achieving state-of-the-art results in numerous machine-learning tasks. While having impressive performance, the size of current models is also exploding. Motivated by efficient inference, many researchers have been devoted to reducing the storage and computational costs of state-of-the-art models. Channel pruning emerges as a promising solution to reduce the size of the model, and it can achieve acceleration without any post-processing steps. Current channel pruning methods are either time-consuming (reinforcement learning, greedy search, etc.) or depend on fixed criteria of channels resulting in poor results.

In this dissertation work, we propose new methods from the perspective of gradient-guided pruning. We then formulate the pruning problem as a constrained discrete optimization problem. Our discrete model compression work aims to solve this constrained problem by using differentiable gates and propagating gradients through a straight-through estimator. We further improve the results in network pruning via performance maximization by adding a performance prediction loss into the constrained optimization problem. The search for sub-networks is then directly guided by the accuracy of a sub-network. The improvement of supervision leads to better pruning results.

On top of previous works, we propose to further improve our algorithms from different perspectives. The first perspective is to disentangle width and importance for finding the optimal model architecture. From this end, we propose to use an importance generation network and a width generation network to generate the importance and width for each layer. Another challenge in previous works is the huge gap between the model before and after network pruning. To mitigate this gap, we first learn a target sub-network during the model training process, and then we use this sub-network to guide the learning of model weights through partial regularization. Based on the success of previous static pruning methods, we further incorporate dynamic pruning for storage-efficient dynamic pruning.

Table of Contents

List of Tables

List of Figures

Preface

I want to express my sincere thanks to Dr. Heng Huang for his consistent support and guidance as my advisor. His expertise and mentorship have been crucial in shaping and conducting my research. I deeply appreciate the opportunities for publication, travel assistance, and research support he has offered me throughout this journey.

I would also like to express my sincere appreciation to Professor Wei Chen, Professor Wei Gao, Professor Zhihong Mao, and Professor Liang Zhan for their generously serving on my PhD dissertation committee. Their valuable suggestions and feedback on my dissertation study and future research plans have been essential to its progress. I truly appreciate the time and guidance they've given me to help further my academic journey.

I am also indebted to every collaborator who has contributed to my PhD journey. I would like to extend a special thanks to Dr. Liang Zhan. Additionally, I want to express my gratitude to Professor Feihu Huang, Professor Lei Luo, Professor Feng Zheng, Professor Xiaoqian Wang, Professor Hongchang Gao, Dr. Zhouyuan Huo, Guodong Liu, Dr. Yanfu Zhang, Wenhan Xian, Dr. Runxue Bao, Dr. An Xu, Dr. Haoteng Tang, Dr. Xiaotian Dou, Junyi Li, Zhengmian Hu, Xidong Wu and Lichang Chen for their collaboration and support. Working with such talented individuals has been an honor, and I value the chance to learn from them and the friendships we've developed.

Finally, I want to express my sincere gratitude to my spouse, Fangyi Li, for her constant love, support, and understanding during the challenges of my PhD journey. Also, I want to express my heartfelt appreciation to my parents for their unwavering love and endless encouragement throughout my academic journey. Their belief in me has been my greatest motivation. I'm forever grateful for their dedication, without which this milestone wouldn't have been achievable.

1.0 Introduction

Convolutional Neural Networks (CNNs) have accomplished great success in many machine learning and computer vision tasks [83, 122, 123, 129, 3]. To deal with real-world applications, recently, the design of CNNs has become more and more complicated in terms of width, depth, etc. [83, 130, 46, 69]. Although these complex CNNs can attain better performance on benchmark tasks, their computational and storage costs increase dramatically. As a result, a typical application based on CNNs can easily exhaust an embedded or mobile device due to its enormous costs. Given such costs, the application can hardly be deployed on resource-limited platforms. To tackle these problems, many methods [43, 42] have been devoted to compressing the original large CNNs into compact models. Among these methods, weight pruning and structural pruning are two popular directions.

Unlike weight pruning or sparsification, structural pruning, especially channel pruning, is an effective way to truncate the computational cost of a model because it does not require any post-processing steps to achieve actual acceleration and compression. Many existing works [112, 88] try to solve this problem by using certain criteria of channels. Although these criteria like L_2 norm or importance score are not hard to calculate, they often can not achieve satisfactory results. One of the reasons is that the criterion of each channel is considered separately, and connections between channels are omitted. Another line of research approaches this problem by using reinforcement learning [50] or evolutionary strategies [102]. These methods often achieve better results but with significantly higher costs.

To obtain efficiency and high performance at the same time, we proposed new methods from the perspective of gradient-guided pruning. We formulate the pruning problem as a constrained discrete optimization problem. Our work discrete model compression [31] aims to solve this constrained problem by using differentiable gates and propagating gradients through a straight-through estimator. We further improve the results in network pruning via performance maximization [30] by adding a performance prediction loss into the constrained optimization problem. The search for sub-networks is then directly guided by the accuracy of a sub-network. The improvement of supervision leads to better pruning results. Compared to existing works, the efficiency is largely

improved due to fast gradient-based search.

On top of previous works, we propose to improve our algorithms from different perspectives. The first perspective is to disentangle the learning of model width and channel importance [33]. From this end, we propose to use two different models to generate the importance and the number of channels for each layer. The previous one-shot pruning method creates a large gap between the model before and after pruning. To tackle this problem, we introduce a novel partial regularization technique [36] to align model weights and the discovered sub-network during the training process, which can produce a high performance sub-network and reduce the gap between the sub-network and the original model. In addition, unlike soft-pruning methods, all model structures are used for training. The partial regularization term contains a partial group lasso regularization on selected weights, and other weights remain intact without modifications. An architecture generator is trained to select which weights should be aligned, and it is also updated during the training process. Our partial regularization formulation is related to partial regularization in lasso [106]. Inspired by the nonmonotone proximal gradient (NPG) method used in [106], we also use a proximal gradient method to solve the partial regularization problem in our setting. Standing on the success of our previous static pruning methods, we further incorporate dynamic pruning into consideration and we propose a novel channel pruning method, which unifies both dynamic and static pruning. Dynamic and static sub-networks are connected by evaluating the static sub-network through dynamic sub-networks instead of training them in parallel.

The organization of this dissertation is as follows: we will first introduce the basic framework of our work: "Discrete Model Compression", then we introduce how to incorporate additional supervision for pruning: "Network Pruning via Performance Maximization" and "Disentangled Differentiable Network Pruning" in Chapter 3 and Chapter 4. We further discuss "Structural Alignment for Network Pruning through Partial Regularization" and "BilevelPruning: Unified Dynamic and Static Channel Pruning for Convolutional Neural Networks" in Chapter 5 and Chapter 6.

2.0 Discrete Model Compression

2.1 Background

Convolutional Neural Networks (CNNs) have achieved great success in computer vision tasks [83, 122, 123, 129, 3]. With more and more sophisticated GPU support on CNNs, the complexity of CNN grows dramatically from several layers [83, 130] to hundreds of layers [46, 69]. Although these complex CNNs can achieve strong performance on vision tasks, there is an unavoidable growth of the computational cost and model parameters. Such a huge computational burden prohibits the model from being deployed on mobile devices and resource-limited platforms. Even if the model can be deployed on mobile devices, the battery will be depleted quickly due to huge computational costs. To tackle such problems, many efforts [43, 42] have been devoted to getting compact sub-networks from the original computational heavy model.

Structural pruning, especially channel pruning, is an efficient way to reduce computational cost since it doesn't require any post-processing steps to acquire acceleration. One of the most challenging parts of structural pruning is how to deal with the natural discrete configuration of channels in each layer. Many existing works [107, 119] try to solve this problem by relaxing discrete values to continuous values. However, such relaxation may lead to a biased estimation of the corresponding pruning criterion, since you can't completely remove the impact of channels with small importance value. Some other methods [103, 112] use the estimation of channel importance to decide whether to prune a channel, nonetheless, the relative importance of a channel can be changed due to the choice of the sub-network. Recently, discrimination-aware pruning [154] has been proposed to explore the impact of discriminative power on channel pruning. Although this method considers the discriminative power of CNNs by adding classifiers on intermediate layers, it does not consider CNN as a whole, which may result in sub-optimal compression results.

To deal with these challenges, we propose a new method of using the discrete gate to turn off or turn on certain channels. By doing so, we can always get the exact model output given different sub-network architectures. Thanks to the precise output estimation of a sub-network, our method is able to consider the discriminative power of a complete sub-network. At the same time,

we do not take the magnitude of a channel into consideration, the global discriminative power is the only criterion. Moreover, we propose an efficient optimization method to obtain the sub-network. Although directly optimizing discrete variables is often non-smooth non-convex and NP-hard, our optimization method can circumvent these difficulties by using the straight-through estimator (STE) [2].

Automatic Model Compression (AMC) [50] is a pioneering method using the discrete channel setting, which is optimized by reinforced learning [65, 63]. Different from AMC, our method is guided by gradients of the loss function when exploring sub-networks from the original CNN. Our method can obtain a sub-network efficiently due to its differentiable nature. On the ImageNet dataset, our method can discover a high performance sub-network satisfying a given budget within 2% of time for regular training (finetuning time is excluded). From this perspective, our method is also related to differentiable architecture search (DARTS) [99].

Our main contributions are summarized as follows:

1) To compress a model, we apply the discrete gate on each channel, which ensures that the output from any sub-networks is correct and unbiased.
2) We use STE to enable back-propagation through discrete variables. To further enlarge the search space of sub-networks, we replace the discrete gate with its stochastic version.
3) To ensure we can get the model with the given computational budget, we further propose resource regularization when exploring potential sub-networks.
4) Extensive experimental results show that our method achieves state-of-the-art results on CIFAR10 and ImageNet with ResNet and MobileNetV2.

2.2 Related Works

Model compression recently has drawn a lot of attention from the compute vision community. In general, current model compression methods can be separated into the following four categories: weight pruning, structural pruning, weight quantization, and knowledge distillation [52].

Weight pruning eliminates model parameters without any assumption on the structure of weights. One of the early works [43] uses L_1 or L_2 magnitude as criterion to remove weights. Under this

setting, parameters lower than a certain threshold are removed, and weights with small magnitude are considered not important. A systematic DNN weight pruning framework [149] has been proposed by using alternating direction method of multipliers (ADMM) [5, 59, 64]. Different from the aforementioned works, SNIP [85] updates the importance of weights by backpropagating gradients from the loss function. Lottery ticket hypothesis [22] is another very interesting weight pruning algorithm, which manifests that small high-performance sub-networks exist within the overparameterized large network at initialization time. Different from the lottery ticket hypothesis, in rethinking the value of network pruning [104], they argue that fine-tuning a pre-trained model is not necessary and show that the pruned model with random initialization achieves better performance. he major drawback of weight level pruning is that they often require a specially designed sparse matrix multiplication library to achieve acceleration.

Different from weight pruning, structural pruning provides a natural way to reduce computational costs. The development of structural pruning is similar to weight pruning in that channels with low magnitude are often regarded as not important [88]. Similar to the idea of magnitude pruning, Group Lasso [140] is also applied on CNNs to structurally make channels or filters to have all 0 values, thus those channels can be safely removed. GrOWL [148] focuses on exploring inter-channel relations on top of sparsity and argues that similar channels can share the same weights. The following researches show that weights with small magnitude could be important [105], and it's difficult for channels under L_1 regularization to achieve exact zero values. To compensate for this, they propose to get exact zero values for each channel by using explicit L_0 regularization [105]. Besides simply using channel magnitude as pruning standards, other methods utilize batchnorm to achieve a similar target, since batchnorm [71] is an indispensable component in recent neural network designs [46, 69]. For each channel, batchnorm uses a scaling factor γ to adjust the magnitude of corresponding feature maps. To achieve the goal of channel pruning, γ is regularized to be sparse and γ fell below a predefined threshold will be set to 0 during model pruning [103]. Other works related to this idea include [143, 70, 77]. Unlike previous methods relying on the magnitude of channels, discrimination-aware pruning [154] utilizes local discriminative power to help channel pruning. AMC [50] achieves the goal of channel pruning in a discrete setting by taking advantage of reinforcement learning. Collaborative channel pruning [119] focuses on pruning channels by utilizing Taylor expansion of the loss function. Our method belongs

to this category, the major difference between our method and previous model pruning methods is that our method strictly uses the discrete setting of channels while it can be optimized through gradient descent.

Weight quantization is another direction for model compression, which focuses on reducing the numerical precision of weights from 32-bit float point value to low bit value. Binary connects [14] and binary neural network [121] push the full precision weight to binary weight values, making the model weights become binary values. The connection between our method and weight quantization is that both of them use STE to estimate the gradient for discrete values.

Besides the aforementioned methods, there are works from other directions. There is a range of methods that focus on pruning weights [111] or structures [115] by utilizing uncertainty in weights. EigenDamge [134] can achieve structural pruning by using a Kronecker-factored eigenbasis. Singular value decomposition [56] is popular for compressing language models. Token merging and pruning [78] are widely used for transformers.

2.3 Proposed Method

2.3.1 Notations

To better describe the proposed approach, we first define some notations. The feature map of each layer can be represented by $\mathcal{F}_l \in \Re^{C_l \times W_l \times H_l}$ where C_l is the number of channels, W_l and H_l are height and width of the current feature map. $\mathcal{F}_{l,c}$ is the feature map of c-th channel from l-th layer. The mini-batch dimension of the feature map is ignored to simplify notations. Throughout the paper, *w.p.* means with probability. $\mathbf{1} = [1, \ldots, 1]^T$ is a vector with all ones. $\text{sign}(\cdot)$ is the popular sign function.

2.3.2 Differentiable Discrete Gate

In this paper, to incorporate the discrete nature of channel pruning, we explicitly consider using discrete-valued gates to represent the opening or closing of a channel. The discrete gate function

can be described as follows:

$$g(\theta) = \begin{cases} 1 & \text{if } \theta \in [0.5, 1], \\ 0 & \text{if } \theta \in [0, 0.5), \end{cases} \tag{2-1}$$

where $\theta \in [0, 1]$ is a learnable parameter in our setting. The discrete gate function is applied after the output feature map of a layer:

$$\mathcal{F}\widehat{\mathcal{F}}_{l,c} = g(\theta_{l,c}) \cdot \mathcal{F}_{l,c}, \tag{2-2}$$

where $\mathcal{F}\widehat{\mathcal{F}}_{l,c}$ is the feature map after pruning.

Since the binary gate function is not differentiable, STE [2] is used to enable gradient calculation, which can be described as follows:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial g(\theta)}, \tag{2-3}$$

where $\mathcal{L}$ is the loss function. Here, the backward propagation of $g(\theta)$ can be understood as an identity function within a certain range. If $\theta \notin [0, 1]$, the gradient will not be calculated, and the θ will be clipped to range $[0, 1]$.

In fact, there are some limitations of the deterministic discrete gate function. For example, once the $g(\theta)$ of certain channels become 0, then these channels probably remain pruned, and they may never be selected as candidate channels. To compensate for such situations, we further propose the stochastic discrete gate to ensure gates with θ lower than 0.5 can be considered as candidate channels again. Specifically, the stochastic discrete gate is achieved by applying stochastic rounding:

$$g(\theta) = \begin{cases} 1 & \textit{w.p. } \theta \\ 0 & \textit{w.p. } 1 - \theta \end{cases} \tag{2-4}$$

where θ is within $[0, 1]$ to satisfy the definition of probability. From this definition, we can see that a channel always has a chance to be sampled if $\theta \neq 0$. Since our method uses the discrete setting, sampling from the stochastic discrete gate is equivalent to sampling a sub-network.

2.4 Model Compression as Constrained Optimization

There are many different ways to represent the pruning objective for model compression. In this paper, we mainly focus on the following channel pruning problem:

$$\begin{aligned} &\min_{\Theta} \mathcal{L}(f(x; \mathcal{W}, \Theta), y) \\ &s.t.\ \mathbf{1}^T\mathbf{g} - p\mathbf{1}^T\mathbf{C} = 0 \\ &\mathbf{g} \in \{0, 1\}^n, \end{aligned} \tag{2-5}$$

where $\mathbf{g} = (g_1, \ldots, g_L)$ is a vector containing all gate values, $\mathbf{g}_l = [g(\theta_{l,1}), \ldots, g(\theta_{l,C_l})]^T$ is the vector containing gate values in l-th layer, Θ are the parameters of discrete gates following the definition in section 2.3.2, $\mathcal{W}$ is the model parameters, p is a predefined threshold working as the pruning rate, and $\mathbf{C} = (C_1, \cdots, C_L)$. Here, $\mathbf{1}^T\mathbf{g}$ is the sum of all gate values, which represents the number of remaining channels, n is the total number of gates, and $\mathbf{1}^T\mathbf{C}$ is the total number of channels. Note that not all layers are included in Eq. (6-5), and the vector $\mathbf{g}$ and $\mathbf{C}$ only contain layers that are used for pruning. There are several remarks on this pruning objective: **1)** The pruning of channels only depends on the discriminate power of its own, channel magnitude is irrelevant during model pruning; **2)** There exists no layer-wise hyper-parameter, only a global hyper-parameter is used to control pruning rate; **3)** During training of the parameters for the gate function, model parameters $\mathcal{W}$ are fixed.

Under this setting, the major advantage of the discrete gate setting is that the impact of pruned channels is precisely reflected in the value of the loss function. If the gates are relaxed in continuous values, then it doesn't possess such good property. In addition, the continuous relaxed gate function causes severe difficulty in solving the optimization problem defined in Eq. (6-5).

For simplicity, we replace the equality constraint with a regularization term and redefine the optimization problem as follows:

$$\min_{\Theta} \mathcal{F}(\Theta) := \mathcal{L}(f(x; \mathcal{W}, \Theta), y) + \lambda\mathcal{R}(\mathbf{1}^T\mathbf{g},\ p\mathbf{1}^T\mathbf{C}), \tag{2-6}$$

where $\mathcal{R}(\cdot, \cdot)$ is the specific regularization function used, which can be a typical regression loss function such as MAE or MSE. In practice, both MAE and MSE are not used, we will talk about

the choice of $\mathcal{R}(\cdot,\cdot)$ later. The constraint $\mathbf{g} \in \{0,1\}^n$ is absorbed into the optimization problem due to the definition of the discrete gate (Eq. (6-1) and (6-4)).

Recently, there has been increasing interest in reducing the float point operations (FLOPs) in model pruning literature. The definition used in Eq. (6-6) along with the definition of the deterministic discrete gate in Eq. (6-1) can easily transform the pruning rate constraint in Eq. (6-5) to FLOPs constraint. Recall that for a single convolution layer l with one sample, the FLOPs calculation can be down as follows:

$$(\text{FLOPs})_l = k_l \cdot k_l \cdot \frac{c_{l-1}}{\mathcal{G}_l} \cdot c_l \cdot w_l \cdot h_l, \tag{2-7}$$

where k_l is the kernel size, $\mathcal{G}_l$ is the number of groups, c_{l-1} and c_l are the number of input and output channels, w_l and h_l are width and height, $(\text{FLOPs})_l$ is the FLOPs of l-th layer. By replacing c_l and c_{l-1} with $\mathbf{g}_l$ and $\mathbf{g}_{l-1}$, we get a new representation of FLOPs while searching for sub-networks:

$$(\text{FLOPs})(\widehat{\text{FLOPs}})_l = k_l \cdot k_l \cdot \frac{\mathbf{1}^T\mathbf{g}_{l-1}}{\mathcal{G}_l} \cdot \mathbf{1}^T\mathbf{g}_l \cdot w_l \cdot h_l, \tag{2-8}$$

Then combining Eq. (5-7), Eq. (6-8) with Eq. (6-6), the original $\mathcal{R}(\mathbf{1}^T\mathbf{g},\ p\mathbf{1}^T\mathbf{C})$ is replaced by the FLOPs regularzation:

$$\mathcal{R}(\hat{T},\ pT), \tag{2-9}$$

where $\hat{T} = \sum_{l=1}^{L} (\text{FLOPs})(\widehat{\text{FLOPs}})_l$ and $T = \sum_{l=1}^{L} (\text{FLOPs})_l$ are remained FLOPs and total FLOPs of the model. Note that we still use p as the only global hyper-parameter to represent the remaining fractions of the FLOPs. By incorporating FLOPs regularization and the objective in Eq. (6-6), we can prune the model to arbitrary levels of FLOPs.

2.4.1 Choice of the Regularization Loss

In order to train the parameters of gates properly, the value of the regularization term should be decreased to near 0 in the early stage of gate training, and remain near 0 for the rest of the training process. The reason we want to keep the regularization term near 0 for most time is to ensure the algorithm have enough time to discover the best possible sub-architecture with the given constraint. Regular regression loss like MAE and MSE can't satisfy this requirement, since their

gradient is either constant or decreasing when close to 0. To overcome this issue, we propose the following regularization loss:

$$\mathcal{R}_{\log}(x, y) = \log(|x - y|+1). \tag{2-10}$$

The plot of gradient and value of this function is shown in Fig. 2. The benefit of this function is that when x is close to target y, the gradient will increase and keep x close to the target value y. The loss function is not differentiable at the point $x = y$ by definition, but sub-gradient can be used here which has been implemented in major deep learning frameworks. In practice, the Eq. (6-10) works well for the appropriate choice of λ, on the contrary, MAE and MSE often fail to keep the value of the regularization term close to 0.

2.4.2 Symmetric Weight Decay

To further expand search space, we propose a symmetric weight decay on the weights of gates, which is inspired by the subgradient of the regularization loss:

$$\frac{\partial \mathcal{R}_{\log}}{\partial \theta_{l,c}} = \begin{cases} \eta_l \cdot \frac{1}{|\hat{T}-pT|+1} \cdot \frac{\hat{T}-pT}{|\hat{T}-pT|}, & \text{if } \hat{T} \neq pT \\ 0, & \text{if } \hat{T} = pT \end{cases} \tag{2-11}$$

where $\eta_l = k_l^2 \cdot \frac{\mathbf{1}^T \mathbf{g}_{l-1}}{\mathcal{G}_l} \cdot w_l \cdot h_l$. Eq. (A.1) indicates that $\mathcal{R}_{\log}$ works like weight decay with different decay values for each layer while the decay value is also different for each training iteration. Since the impact of $R_{\log}$ on θ can be expressed by weight decay, the stochasticity of the gates can be increased in a similar way.

Based on the above arguments, we can explore larger search spaces by applying symmetric weight decay on each $\theta_{l,c}$. The goal of doing this is to slow down the pace of gate parameters to become deterministic (approach 0 or 1). As a result, the search space is enlarged:

$$\theta_{l,c} = \theta_{l,c} - \beta \text{sign}(\theta_{l,c} - 0.5), \tag{2-12}$$

where β is the hyper-parameter to control the strength of weight decay.

2.5 Our DMC Algorithm

We have introduced the core idea of our method, and the discrete model compression (DMC) algorithm is presented in Algorithm 1. During freezing trainable parameters, the batchnorm running statistics are also frozen. It should be emphasized again that the gate learning process is isolated from the training of the model parameters. In this way, we can prune any pre-trained models without modifications. In the calculation of $\mathcal{L}$, the sub-networks are drawn from the stochastic discrete gate. When calculating FLOPs regularization and during model pruning, the deterministic version of the gate is used. Both stochastic and deterministic calculations share the same $\theta_{l,c}$. Our method also has the potential to be applied to cross-modal tasks [81, 29, 87, 86] and graph neural networks [152].

2.6 Experiments

2.6.1 Settings

2.6.1.1 Implementation Detail

We use CIFAR-10 [82] and ImageNet [15] to verify the performance of our method. Our method only requires one hyperparameter p to control the FLOPs budget. For all experiments, we use resource regularization with $\mathcal{R}_{\log}$ defined in Eq. (6-10). As a result, p decides how much FLOPs are preserved for each experiment. λ decides the regularization strength in our method. We choose $\lambda = 4$ in all CIFAR-10 experiments and $\lambda = 8$ for all ImageNet experiments. For CIFAR-10, we compare with other methods on ResNet-56 and MobileNetV2. For ImageNet, we select ResNet-34, ResNet50, ResNet101 and MobileNetV2 as our target models. The reason we choose these models is because that ResNet [46] models and MobileNetV2 [126] are much harder to prune than earlier models like AlexNet [83] and VGG [130]. For CIFAR-10 models, we train it from scratch following the code from PyTorch examples. After pruning the model, we finetune the model for 160 epochs using SGD with start learning rate 0.1, weight decay 0.0001 and momentum 0.8, the learning rate is multiplied by 0.1 at epoch 80 and 120. For ImageNet models, we directly

use the pre-trained models released from pytorch [117]. After pruning, we finetune the model for 100 epochs using SGD with start learning rate 0.01, weight decay 0.0001 and momentum 0.9, and the learning rate is scaled by 0.1 at epoch 30, 60 and 90. For MobileNetV2 on ImageNet, we choose weight decay as 0.00004 which is usd in the original paper [126]. Both CIFAR-10 and ImageNet finetuning hyperparameters are similar to those used in Collaborative Channel Pruning (CCP) [119]. During gate training, we choose the β of symmetric weight decay (Eq. (2-12)) as 0.0001. We randomly choose $2,500$ and $10,000$ samples as the dataset for training gate (D_{gate}) for CIFAR-10 and ImageNet separately. We didn't create a standalone validation set for training gate in order to directly use pre-trained models. In the gate training process, we use ADAM [79] optimizer with a constant learning rate 0.001 and train gate parameters for 300 epochs. All the codes are implemented with pytorch [117]. The experiments are conducted on a machine with 4 Nvidia Tesla P40 GPUs.

2.6.1.2 Placement of Gate

. Where to put gates is a crucial problem to best approximate the output from actual sub-networks when corresponding channels are pruned. Following the settings in NAS works [99, 155], we regard Conv-BN-Relu as a complete block, thus the gates are always placed after Relu activation functions. To better simulate the results of a sub-network, we place two individual gates for a bottleneck block in a ResNet. For MobileNetV2, we place two gates in an inverted residual block, and the two gates share the same set of parameters due to the nature of depth-wise convolution. Details are shown in Fig. 3. Following the above settings, outputs from sampled sub-networks can well approximate the outputs from the actual compact network.

Table 1: Comparison results on CIFAR-10 dataset with ResNet-56 and MobileNetV2. Δ-Acc represents the performance changes before and after model pruning. +/- indicates increase or decrease compared to baseline results. WM represents width multiplier used in the original design of MobileNetV2, this result is from DCP [154] paper.

Method	Architecture	Baseline Acc	Pruned Acc	Δ-Acc	Pruned FLOPs
Channel Pruning [51]	ResNet-56	92.80%	91.80%	-1.00%	50.0%
AMC [50]		92.80%	91.90%	-0.90%	50.0%
Pruning Filters [88]		93.04%	93.06%	+0.02%	27.6%
Soft Prunings [48]		93.59%	93.35%	-0.24%	52.6%
DCP [154]		93.80%	93.59%	-0.31%	50.0%
DCP-Adapt [154]		93.80%	93.81%	+0.01%	47.0%
CCP [119]		93.50%	93.42%	-0.08%	**52.6%**
DMC(ours)		93.62%	93.69%	**+0.07%**	50.0%
WM* [154]	MobileNetV2	94.47%	94.17%	-0.30%	26.0%
DCP [154]		94.47%	94.69%	+0.22%	26.0%
DMC(ours)		94.23%	94.49%	**+0.26%**	**40.0%**

2.6.2 Results on CIFAR-10

In Tab. 1, we show all the comparison results on CIFAR-10. For ResNet-56, our method performs much better than early methods [51, 50, 88, 48]. Specially, when compared with Soft Pruning, our method is better than their results by 0.31% on Δ-Acc given similar pruned FLOPs (52.6% vs 50%). Discrimination-aware pruning utilizes local discrimination criteria when pruning the model. Our method outperforms DCP [154] by 0.38% on Δ-Acc with the same pruned FLOPs. Moreover, our method outperforms DCP-adapt (a stronger version of DCP) by 0.06% give similar pruned FLOPs. Collaborative filter pruning [119] is one of the most recent works on channel pruning which considers the correlation between different weights when applying Taylor expansion on the loss function. Our method still outperforms their result by 0.15% on Δ-Acc. Such observation may indicate that our method also implicitly considers weights correlation during the search for

optimal sub-network. For MobileNetV2, our method outperforms DCP by 0.04% on Δ-Acc, while pruning 14% more FLOPs than DCP. This shows that global discrimination-aware is better than local discrimination-aware.

2.6.3 Results on ImageNet

In Tab. 23, we present all the comparison results on ImageNet. All results are adopted from their original papers except for ThiNet on MobileNetV2. To establish a high-quality baseline, the comparison methods are mainly chosen from recently published papers. Specially, DCP [154], CCP [119], IE [112], FPGM [49] and GAL [98] are from this category. Such high-quality baselines can help us better understand the benefit of using discrete channel settings.

For ResNet-34, our method can prune 43.4% FLOPs while only resulting in 0.73% and 0.31% performance drops on Top-1 accuracy and Top-5 accuracy separately. FPGM prunes slightly less FLOPs compared with our method (41.1% vs 43.3%), however, it causes larger damage to the final performance than our method (0.56% worse with Top-1 accuracy and 0.23% worse with Top-5 accuracy). The other two methods only prune a small amount of FLOPs (around 25%). Our method has a lower Top-1 accuracy compared with IE, but we prune 1.8 times as much FLOPs as IE. For ResNet-50, our method achieves the best Δ-Acc Top-1 and Δ-Acc Top-5 accuracy compared with all other methods. Among all comparison methods, CCP has the smallest performance gap with our method. Specifically, our DMC algorithm outperforms the state-of-the-art pruning algorithm CCP by 0.14% at Top-1 accuracy with slightly more pruned FLOPs (55.0% vs 54.1%). For other comparison methods, our DMC algorithm has advantages on Top-1 accuracy varying from 0.26% to 3.55%.

For ResNet-101, our method also achieves the best Δ-Acc Top-1 and Δ-Acc Top-5 accuracy. Moreover, our DMC algorithm prunes a much larger amount of FLOPs than all the other methods (56% vs 47% the second largest). After pruning 56% of FLOPs, the pruned ResNet-101 only has 3.43 GFLOPs which is even less than the vanilla ResNet-50 (4.09 GFLOPs). At such a high pruning rate for FLOPs, the performance of pruned ResNet-101 even increases by 0.03% and 0.04% for Top-1 and Top-5 accuracy respectively. For MobileNet-V2, our method largely increases the results obtained by DCP. With similar amount of pruned FLOPs (46% vs 44.7%), DMC outper-

forms DCP by 2.38% and 1.94% for Δ-Acc Top-1 and Δ-Acc Top-5 accuracy. Such significant improvement further shows that the global discrimination-aware pruning utilized in our method has obvious advantages when compared with local discrimination-aware pruning utilized in DCP. Global discrimination-aware is a direct consequence of discrete channel settings introduced in our algorithm.

We further plot the channel configurations for each layer of ResNet-50 and MobileNetV2 after pruning in Fig. 4. Since early layers often have a large impact on FLOPs, most early layers are aggressively pruned. But there are some exceptions, layer-1 and layer-8 have a much higher preserved number of channels compared to adjacent layers in ResNet-50. Similar observations hold for layer-3 and layer-6 in MobileNetV2. These layers are regarded as important components in our method.

2.6.4 Impact of Gate Placement

In Tab. 3, we show how gate placement impacts the performance of the pruned model. 1-Gate represents removing gates with the dashed line in Fig 3. 2-Gate and 2-Gate-Shared are the original settings described in section 2.6.1. From this table, we can see that precise estimation of sub-network performance results in improvements in the final model ($+0.29\%$ on Top-1 for ResNet, $+0.64\%$ on Top-1 for MobileNetV2). This experiment clearly shows that the discrepancy between the precise and imprecise estimation of sub-network during gate training.

2.6.5 Understanding the Training of Discrete Gates

We draw related figures about the gate training process in Fig. 5. Four experiments are conducted given different values of p from 0.35 to 0.8 on CIFAR-10 with ResNet-56. Fig. 5 (a)-(d) show the test performance along with regularization loss. It's quite clear that there are two stages for gate training. In the first stage, the test accuracy of the sub-network sharply drops, at the same time, regularization loss drops too. In the second stage, regularization loss is near 0, the test performance continues to increase until the end of the training process. This shows that our DMC algorithm can consistently find sub-networks with better performance. Fig. 5 (e)-(h) show the progress of pruning rate for different layers. We can see the pruning rate of a layer dramatically

decreases at the beginning. However, within the process of the second training stage, some pruned channels of certain layers are recovered. This observation shows that our method indeed explores the search space instead of stacking at a trivial solution with a fixed sub-network. In Fig.5 (i)-(l), we plot the value of $s = \frac{\sum_{l=1}^{L}\sum_{c=1}^{C_l}|\theta_{l,c}-0.5|}{n}$. This value can roughly measure whether the discrete gate is inclined to deterministic (θ is close to 0 or 1) or stochastic (θ is close to 0.5). At the beginning of training, s dramatically drops, depicting that increased stochasticity when sampling a sub-network. After training for a while, there is a turning point that the gates start to become more deterministic. This is mainly because that the information from the samples is utilized to reduce the stochasticity and make it more confident towards the final sub-network.

2.7 Conclusions

In this paper, we proposed an effective discrete model compression method to prune CNNs given certain resource constraints. By turning deterministic discrete gate to stochastic discrete gate, moreover, our method can explore larger search space of sub-networks. To further enlarge the space, we introduced the symmetric weight decay on the gate parameters inspired by the fact that regularization loss can be regarded as weight decay. Our method also benefits from the exact estimation of sub-networks' outputs because of a combination of the precise placement of gates and the discrete setting. Extensive experiments results on ImageNet and CIFAR-10 show that our method outperforms state-of-the-art methods.

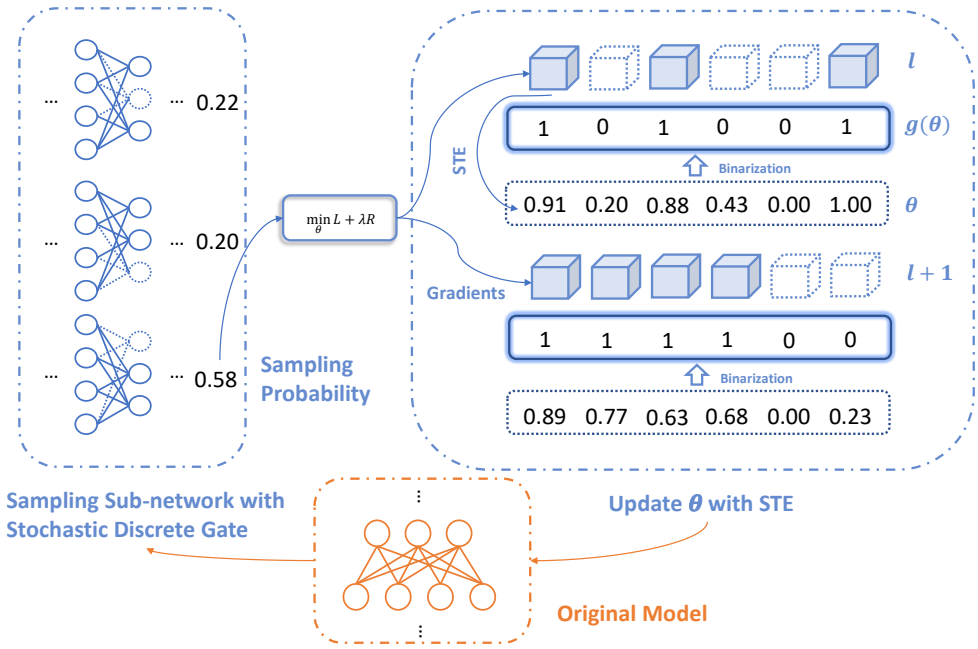
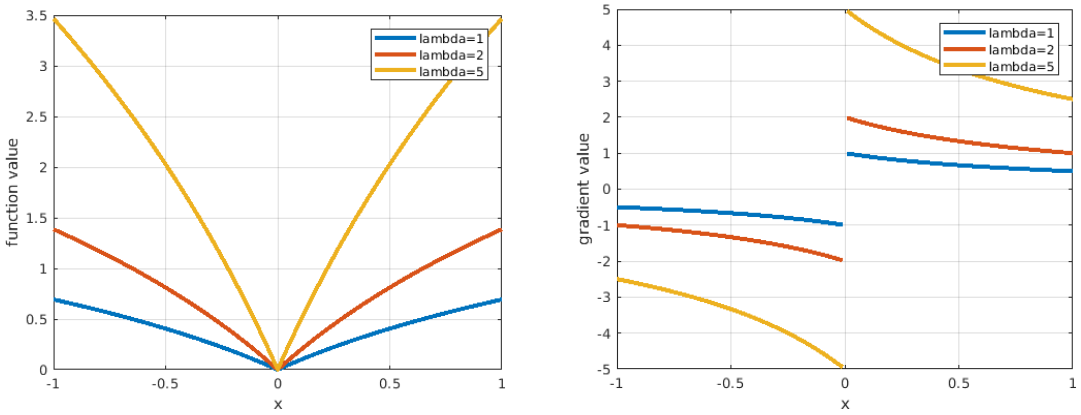


Figure 1: The gate training process of the proposed method. A sub-network is sampled according to Eq. (6-4). Then θ is optimized through STE with gradient descent. At the next iteration, the sub-network is sampled with the updated θ.



(a) function value of the regularization

(b) gradients of the regularization

Figure 2: The values and gradients of the resource regularization, y is set to 0 for better visualization.

Algorithm 1: Discrete Model Compression

input: dataset for training gate, D_{gate}; remaining rate of FLOPs, p; regularization hyper-parameter, λ; symmetric weight decay parameter β; gate training epochs num-E; pre-trained model f.

Freeze $\mathcal{W}$ and batchnorm parameters in f.

Initialize all $\theta_{l,c}$ to 1.

for $e := 1$ *to num-E* **do**

  shuffle(D_{gate})

  foreach x, y *in* D_{gate} **do**

    1. forward calculation: $\min_{\Theta} \mathcal{F}(\Theta) = \mathcal{L}(f(x; \mathcal{W}, \Theta), y) + \lambda \mathcal{R}_{\log}$

    2. calculate gradient w.r.t $\theta_{l,c}$.

    3. update each $\theta_{l,c}$ by ADAM optimizer.

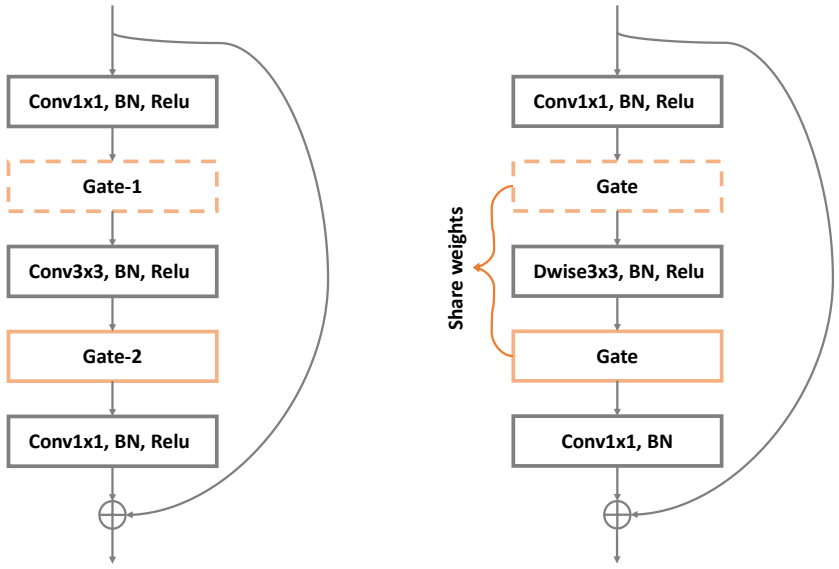
    4. apply symmetric weight decay on $\theta_{l,c}$ defined in Eq. (2-12).

    5. clip each $\theta_{l,c}$ to [0,1].

  end

end

return model f with the final Θ.

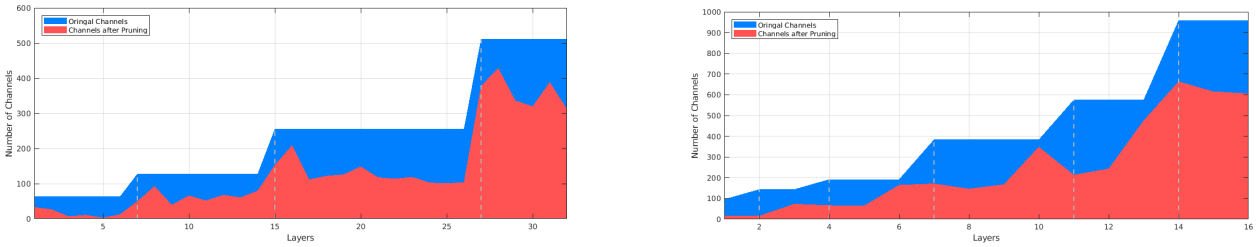


(a) Bottleneck Block (b) Inverted Residual Block

Figure 3: Gate placement for different architectures. (a) Bottleneck Block for ResNet. (b) Inverted Residual Block for MobileNetV2.

Table 2: Comparison results on ImageNet dataset with ResNet-34, ResNet-50, ResNet-101 and MobileNetV2. Δ-Acc represents the performance changes before and after model pruning. +/- indicates increase or decrease compared to baseline results. ThiNet on MobileNetV2 results are from DCP [154] paper.

Method	Architecture	Baseline Top-1 Acc	Baseline Top-5 Acc	Δ-Acc Top-1	Δ-Acc Top-5	Pruned FLOPs
Pruning Filters [88]	ResNet-34	73.23%	-	-1.06%	-	24.8%
Soft Prunings [48]		73.93%	91.62%	-2.09%	-1.92%	41.1%
IE [112]		73.31%	-	**-0.48%**	-	24.2%
FPGM [49]		73.92%	91.62%	-1.29%	-0.54%	41.1%
DMC(ours)		73.30%	91.42%	-0.73%	-0.31%	**43.4%**
Soft Pruning [48]	ResNet-50	76.15%	92.87%	-1.54%	-0.81%	41.8%
IE [112]		76.18%	-	-1.68%	-	45%
FPGM [49]		76.15%	92.87%	-1.32%	-0.55%	53.5%
GAL [98]		76.15%	92.87%	-4.35%	-2.05%	55.0%
DCP [154]		76.01%	92.93%	-1.06%	-0.61%	**55.6%**
CCP [119]		76.15%	92.87%	-0.94%	-0.45%	54.1%
DMC(ours)		76.15%	92.87%	**-0.80%**	**-0.38%**	55.0%
Rethinking [143]	ResNet-101	77.37%	-	-2.10%	-	47.0%
IE [112]		77.37%	-	-0.02%	-	39.8%
FPGM [49]		77.37%	93.56%	-0.05%	0.00%	41.1%
DMC(ours)		77.37%	93.56%	**+0.03%**	**+0.04%**	**56.0%**
ThiNet* [108]	MobileNetV2	70.11%	-	-6.40%	-4.60%	44.7%
DCP [154]		70.11%	-	-5.89%	-3.77%	44.7%
DMC(ours)		71.88%	90.29%	**-3.51%**	**-1.83%**	**46.0%**

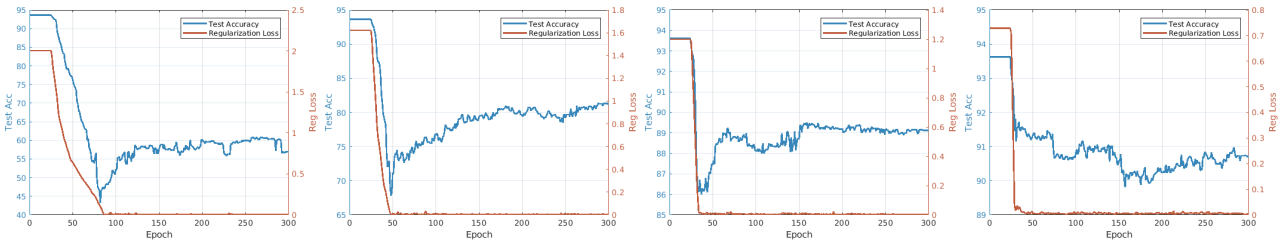


(a) ResNet-50 (b) MobileNetV2

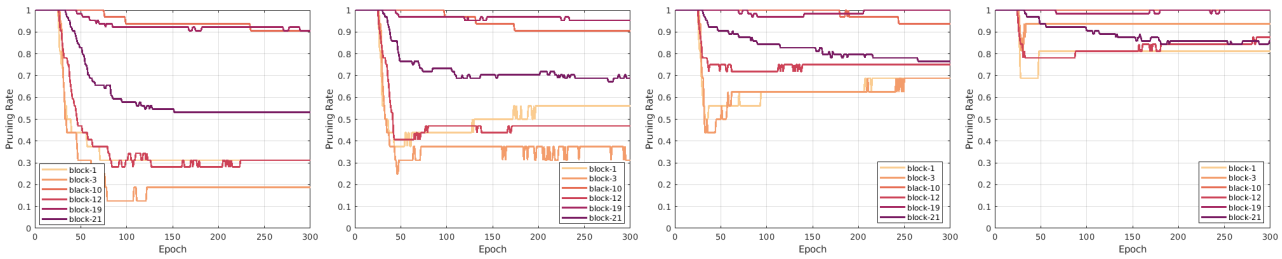
Figure 4: Networks discovered by our method from ResNet-50 and MobileNetV2. Dashed line indicates channel number changes in the original model.

Table 3: Performance of pruned models given different gate settings on ImageNet.

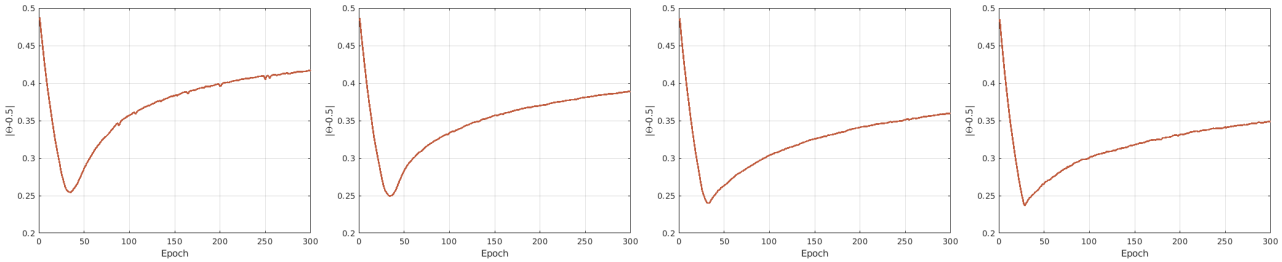
Gate Setting	Architecture	Top-1 Acc	Top-5 Acc	Pruned FLOPs
1-Gate	ResNet-50	75.06%	92.41%	55.2%
2-Gate		75.35%	92.49%	55.0%
1-Gate	MobileNetV2	67.73%	88.14%	45.3%
2-Gate-Shared		68.37%	88.46%	46.0%



(a) 35% FLOPs (b) 50% FLOPs (c) 65% FLOPs (d) 80% FLOPs



(e) 35% FLOPs (f) 50% FLOPs (g) 65% FLOPs (h) 80% FLOPs



(i) 35% FLOPs (j) 50% FLOPs (k) 55% FLOPs (l) 80% FLOPs

Figure 5: (a)-(d): Test set performance and regularization loss with the progress of gate training. (e)-(h): Pruning rate on different layers with the progress of gate training. (i)-(l): Evolution of gate randomness during the progress of gate training.

3.0 Network Pruning via Performance Maximization

3.1 Background

Convolutional Neural Networks (CNNs) have demonstrated great successes in many computer vision and machine learning applications, like classification [83], detection [122, 123], action recognition [129] and self-driving cars [3]. To achieve better performances on these tasks, the design of CNNs becomes more and more complex in terms of depth and width [130, 46, 69] since AlexNet [83]. However, the huge consumption of computing power and memory footprint prevents these complex CNNs to be deployed on embedded or mobile devices. To overcome this problem, model compression emerges as a promising solution to get a compact sub-network from the original model. Popular model compression techniques include weight pruning [43], quantization [7], structural pruning [88] and so on.

Channel pruning, which belongs to structural pruning, effectively reduces FLOPs and memory footprint from the original model without any post-processing steps. On the contrary, weight pruning or quantization usually requires specifically designed software or hardware to achieve actual acceleration. As a result, we aim to develop a novel model compression method for channel pruning.

In the context of channel pruning, a sub-network with high accuracy is believed as a good candidate for the final solution [50, 102]. To find such a sub-network, many existing channel pruning approaches [77, 145, 154] use the classification loss as guidance. However, the classification loss is not always a good approximation to the accuracy, which is also termed as loss-metric mismatch [58]. To tackle this problem, we train a performance prediction network to predict the accuracy of sub-networks. We then guide the search of sub-networks by directly maximizing the accuracy. In addition, we do not abandon the classification loss (usually, cross-entropy loss), and both classification loss and performance maximization are considered in the final pruning problem defined in Eq. 3-8, which is inspired by the idea of multi-objective learning. The rationale behind this is that both the classification loss and performance maximization provide useful but different information for pruning, and merging them will lead to better results.

The training of the performance prediction network has several difficulties. How to collect samples for training this network is not clear. Random sampling often produces trivial sub-networks (performance near-random guessing). To get meaningful sub-networks, we start from the original network and prune it to the given budget using a differentiable pruning approach. We can directly use the sub-network and mini-batch accuracy as a sample to train the performance prediction network during this pruning process. However, only using the latest sub-network will lead to catastrophic forgetting [24], where the performance prediction network may forget the information about previous sub-networks. To tackle this issue, we use an episodic memory module to collect samples along the pruning trajectory. Directly using these samples is problematic since the accuracy distribution of these samples is far from uniform. This problem is solved by re-sampling these samples. With above techniques, the performance prediction network is incrementally trained during the pruning process. After the performance prediction network visits enough samples and is confident enough, it is then put into the pruning process to provide additional supervision for channel pruning. Since the training of the performance prediction network and pruning proceed simultaneously, there is no extra cost.

Our main contributions can be summarized as follows:

1) We propose a novel channel pruning method for CNNs by directly maximizing the accuracy of sub-networks. To the best of our knowledge, this is the first paper to consider the problem of loss-metric mismatch for network pruning.
2) We train a performance prediction network, and use it as a proxy of accuracy metric for sub-networks. Our method further leverages the benefits from both performance maximization and classification loss to guide search of sub-networks.
3) Extensive experimental results show that our method can achieve the state-of-the-art performance with ResNet, MobileNetV2, and ShuffleNetV2+ on ImageNet and CIFAR-10.

3.2 Related Works

3.2.1 Model Compression

Weight pruning. Weight pruning tries to eliminate weights from the original model. Early works either add sparsity induced penalty on the weights [139] or remove weights based on the sensitivity [84, 76]. A more recent weight pruning work [43] prunes weights according to their L_2 or L_1 norms. Another work by *Dong et al.* [18] explored weight pruning based on second derivative information. Other weight pruning methods include data-driven pruning [54], variational dropout [111], utilizing determinantal point process [110] and so on. Different from the above works, the lottery ticket hypothesis [22] argues that there exist good sub-networks within a randomly initialized large network. Follow up works [23, 114] show that this claim can be extended to different architectures and datasets. While weight pruning achieves many good results when compressing a neural network, the saving in terms of computational cost is not optimal since the sparse weight matrices cannot be efficiently utilized by modern hardware.

Structural pruning. Structural pruning aims to remove redundant structures such as filters, channels, or layers. Unlike weight pruning, structural pruning can reduce inference time without specialized hardware or software support. Given a pre-trained model, filter pruning [88] removes filters based on their L_1 norms. Soft filter pruning [48] keeps all filters during training but resets least important ones (smaller norm). These two methods use group norms to indicate the importance of structures. Sparse structure selection [70] adds learnable scaling factors to prune neurons or layers with the sparsity regularization. Discrimination-aware pruning [154] considers both classification loss and norms to guide the pruning of the model. Gate decorator [145] inserts learnable factors for each channel and then uses Taylor expansion of the loss to estimate the global importance. Operation-aware pruning [75] makes a joint consideration of batchnorm and ReLU operations and learns differentiable masks for individual channels. These methods all use classification loss to help the structural pruning. Automatic model compression (AMC) [50] adapts reinforcement learning for structural pruning. With reinforcement learning, model accuracy can be directly used in reward function to guide the pruning process. Metapruning [102] utilizes a hypernet to predict the weights of sub-networks, and it then applies the evolutionary search for

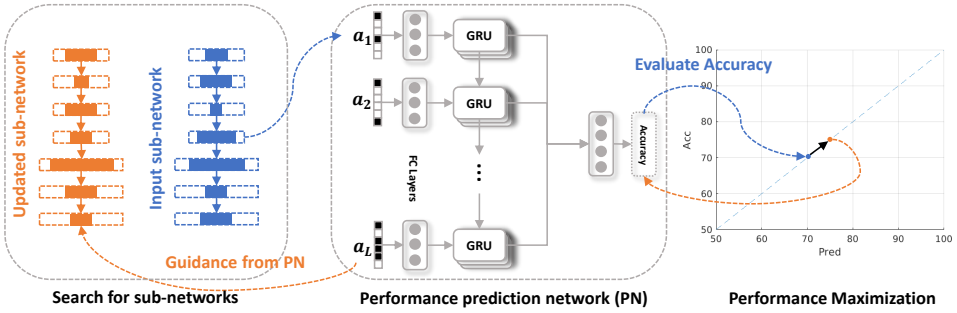


Figure 6: The flowchart of performance maximization process. A sub-network is first sampled from differentiable gates. The performance prediction network is then used as a proxy to maximize the accuracy of sub-networks.

pruning. With an evolutionary search, accuracy is directly used as guidance.

Differentiable pruning approaches [75, 145, 77, 32] are more efficient and easy to train compared to reinforcement learning or evolutionary search. However, it can not use the accuracy to guide pruning due to the accuracy metric is not differentiable. The mismatch of accuracy and classification loss may lead to inferior performance. To leverage the benefits from both sides, we aim to maximize the accuracy of sub-networks within a differentiable pruning framework.

3.2.2 Performance Prediction

To our best knowledge, predicting the performance of a neural network is not well studied within the context of model compression. There exist several works to predict the accuracy of a neural network based on some different conditions. A recent work [133] tries to connect the model accuracy with the weights of the model. With simple statistics of weights, accuracy predictors can correctly rank neural networks by their performance. In [74], they use margin distributions of multiple layers to predict the generalization gaps of neural networks. A more related work [72] trains an accuracy predictor for neural architecture search (NAS) to include both architecture information and characterization of the dataset-difficulty. Training an accuracy predictor for NAS is quite time-consuming, and every network has to be trained from scratch to provide its accuracy.

We do not want to spend so much computational cost for acquiring these samples. To save costs and get meaningful sub-networks, we prefer to collect sub-networks in-place during the pruning process.

3.3 Network Pruning via Performance Maximization

3.3.1 Notations

To better describe our approach, necessary notations are introduced first. In a CNN, the feature map of i-th layer can be represented by $\mathcal{F}_i \in \mathcal{R}^{C_i \times W_i \times H_i}$, $i = 1, \ldots, L$, where C_i is the number of channels, W_i and H_i are height and width of the current feature map, L is the number of layers. The mini-batch dimension of feature maps is ignored to simplify notations. $\mathbb{1}(\cdot)$ is the indicator function. $\odot$ is the element-wise product.

3.3.2 Generate Sub-networks

As we discussed previously, directly sampling sub-networks often produces trivial results, especially when pruning rate is high. To train a performance prediction network for channel pruning, we do not need all sub-networks. Suppose the FLOPs of the original model is T_{total} and the pruning rate is p, we are interested in sub-networks with FLOPs from pT_{total} to T_{total}. Sub-networks with FLOPs lower than pT_{total} are discarded since they do not satisfy FLOPs constraint. As a result, we prefer to generate meaningful sub-networks with certain FLOPs as training samples for the performance prediction network.

We start to generate these sub-networks by pruning the original model to the target FLOPs pT_{total}. To achieve this, we first introduce the basic differentiable pruning algorithm. We use differentiable gates to characterize a channel. For ith layer, gates are defined as:

$$o_i = 1/(1 + e^{-(w_i + s)/\tau}), \tag{3-1}$$

where $1/(1 + e^{-x})$ is a sigmoid function, $o_i \in \mathcal{R}^{C_i}$ and $o_i \in [0, 1]$, $w_i \in \mathcal{R}^{C_i}$ are learnable parameters of this gate, s is sampled from Gumbel distribution: $s \in \text{Gumbel}(0, 1)$, and τ is a

hyperparameter to control sharpness. o_i here is continuous, to precisely generate sub-networks, we further round it to 0 or 1:

$$a_i = \mathbb{1}_{o_i > \frac{1}{2}}(o_i), \tag{3-2}$$

where $a_i \in \{0, 1\}^{C_i}$. Since the indicator function $\mathbb{1}(\cdot)$ is not differentiable, we use straight-through estimator [2] to calculate gradients. The differentiable gate in Eq. 3-1 and Eq. 3-2 uses Gumbel-Solftmax [73] technique to approximate Bernoulli distribution. Although there are alternative techniques to approximate Bernoulli distribution, we found that the difference is not significant.

To prune channels of a CNN, we apply gates on the feature map $\mathcal{F}_i$:

$$\hat{\mathcal{F}}_i = a_i \odot \mathcal{F}_i, \tag{3-3}$$

where a_i is expanded to the same size of $\mathcal{F}_i$. The optimization of the pruning process is given by:

$$\min_{w} \mathcal{L}(f(x; \mathbf{a}, \Theta), y) + \mathcal{R}(T(\mathbf{a}), pT_{\text{total}}), \tag{3-4}$$

where $\mathbf{w}$ contains all learnable weights of gates defined in Eq. (3-1). Here $\mathbf{a}$ is a vector representing the structure of a CNN: $\mathbf{a} = \text{cat}(a_1, \cdots, a_i, \cdots, a_L), i = 1, \cdots, L$, $T(\mathbf{a})$ is the FLOPs defined by the sub-network structure $\mathbf{a}$, x, y are input images and labels, $f(\cdot; \mathbf{a}, \Theta)$ here is a CNN parameterized by Θ, and its structure is decided by $\mathbf{a}$. $\mathcal{R}(T(\mathbf{a}), pT_{\text{total}}) = \log(\max(T(\mathbf{a}), pT_{\text{total}})/pT_{\text{total}})$ is the regularization term to push sub-networks to reach the target FLOPs.

During the optimization of Eq. 3-4, many sub-networks with different structures $\mathbf{a}$ are generated. If accuracy q is calculated based on $\mathbf{a}$ given a mini-batch, we can have a pair of sample ($\mathbf{a}$, q) representing a sub-network and its accuracy. Although the mini-batch accuracy may not be a good proxy of the true accuracy, we have a starting point at least.

3.3.3 Performance Prediction Network

Once we have ($\mathbf{a}$, q), we can train a neural network to predict the performance given the structure of a sub-network. We firstly define the performance prediction network: $q_{\text{pred}} = \text{PN}(\mathbf{a})$. $\text{PN}(\cdot)$ is the proposed performance prediction network. We use the sigmoid function as the output activation, and q_{pred} is in the range between 0 and 1.

The performance prediction network is composed of fully-connected layers and GRU [12]; the detailed settings are listed in the supplementary materials. In short, fully-connected layers transform each layer's structure vector into a compact representation, and GRU is used to connect different layers. We use GRU since a_{i-1} and a_i have implicit dependence. By doing so, the performance prediction network has the potential to capture complex interactions within a sub-network.

The optimization of PN is a regression problem, we use mean absolute error loss (MAE) to optimize it:

$$\min_{\mathbf{w}_{\text{P}}} \mathcal{L}_{\text{P}} = |q - \text{PN}(\mathbf{a})|, \tag{3-5}$$

where $\mathbf{w}_{\text{P}}$ is the weights of the performance prediction network. The target q is also normalized within $[0, 1]$ to facilitate the training.

3.3.4 Episodic Memory Module

The early version of this work directly utilizes the sub-network from the current iteration to train the performance prediction network. However, we found that it only deteriorates the pruning process. After carefully examining the results, the performance prediction network can hardly predict early sub-networks. This phenomena is known as catastrophic forgetting [24]. To overcome this issue, we need to periodically replay previous sub-networks. We further propose an episodic memory module to remember early sub-networks. The episodic memory is defined as: $\text{EM} = (\mathcal{A}, \mathcal{Q})$, where $\mathcal{A} \in \mathcal{R}^{m \times K}$ and $\mathcal{Q} \in \mathcal{R}^{K}$, m is the length of vector $\mathbf{a}$ and K is the current size of the episodic memory. When adding one sub-network to the episodic memory, K is increased by 1, and K is smaller than K_{max}.

As we mentioned before, mini-batch accuracy is not a good estimation of the accuracy. On the other hand, the computational cost is too expensive if we use the whole training dataset to calculate

the accuracy. To leverage efficiency and precision, we collect sub-networks and corresponding mini-batch accuracy for every c iterations to construct an enhanced representation of sub-networks. The enhanced representation of sub-networks is:

$$\bar{\mathbf{a}} = \mathbb{1}_{\mathbf{a}>\frac{1}{2}}(\frac{1}{c}\sum_{i=1}^{c}\mathbf{a}_i),\ \bar{q} = \frac{1}{c}\sum_{i=1}^{c}q_i. \tag{3-6}$$

Within certain iterations, sub-networks produced by Eq. 3-1 and Eq. 3-2 are similar, due to the nature of differentiable pruning. Thus, using $(\bar{\mathbf{a}}, \bar{q})$ as a sample is reasonable. If c is too large, then above arguments are not valid, and the enhanced representation is useless. We do not calculate gradient when collecting sub-networks.

Suppose we already have K sub-networks in the episodic memory module, then the EM is updated by:

$$\begin{cases} \mathcal{A}_i = \bar{\mathbf{a}},\ \ i = \underset{i}{\operatorname{argmin}} \left|\mathcal{Q}_i - \bar{q}\right| & \text{if } K = K_{max}, \\ \mathcal{A}_{K+1} = \bar{\mathbf{a}} & \text{otherwise.} \end{cases} \tag{3-7}$$

$\bar{\mathbf{a}}$ is the sub-network defined in Eq. 3-6, the update of $\mathcal{Q}$ is done in a similar way. When $K < K_{\text{max}}$, the update of episodic memory is a simple insert process. When $K = K_{\text{max}}$, we replace the item in the episodic memory with the closest accuracy to the current sample. In fact, most of sub-networks during the pruning process have similar performance after the target FLOPs is met. As a result, we use K_{max} to encourage the diversity of sub-networks.

3.3.5 Imbalanced Accuracy Distribution

In Fig. 7, we plot the empirical distribution of the accuracy from sub-networks during the pruning process. In the figure, the accuracy is concentrated around 84. To prevent the performance prediction network from providing trivial solutions and making it converge faster, we re-sample the sub-networks according to their accuracy. All sub-networks are split into N groups according to $\mathcal{Q}$ with equal margin $\frac{1}{N-1}(\max(\mathcal{Q}) - \min(\mathcal{Q}))$. Then, we count sub-networks in each group and re-sample them according to the inverse of their count. This is equivalent to create N pseudo-classes and conduct re-sampling.

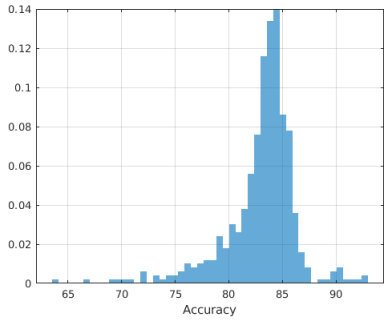


Figure 7: Empirical distribution of the accuracy from sub-networks collected during the pruning process. The results are based on ResNet-56 on CIFAR-10.

3.3.6 Performance Maximization

After having a relative confident performance prediction network, we start to maximize the performance for searching better sub-networks. The performance of a sub-network can be represented as $\text{PN}(\mathbf{a})$, thus we can maximize $\text{PN}(\mathbf{a})$ as a proxy of accuracy. $\underset{w}{max}\ \text{PN}(\mathbf{a})$ is equivalent to $\underset{w}{min}\ \frac{1}{\text{PN}(\mathbf{a})}$. To stabilize the training, we optimize the follwing problem instead: $\underset{w}{min}\ \log(\frac{1}{\text{PN}(\mathbf{a})})$. The overall optimization problem is shown in the following equation:

$$\begin{aligned}\underset{w}{min}\ \mathcal{J}(\mathbf{w}) =& \mathcal{L}(f(x;\mathbf{a},\Theta),y) + \gamma_{(K,\mathcal{L}_\text{P})} \cdot \log(\frac{1}{\text{PN}(\mathbf{a})}) \\ &+ \lambda\mathcal{R}(T(\mathbf{a}), pT_\text{total}),\end{aligned} \tag{3-8}$$

where $\gamma_{(K,\mathcal{L}_\text{P})}$ is a function to reflect the confidence of the performance prediction network and it is used to automatically control the magnitude of $\log(\frac{1}{\text{PN}(\mathbf{a})})$, λ is used to control the magnitude of the regularization, and the other terms are introduced in Eq. 3-4. $\gamma_{(K,\mathcal{L}_\text{P})}$ is defined as: $\gamma_{(K,\mathcal{L}_\text{P})} = \mathbb{1}_{K \geq \frac{K_\text{max}}{4}}(K) \cdot (1-\mathcal{L}_p)^2$, and the range of $\gamma_{(K,\mathcal{L}_\text{P})}$ is $[0,1]$. Usually, lower $\mathcal{L}_\text{P}$ indicates higher confidence of $\text{PN}(\cdot)$. However, the training of PN is an incremental learning task, $\mathcal{L}_\text{P}$ maybe unreliable until PN visits enough samples.

Although there exists loss-metric mismatch, the information from the loss function and performance maximization still has some overlaps. Since we already use the classification loss, it's desirable to acquire unique information from performance maximization. To achieve this, we make

Algorithm 2: Network Pruning via Performance Maximization

Input: D, p, λ, E, f, $K_{\max}$, c, b_{P}.
Initialization: initialize $\mathbf{w}$; randomly initialize $\mathbf{w}_{\mathrm{P}}$ for PN. initialize $K = 0$
for $e := 1$ *to* E **do**
  shuffle(D)
  for *a mini-batch* (x, y) *in* D **do**
    1. generate a structure vector $\mathbf{a}$ from Eq. 3-1 and Eq. 3-2 and its accuracy q
    2. update the sub-network according to Eq. 3-6.
    3. update **EM** with Eq. 3-7 and K every c iterations.
    if $K > b_P$ **then**
      4. update $\mathbf{w}_p$ with Eq. 3-5 with a mini-batch sampled from **EM**.
    end
    5. calculate gradients for $\mathbf{w}$ by backpropagation through Eq. 3-8.
    6. modify gradients according to Eq. 3-9.
    7. update $\mathbf{w}$ with ADAM.
  end
end
return $\mathbf{w}$.

the gradients orthogonal to each other. Let $g_{\mathcal{L}}^i = \frac{\partial \mathcal{L}}{\partial \mathbf{w}_i}$ represents the gradient from the classification loss of ith layer, and $g_{\mathrm{P}}^i = \frac{\partial \log(\frac{1}{\mathrm{PN}_{(\mathbf{a})}})}{\partial \mathbf{w}_i}$ be the gradient from performance maximization. The modified gradients from these two terms are:

$$g^i = g_{\mathcal{L}}^i + \hat{g}_{\mathrm{P}}^i, \tag{3-9}$$

where g_{P}^i is decomposed to two parts: $g_{\mathrm{P}}^i = \hat{g}_{\mathrm{P}}^i + \bar{g}_{\mathrm{P}}^i$, $\hat{g}_{\mathrm{P}}^i \perp g_{\mathcal{L}}^i$, and $\bar{g}_{\mathrm{P}}^i$ has the same direction with $g_{\mathcal{L}}^i$.

The overall algorithm is shown in Alg. 2. The process of performance maximization is shown in Fig. 6. The explanation of inputs is listed here: D: dataset, p: pruning rate of FLOPs, λ is introduced in Eq. 3-8, E: number of trainig epochs, f: the pre-trained CNN, $K_{\max}$ and c: hyperparmeters for episodic memory, and b_{P}: mini-batch size when training PN. As shown in Alg. 2, we perform channel pruning and training of PN simultaneously with little extra computational cost. The calculation of $g_{\mathbf{P}}$ and updates of $\mathbf{w}_{\mathbf{P}}$ is much cheaper compared to $g_{\mathcal{L}}$. The problem in Eq. 3-8 simultaneously minimizes classification loss and maximizes accuracy of sub-networks, thus better aligns loss and accuracy. Given the complexity of the problem, solely using the classification loss or performance maximization may lead to sub-optimal results. Moreover, the information from two perspectives is different, and we can achieve a better result by merging them.

Table 4: Comparison on the accuracy changes (Δ-Acc) and reduction in FLOPs of various channel pruning algorithms on CIFAR-10. +/- indicates increase/decrease compared to baselines.

Method	Architecture	Baseline Acc	Pruned Acc	Δ-Acc	$\downarrow$ FLOPs
AMC [50]	ResNet-56	92.80%	91.90%	-0.90%	50.0%
SFP [48]		93.59%	93.35%	-0.24%	52.6%
DCP [154]		93.80%	93.81%	+0.01%	47.0%
CCP [119]		93.50%	93.42%	-0.08%	**52.6%**
HRank [96]		93.26%	92.17%	-0.09%	50.0%
Pruning Criterion [47]		93.59%	93.24%	-0.35%	52.6%
NPPM(ours)		93.04%	93.40%	**+0.36%**	50.0%
WM [154]	MobileNetV2	94.47%	94.17%	-0.30%	26.0%
DCP [154]		94.47%	94.69%	+0.22%	26.0%
NPPM(ours)		94.23%	94.75%	**+0.52%**	**47.0%**

The proposed PN shares certain properties of the value function [80]. In the context of reinforcement learning, the value function is trained along the way of exploring the search space. The PN is also trained when searching sub-networks. However there exists some obvious differences. A value function is generally used in a Markov Decision Process, aiming to reduce the variance of gradient-based policy optimization methods but giving no direct guidance. While our method considers a stochastic optimization problem, the PN directly guides the search of sub-networks.

3.4 Experiments

3.4.1 Implementation Details

In the experiment section, our method is dubbed as NPPM (**N**etwork **P**runing via **P**erformance **M**axmization). We use CIFAR-10 [82] and ImageNet [15] to verify the performance of our method, as they are used in many model compression works.

On CIFAR-10, we evaluate our method on ResNet-56 and MobileNetV2. For ImageNet, ResNet-34/50/101 [46], MobileNetV2 [126] and ShuffleNetV2+ [109, 125] are used for evaluation. ShuffleNetV2+ is an improved version of ShuffleNetV2 which has similar performance with MobileNetV3 [53]. These models are generally harder to prune compared with AlexNet or VGG. We use p to decide how much FLOPs should be removed, the detailed choices of p are listed in supplementary materials. We choose $\lambda = 2$ used in Eq. 3-8 for all experiments.

We train ResNet-56 and MobileNetV2 on CIFAR-10 from scratch following pytorch examples. After pruning, we finetune the model for 200 epochs using SGD with a start learning rate 0.01, weight decay 0.0001, and momentum 0.9. The learning rate is decayed to 0.01 and 0.001 at epoch 100 and 150. One benefit of our method is that we can directly prune pre-trained models. Thus, we use pre-trained models released from pytorch or their official implementation on ImageNet. After pruning, we finetune ResNet models for 100 epochs using SGD with a start learning rate 0.01, and the learning rate is multiplied by 0.1 at epoch 30, 60 and 90. For MobileNetV2 on ImageNet, we use cosine annealing scheduler with a start learning rate 0.01 and also finetune for 100 epochs following their original paper [126]. For ShuffleNetV2+, we decay the learning rate at every step and finetune for 100 epochs with a start learning rate 0.1 following the original settings [109, 125] too.

When training $\mathbf{w}$ and $\mathbf{w}_{\text{P}}$, we use ADAM [79] optimizer with a constant learning rate 0.001 and train them for 200 epochs. To produce a near all-one vector for $\mathbf{a}$, $\mathbf{w}$ is initialized to 3. The training is conducted on a subset of the whole dataset. We use 2,500 and 10,000 samples for CIFAR-10 and ImageNet separately. A stand-alone validation set is not necessary; subsets come from the training set directly. We set K_{max} and c as 500 and 5 on both datasets. The mini-batch size is 64, 128, and 512 for the performance prediction network, CIFAR-10, and ImageNet. All codes are implemented with pytorch [118]. The experiments are conducted on a machine with 4 Nvidia Tesla P40 GPUs.

3.4.2 CIFAR-10 Results

In Tab. 7, we present the results of ResNet-56 and MobileNetV2 on CIFAR-10. Our method has the best Δ-Acc with ResNet-56. After pruning, our method improves the baseline performance by 0.36%. Our method is better than the second best DCP by 0.35% in terms of Δ-Acc with

similar FLOPs (ours 50% vs. DCP 47%). The advantage of our method is quite obvious when comparing with other methods. HRank and Pruning Criterion are two recent methods, and our method can achieve better accuracy and Δ-Acc than these two methods. For MobileNetV2 on CIFAR-10, our method can prune 47% of FLOPs, and the accuracy is improved by 0.52%. Our method prunes most FLOPs and achieves the best accuracy. Compared with DCP, our method prunes 21% more FLOPs and still achieves better accuracy. The results of CIFAR-10 show that performance maximization can improve network pruning.

3.4.3 ImageNet Results

We present all results for ImageNet in Tab. 8. The FLOPs of the original models are 3.68G, 4.12G and 7.85G for ResNet-34, ResNet-50 and ResNet-101. The FLOPs of MobileNetV2 and ShuffleNetV2+(Small) are 314M and 156M. Compared to CIFAR-10, ImageNet is more reliable when evaluating model compression methods.

ResNet-34: With ResNet-34, our method can achieve the best top-1 accuracy by pruning 44% of FLOPs. Our method largely outperforms FPGM and SFP when pruning similar FLOPs (44.0% vs. 41.1%). Specifically, the pruned top-1 accuracy is 1.17% and 0.38% higher than SFP and FPGM separately, and similar observations hold for other measurements, like Δ Top-1 accuracy. IE and Pruning Filters prune around 24% FLOPs. Usually, pruned Top-1 accuracy is higher with a smaller amount of pruned FLOPs. Our method can prune 20% more FLOPs and still achieves better performance than IE. In short, our method prunes more FLOPs with less performance drop on ResNet-34.

ResNet-50: ResNet-50 is a very popular model when evaluating pruning algorithms. Thus, more comparison methods are listed. Our approach can prune 56.0% of FLOPs with marginally performance loss on top-1 and top-5 accuracy (0.19% and 0.12% separately). LeGR can achieve the second best result on pruned top-1 accuracy, and our method prunes 14% more FLOPs with less accuracy drop (0.26% and 0.21% higher on pruned Top-1 acc and Δ Top-1 acc). GBN and SCP have similar performance with Δ Top-1 accuracy, and their performance is higher than other comparison methods with similar FLOPs. Our approach can outperform GBN and SCP by at least 0.43% with Δ Top-1 accuracy. Overall speaking, pruning methods guided by the classification loss [145, 75, 154] have better results than rest approaches. On top of the classification loss, our

method utilizes information from performance maximization. The superb performance on ResNet-50 again demonstrates the effectiveness of the proposed performance maximization.

ResNet-101: ResNet-101 is a parameter-heavy model, and it is easier to prune compared to ResNet-34 and ResNet-50. With the same pruning rates of ResNet-50 (remove 56% FLOPs), our method can achieve $77.83\%/93.77\%$ Top-1/Top-5 accuracy, which is even higher than the original model ($+0.46\%/+0.21\%$ with Δ Top-1/Δ Top-5). IE and FPGM can prune around 40% of FLOPs with little accuracy drops. Compared with these methods, our approach can prune 16% more FLOPs (around 1.3G FLOPs) while achieving performance gain. Moreover, the FLOPs of the pruned model from our method has fewer FLOPs than the original ResNet-34 and ResNet-50 (pruned ResNet-101: 3.46G, ResNet-34/ResNet-50: 3.68G/4.12G.)

MobileNetV2: MobileNetV2 is a computationally efficient model, which makes it harder to prune. All methods prune around 30% of FLOPs. AMC, LeGR, and MetaPruning have a clear advantage over the uniform baseline, but they are worse than Greedy Pruning. Our method outperforms Greedy Pruning by 0.42% with Top-1 accuracy.

ShuffleNetV2+: ShuffleNetV2+ is a highly efficient model with a similar performance to MobileNetV3. On ShuffleNetV2+, we compare our method against uniform pruning and DG (differentiable gate in Eq. 3-4). DG can be seen as a variant of our method without performance maximization. Our method is better than uniform pruning by 1.14% on Top-1 accuracy. By directly comparing our method and DG, we can see that performance maximization helps the search of sub-networks and results in 0.44% improvements. The results on ShuffleNetV2+ and MobileNetV2 show that performance maximization improves the quality of sub-networks for both parameter-heavy models and computation-efficient models.

3.4.4 Analysis and Discussion

To provide a deeper understanding of our method, we plot the predictions from PN and the similarity between layer-wise gradients from two losses: $\text{sim}^i = \frac{(g^i_{\mathcal{L}})^T (g^i_{\text{P}})}{\|g^i_{\mathcal{L}}\| \|g^i_{\text{P}}\|}$ in Fig. 9. To plot predictions from PN, we sample 100 sub-networks in **EM** and calculate the accuracy on the sub-set. From Fig. 9 (e)$\sim$(h), we can see that the predicted performance closely matches the actual accuracy, which demonstrates that the information from PN is trustworthy. From the results of Fig. 9 (a)$\sim$(d), it's obvious that the gradients from the classification loss and PN are different ($\max(\text{sim}^i) < 0.55$),

the similarity becomes smaller when the dataset becomes more complex. These results show that the performance prediction network can provide different and reliable information to help search for sub-networks. Another interesting observation is that later layers often have smaller gradient similarity, showing that they are more sensitive to performance maximization.

In Fig 13 (a), we plot the results of different settings. PM w/o GM represents using PM without gradient modification, and DG represents differentiable gate again. Obviously, PM can achieve the best performance during the search of sub-networks. λ does not have strong impacts on the results, but if λ is too large, it may hinder the pruning process.

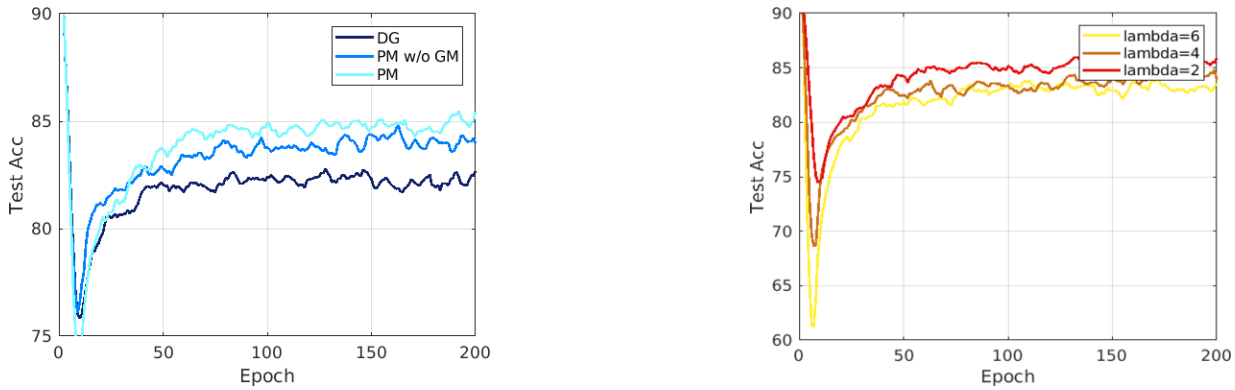
There is an ongoing debate on whether finetuning is useful when pruning neural networks [104]. Our results (Tab. 6) show that finetuning is necessary to achieve ideal performance on computation efficient models. The margin between finetuning and training from scratch is clear, demonstrating that both sub-network architecture and pre-trained weights are essential.

3.5 Conclusion

In this paper, we studied how to simultaneously achieve low loss value and high accuracy when searching for sub-networks. By using an episodic memory module and re-sampling techniques, we are able to train a performance prediction network in-place during pruning, which also saves computational resources. By utilizing information from the classification loss and performance maximization, our method is able to find good sub-networks during pruning. Extensive experiments on CIFAR-10 and ImageNet demonstrate that our method achieves state-of-the-art results.

Table 5: Comparison on the Top-1/Top-5 accuracy changes (Δ Top-1/Top-5) and reduction in FLOPs of various channel pruning algorithms on ImageNet. +/- indicates increase/decrease compared to baselines.

Method	Architecture	Pruned Top-1	Pruned Top-5	Δ Top-1	Δ Top-5	↓ FLOPs	FLOPs
Pruning Filters [88]	ResNet-34	72.17%	-	-1.06%	-	24.8%	2.77G
SFP [48]		71.84%	89.70%	-2.09%	-1.92%	41.1%	2.17G
IE [112]		72.83%	-	-0.48%	-	24.2%	2.79G
FPGM [49]		72.63%	91.08%	-1.29%	-0.54%	41.1%	2.16G
NPPM(ours)		**73.01%**	**91.30%**	**-0.29%**	**-0.12%**	**44.0%**	2.06G
DCP [154]	ResNet-50	74.95%	92.32%	-1.06%	-0.61%	55.6%	1.83G
CCP [119]		75.21%	92.42%	-0.94%	-0.45%	54.1%	1.89G
MetaPruning [102]		75.40%	-	-1.20%	-	51.2%	2.01G
GBN [145]		75.18%	92.41%	-0.67%	-0.26%	55.1%	1.85G
HRank [96]		74.98%	92.33%	-1.17%	-0.54%	43.8%	2.32G
Hinge [90]		74.70%	-	-1.40%	-	54.4%	1,88G
DSA [116]		74.69%	92.45%	-1.33%	-0.80%	50.0%	2.06G
SCP [75]		75.27%	92.30%	-0.62%	-0.68%	54.3%	1.88G
LeGR [9]		75.70%	92.70%	-0.40%	-0.20%	42.0%	2.39G
NPPM(ours)		**75.96%**	**92.75%**	**-0.19%**	**-0.12%**	**56.0%**	1.81G
Rethinking [143]	ResNet-101	77.37%	-	-2.10%	-	47.0%	4.16G
IE [112]		77.35%	-	-0.02%	-	39.8%	4.72G
FPGM [49]		77.32%	93.56%	-0.05%	0.00%	41.1%	4.80G
NPPM(ours)		**77.83%**	**93.77%**	**+0.46%**	**+0.21%**	**56.0%**	3.46G
MobileNetV2 0.75 [126]	MobileNetV2	69.80%	89.60%	-2.00%	-1.40%	30.0%	220M
AMC [50]		70.80%	-	-1.00%	-	30.0%	220M
MetaPruning [102]		71.20%	-	-0.80%	-	**30.7%**	217M
LeGR [9]		71.40%	-	-0.40%	-	30.0%	220M
Greedy Pruning [144]		71.60%	-	-0.40%	-	30.0%	220M
NPPM(ours)		**72.02%**	**90.26%**	**+0.02%**	**-0.12%**	29.7%	221M
Uniform	ShuffleNetV2+(Small)	71.92%	90.61%	-2.18%	-1.09%	23.1%	120M
DG		72.62%	91.00%	-1.48%	-0.70%	23.8%	119M
NPPM(ours)		**73.06%**	**91.10%**	**-1.04%**	**-0.60%**	**25.0%**	117M

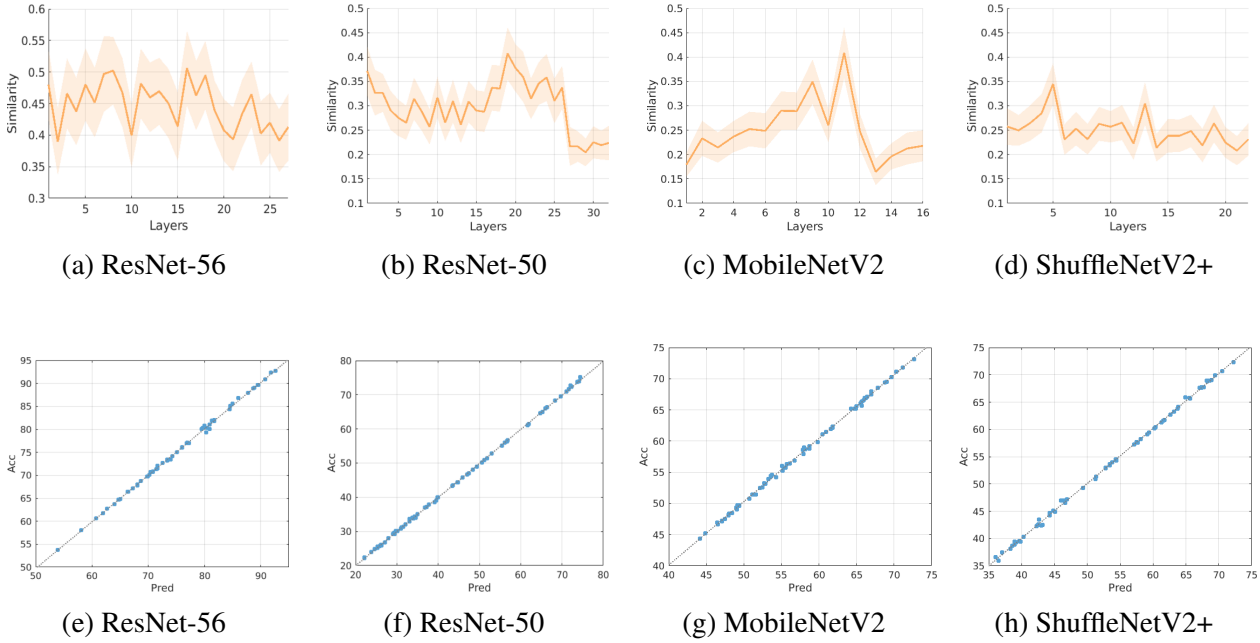


(a) (b)

Figure 8: (a): Test accuracy of sub-networks given different pruning settings. (b): Test accuracy of sub-networks given different choice of λ. Both experiments are on CIFAR-10 with ResNet-56

Table 6: Difference between finetuning and training from scratch for our method.

Setting	Architecture	Pruned Top-1	Pruned Top-5	Δ Top-1	Δ Top-5
Finetune	ShuffleNetV2+(Small)	73.06%	91.10%	-1.04%	-0.60%
Scratch		72.32%	90.86%	-1.78%	-0.84%
Finetune	MobileNetV2	72.02%	90.26%	+0.02%	-0.12%
Scratch		71.14%	89.71%	-0.86%	-0.67%



(a) ResNet-56 (b) ResNet-50 (c) MobileNetV2 (d) ShuffleNetV2+

(e) ResNet-56 (f) ResNet-50 (g) MobileNetV2 (h) ShuffleNetV2+

Figure 9: (a)∼(d): Gradient similarity between different losses. Shaded area shows standard deviation. (e)∼(h): Predicted accuracy and actual accuracy for some sub-networks. ResNet-56 is evaluated on CIFAR-10 and rest models are on ImageNet.

4.0 Disentangled Differentiable Network Pruning

4.1 Background

Convolutional Neural Networks (CNNs) have accomplished tremendous success in various computer vision tasks [83, 122, 123, 129, 3]. To deal with real-world applications, recently, the design of CNNs has become more and more complicated in terms of width, depth, etc. [83, 130, 46, 69]. These complex CNNs can attain better performance on benchmark tasks, but their computational and storage costs increase dramatically. As a result, a typical application based on CNNs can quickly exhaust an embedded or mobile device due to its enormous costs. Given such costs, the application can hardly be deployed on resource-limited platforms. To tackle these problems, many researches [43, 42] have been devoted to compressing the original large CNNs into compact models. Among these methods, weight pruning and structural pruning are two popular topics for model compression.

Unlike weight pruning or sparsification, structural pruning, especially channel pruning, is an effective way to truncate the computational cost of a model because it does not require any post-processing steps to achieve actual acceleration and compression. Many existing works [77, 145, 32, 75] try to discover compact sub-networks by optimizing a regularized loss function through differentiable operations. Usually, the width of a layer and the importance of each channel are entangled in this setting since they use one learnable parameter to characterize each channel. Specifically, the FLOPs or parameter constraints implicitly restrict the search space of the pruned model. On the other hand, search based pruning algorithms (through reinforcement learning [50], evolutionary algorithms [102] and so on) can directly generate the width of each layer with flexible importance definition, which leads to competitive performance. However, the costs of search based method is usually more expansive.

Previous differentiable pruning methods [77, 145, 32, 75] entangle width and importance, limiting the potential search space of sub-networks. To tackle this problem, we aim to disentangle the learning of width and importance, and consequently make pruning more flexible. The **disentanglement of pruning** can be understood as using *independent parameterization for channel*

importance and layer width. To achieve this, we first observe that the width of a certain layer can be represented by k of the top-k operation. Inspired by the definition of top-k, we then relax the hard equality constraint used in previous works to a soft regularization term, where k and importance scores can be optimized separately. By doing so, we partially disentangle the importance and width of a layer for pruning. Under our setting, the choices of channel importance become more flexible compared to previous works. Additionally, the width k of each layer can be generated directly, which shares similar property of search based algorithms. Following previous works, we also formulate the channel pruning problem as a constrained optimization problem, which can be efficiently optimized through regular SGD methods. Compared to differentiable pruning methods [77, 145, 32, 75], our method disentangles the learning of width and channel importance, which potentially enlarge the search space. Compared to search based algorithms [50, 102], our method provides a way to efficiently generate and optimize width without additional costs.

To make the learning efficient, we further parameterize the importance and width by using two neural networks. We use an importance generation network to capture inter-channel and inter-layer relationships. Similarly, a width generation network is used to generate the width of each layer. A soft constraint term is then used to connect importance and width. With these techniques, our method can outperform state-of-the-art pruning methods on CIFAR-10 and ImageNet datasets. Our contributions can be summarized as:

1) We aim to disentangle the learning of width and importance for differentiable channel pruning, which is achieved by relaxing the equality constraint derived by the definition of the top-k operation. By relaxing the equality constraint, width and importance can be parameterized independently. We also extend the discrete top-k masks to soft top-k masks with a smoothstep function allowing custom width for soft windows.
2) To improve the learning efficiency, we parameterize the importance of each channel and width of each layer by using neural networks. The importance generation network is used to capture inter-channel and inter-layer relationships. The width generation network shares similar intuition.
3) Extensive experiments on CIFAR-10 and ImageNet show that our method can outperform existing channel pruning methods on ResNets and MobileNetV2/V3.

4.2 Related Works

Recently, model compression has drawn a lot of attention from the community. Weight pruning and structural pruning are two popular directions.

4.2.1 Weight Pruning

Weight pruning eliminates redundant connections without assumptions on the structures of weights. Weight pruning methods can achieve a very high compression rate while they need specially designed sparse matrix libraries to achieve acceleration and compression. As one of the early works, [43] proposes to use L_1 or L_2 magnitude as the criterion to prune weights and connections. SNIP [85] updates the importance of each weight by using gradients from the loss function. Weights with lower importance will be pruned. Lottery ticket hypothesis [22] assumes there exist high-performance sub-networks within the large network at initialization time. Besides one-shot pruning, repeated pruning and fine-tuning can lead to better performance but with larger costs. In rethinking network pruning [104], they challenge the typical model compression process (training, pruning, fine-tuning) and argue that fine-tuning is not necessary. Instead, they show that training the compressed model from scratch with random initialization can obtain better results.

4.2.2 Structural Pruning

One of the previous works [88] in structural pruning uses the sum of the absolute value of kernel weights as the criterion for filter pruning. Instead of directly pruning filters based on magnitude, structural sparsity learning [140] is proposed to prune redundant structures with Group Lasso regularization. On top of structural sparsity, GrOWL regularization is applied to make similar structures share the same weights [148]. One of the problems when using Group Lasso is that weights with small values could still be important, and it is difficult for structures under Group Lasso regularization to achieve exact zero values. As a result, [105] proposes to use explicit L_0 regularization to make weights within structures have exact zero values. Besides using the magnitude of structure weights as a criterion, other methods utilize the scaling factor of batchnorm to achieve structure pruning, since batchnorm [71] is widely used in recent neural network designs [46, 69].

A straightforward way to achieve channel pruning is to make the scaling factor of batchnorm to be sparse [103]. If the scaling factor falls below a certain threshold, the channel will be removed. Structure sparse selection [70] extends the idea of using scaling factors to more structures, like an entire layer. Another line of research formulates pruning as a constrained optimization problem [77, 145, 32, 75, 150, 25], and they use learnable parameters (also called gate parameters) to control each channel. These parameters are differentiable in their setting, which enables an efficient end-to-end optimization process. Though these methods have succeeded in channel pruning, the width of each layer and the importance of each channel are entangled, limiting the search space. Besides using gates, Collaborative channel pruning [119] tries to prune channels by using Taylor expansion. Greedy forward selection [144] is proposed to find good sub-networks, which starts from an empty network and greedily adds important channels from the original network. In Automatic Model Compression (AMC) [50], they use policy gradient to update the policy network. This policy network is then used to generate the width of each layer. MetaPruning [102] uses a hypernet to generate parameters for sub-networks, and evolutionary algorithms are utilized to find the best configuration (width) of sub-networks. Our method can generate width directly like MetaPruning and AMC. In addition, our method can be optimized more efficiently through regular stochastic gradient methods.

4.2.3 Other Methods

Besides weight and channel pruning methods, there are works from other perspectives, including bayesian pruning [111, 115], weight quantization [14, 121], pruning for fairness [151], and knowledge distillation [52].

4.3 Proposed Method

4.3.1 Preliminary

To better describe our proposed approach, necessary notations are introduced first. In a CNN, the feature map of lth layer can be represented by $\mathcal{F}_l \in \Re^{C_l \times W_l \times H_l}$, $l = 1, \ldots, L$, where C_l is the

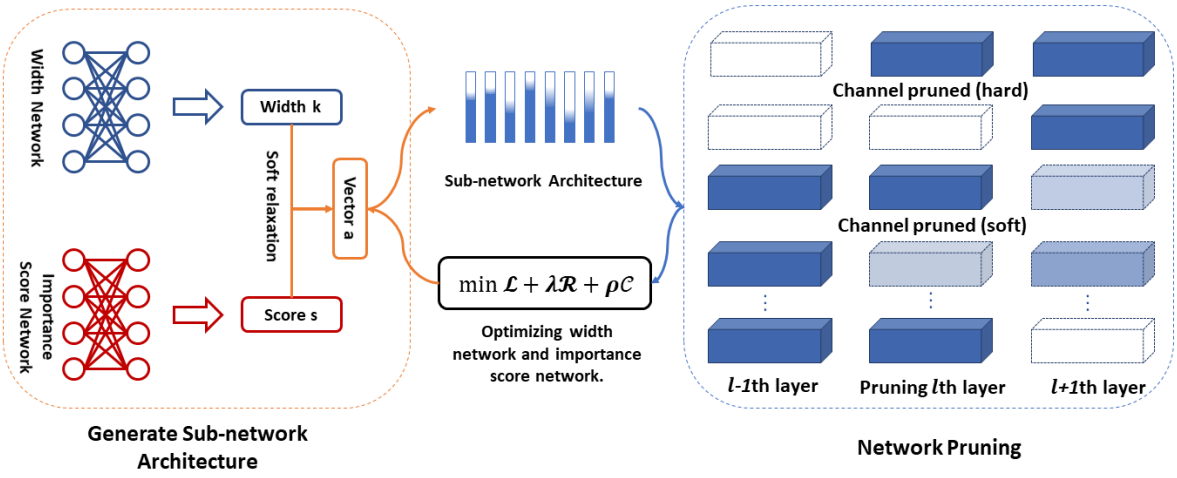


Figure 10: Flowchart of our proposed method. Importance score generator g_s and width generator g_k are used to generate the importance score and width. We then use them to generate the mask vector $\mathbf{a}$, and it is used to produce the sub-network architecture. The network is pruned according to $\mathbf{a}$. Finally, g_k and g_s are optimized by minimizing the loss function.

number of channels, H_l and W_l are height and width of the current feature map, L is the number of layers. The mini-batch dimension of feature maps is ignored to simplify notations. sigmoid$(\cdot)$ is the sigmoid function. round$(\cdot)$ rounds inputs to nearest integers. $\mathbb{1}(\cdot)$ is the indicator function.

Usually, differentiable channel pruning algorithms aim to solve the following problem:

$$\min_{\Theta} \ \mathcal{J}(\Theta) = \mathcal{L}(f(x;\Theta,\mathcal{W}),y) + \lambda\mathcal{R}(\Theta), \tag{4-1}$$

where x, y are input samples and their labels. $\mathcal{W}$, $\mathcal{L}$, and $\mathcal{R}$ are model weights, loss functions, and regularization functions on parameters or FLOPs. Θ are learnable parameters to decide whether to prune the channel. There are many ways to characterize a channel, such as Gumbel-sigmoid approximation [73], shape function [77], and so on. $\mathcal{R}$ is the regularization function to control the number of channels or FLOPs of each layer. Our method aims to disentangle the learning of width and channel importance. As a result, Θ will be reparameterized into two variables: importance scores $\mathbf{s}$ and layer width $\mathbf{k}$. How to achieve such a disentanglement will be detailed in this section.

4.3.2 Top-k Operation

In this section, we will introduce how to parameterize the width of a layer. Let us denote the importance score vector for each channel of a layer as $\mathbf{s} = [s_1, \ldots, s_{C_l}]$. Suppose we need to select k most importance channels out of C_l channels, we can use a top-k mask vector $\mathbf{a}$, which is given by:

$$a_i = \begin{cases} 1 & \text{if } s_i \text{ is a top-}k \text{ element in } S, \\ 0 & \text{otherwise.} \end{cases} \tag{4-2}$$

The process of selecting top-k channels is a natural way for channel pruning, and k represents the width of this layer. The relationship between k and a_i can be represented by the following equation:

$$k = \sum_{i=1}^{C_l} a_i. \tag{4-3}$$

Gradients with respect to k through Eq. 6-2 are not defined. Except Eq. 6-2, we can use an alternative surrogate to represent k:

$$k = \frac{1}{C_l} \sum_{i=1}^{C_l} \underset{s_i > s_0}{1}(s_i), \tag{4-4}$$

where s_0 is a value between kth and $k+1$th value, and the indicator function $\underset{s_i > s_0}{1}(\cdot)$ returns 1 if $s_i > s_0$, otherwise it returns 0. Here, to unify the learning of different layers, we abuse the notation k to represent the normalized version of $k \in [0, 1]$. If we enforce the hard equality defined in Eq. 6-3, it still entangles the learning of importance and width. We then replace it with a regularization term:

$$\mathcal{C}(k, \mathbf{s}) = \| k - \frac{1}{C_l} \sum_{i=1}^{C_l} \text{sigmoid}((\bar{s}_i - \bar{s}_0)/t) \|^2, \tag{4-5}$$

which does not enforce a hard constraint and $\bar{s}$ is the unnormalized importance score (outputs before the final activation of g_s defined in Eq. 4-8). We also relax the indicator function with the sigmoid function of temperature t to facilitate gradient calculations. In practice, we let $\bar{s}_0 = 0$, so that the importance score will match k automatically. The gradients with respect to k can be obtained by utilizing this regularization term, and the width of each layer can be optimized using

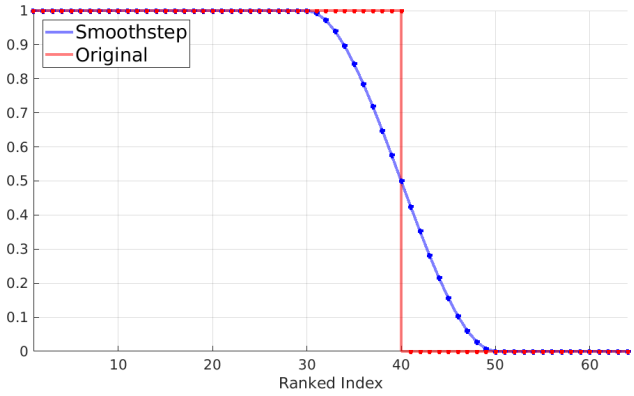


Figure 11: Soft relaxation vs. naive binary mask. In this figure, we choose $C_l = 64$, $\gamma = 20$, $C_l k = 40$. Solid line represent the continuous function. Square dots represent the actual values taken by the vector $\tilde{\mathbf{a}}$.

SGD or other stochastic optimizers. Finally, we achieve pruning by inserting the vector $\mathbf{a}$ to the feature map $\mathcal{F}_l$:

$$\hat{\mathcal{F}}_l = \mathbf{a} \odot \mathcal{F}_l, \tag{4-6}$$

where $\hat{\mathcal{F}}_l$ is the pruned feature map, $\odot$ is the element-wise product, and $\mathbf{a}$ is first resized to have the same size of $\mathcal{F}_l$.

4.3.3 Soft Top-k Operation with Smoothstep Function

When performing the discrete top-k operation, we place 1 to the first k elements of $\tilde{\mathbf{a}}$. Similarly, we use a smoothstep function [45] to generate values for soft relaxed $\tilde{\mathbf{a}}$:

$$\text{Smoothstep}(x) = \begin{cases} 0, & \text{if } x \leq -\gamma/2 + C_l k, \\ -\frac{2}{\gamma^3}(x - C_l k)^3 + \frac{3}{2\gamma}(x - C_l k) + \frac{1}{2}, & \\ \quad \text{if } -\gamma/2 + C_l k \leq x \leq \gamma/2 + C_l k, & \\ 1, & \text{if } x \geq \gamma/2 + C_l k. \end{cases} \tag{4-7}$$

In smoothstep function, γ controls the width of the soft relaxation. $C_l k$ represents the center of the soft relaxation. Outside $[-\gamma/2 + C_l k, \gamma/2 + C_l k]$, smoothstep function performs binary rounding.

We provide the comparison between smoothstep function and naive binary masks in Fig. 11. The value of $\tilde{\mathbf{a}}_i = \text{Smoothstep}(i)$. The soft version of $\mathbf{a}$ can be obtained by $\mathbf{a} = P^T\tilde{\mathbf{a}}$. Here, we abuse the notation of $\tilde{\mathbf{a}}$ and $\mathbf{a}$ for the soft relaxed mask vector.

To satisfy Eq. 6-2, we need $C_l k \approx \sum a_i$. As a result, the center of the soft window should be at $C_l k$. Other functions like sigmoid$(\cdot)$ can also interpolate between $[0, 1]$. We chose the smoothstep function since it provides an easy way to control the width of soft relaxation. If $C_l k$ is close to C_l (when k is close to 1), the soft range of $\tilde{\mathbf{a}}$ is not symmetric anymore on k. We adjust γ to round$(C_l - C_l k)$ to ensure $C_l k \approx \sum a_i$.

Binary values are often used to control the opening or closing of a channel. However, it is better to use soft relaxed values in certain circumstances. We apply soft relaxation on the mask vector $\mathbf{a}$ for several reasons. In practice, it is hard for us to generate k with discrete values, and discrete constraints on kC_l dramatically increase the difficulty of optimization. Thus, the generated k is within $[0, 1]$. If only binary values are used, then $kC_l = 9.1$ and $kC_l = 8.5$ will produce the same $\mathbf{a}$. Soft relaxation can produce unique $\mathbf{a}$ when $kC_l = 9.1$ or $kC_l = 8.5$. Another benefit of soft relaxation is that we can evaluate more channels compared to the discrete settings. Let us first reindex the vector $\mathbf{s}$ as $\tilde{\mathbf{s}}$ based on the monotone decreasing order of $\mathbf{s}$, then $\tilde{\mathbf{s}} = P\mathbf{s}$, where P is a permutation matrix. Since $\mathbf{a}$ and $\mathbf{s}$ have one-to-one correspondence, sorting $\mathbf{a}$ according to $\mathbf{s}$ can be represented as $\tilde{\mathbf{a}} = P\mathbf{a}$.

4.3.4 Generating Width and Importance Score

To provide importance score $\mathbf{s}$ for each channel, we use a neural network g_s to learn it from the dataset:

$$\mathcal{S} = g_s(x_s, \Theta_s), \tag{4-8}$$

where $\mathcal{S} = (\mathbf{s}_1, \cdots, \mathbf{s}_L)$ is the collection of all scores across different layers, Θ_s are learnable parameters of g_s, and x_s is the input of g_s. Before training, we generate x_s randomly, and it is kept fixed during training. We can also use a learnable x_s, which results in similar performance. Previous pruning methods often use a single parameter to control each channel, which can not obtain inter-channel and inter-layer relationships. As a result, g_s is designed to be composed with GRU [10] and fully connected layers. Basically, we use GRU to capture inter-layer relationships,

Algorithm 3: Disentangled Differentiable Network Pruning

Input: D, p, λ, ρ, E, f,
Freeze $\mathcal{W}$ in f.
Initialization: initialize x_s and x_k for g_s and g_k; randomly initialize Θ_s and Θ_k
for $e := 1$ *to* E **do**
  shuffle(D)
  for *a mini-batch* (x, y) *in* D **do**
    1. generate the width of each layer $\mathbf{k}$ from g_k by using Eq. 4-9
    2. generate the importance score of each layer $\mathcal{S}$ from g_s using Eq. 4-8.
    3. produce the soft mask vector $\tilde{\mathbf{a}}$ with Eq. 4-7, and obtain $\mathbf{a} = P^T\tilde{\mathbf{a}}$
    4. calculate gradients for Θ_s : $\frac{\partial J}{\partial \Theta_s} = \frac{\partial \mathcal{L}}{\partial \Theta_s} + \rho\frac{\partial \mathcal{C}}{\partial \Theta_s}$ and Θ_k : $\frac{\partial J}{\partial \Theta_k} = \lambda\frac{\partial \mathcal{R}}{\partial \Theta_k} + \rho\frac{\partial \mathcal{C}}{\partial \Theta_k}$ separately.
    5. update Θ_k and Θ_s with ADAM.
  end
end
Pruning the model with resulting g_k and g_s, and finetune it.

and fully connected layers are for inter-channel relationships. The additional computational costs introduced by g_s is trivial, and it has little impact on the training time. Since $\mathcal{S}$ is not directly involved in the forward computation, we use straight-through gradient estimator [2] to calculate the gradients of it: $\frac{\partial \mathcal{J}}{\partial \mathbf{s}} = \frac{\partial \mathcal{J}}{\partial \mathbf{a}}$. We also want to emphasize that it's crucial to use simgoid$(\cdot)$ as the output activation for g_s. Using absolute values [127] or other functions incurs much larger errors when estimating the gradients. This is probably because simgoid$(\cdot)$ better approximates binary values.

We also use a neural network g_k to generate the width for each layer:

$$\mathbf{k} = g_k(x_k, \Theta_k), \tag{4-9}$$

where x_k is the input to g_k, Θ_k are parameters for g_k, and $\mathbf{k} = [k_1, \cdots, k_L]$ is a vector contains width of all layers. The output activate function is the sigmoid function again, since we need to restrict the range of $k \in [0, 1]$. g_k is composed of fully connected layers. In addition, like x_s, x_k is also generated randomly, and it is kept fixed when training g_k.

4.3.5 The Proposed Algorithm

With techniques introduced in previous sections, we can start to prune the network. The network pruning problem in our setting can be formulated as the following problem:

$$\begin{aligned}\min_{\Theta_k,\Theta_s} \mathcal{J}(\Theta_k,\Theta_s) =&\{\mathcal{L}(f(x;\mathcal{A},\mathcal{W}),y) + \lambda\mathcal{R}(T(\mathbf{k}),pT_{\text{total}}) \\ &+ \rho\mathcal{C}(\mathbf{k},\mathcal{S})\}\end{aligned} \tag{4-10}$$

where (x, y) is the input sample and its corresponding label, $f(x;\mathcal{A},\mathcal{W})$ is a CNN parameterized by $\mathcal{W}$ and controlled by $\mathcal{A} = [\mathbf{a}_1, \cdots, \mathbf{a}_L]$, $\mathcal{R}$ is the FLOPs regularization term, $T(\mathbf{k})$ is the FLOPs of the current sub-network, p is the pruning rate, T_{total} is the total prunable FLOPs, $\mathcal{J}$ is the overall objective function, $\mathcal{C}(\mathbf{k},\mathcal{S})$ is defined in Eq. 6-4, and $\rho,\ \lambda$ are hyper-parameters for $\mathcal{C},\ \mathcal{R}$ separately. We let $\mathcal{R}(x, y) = \log(\max(x, y)/y)$, which can quickly push $\mathcal{R}$ to approach 0. Our objective function defined in Eq. 4-10 can be optimized efficiently by using any stochastic gradient optimizer. Using learnable importance scores produces quite strong empirical performance. If a better learning mechanism for importance score is designed, it can also be merged into our algorithm.

The overall algorithm is given in Algorithm 3. The input of Algorithm 3 are D: a dataset for pruning, p: the pruning rate defined in Eq. 4-10, λ and ρ: hyper-parameter for $\mathcal{R}$ and $\mathcal{C}$, f: a neural network to be pruned and E: the number of pruning epochs. In order to facilitate pruning, we usually choose D as a subset of the full training set. In step 4 of Algorithm 3, the gradients of Θ_k and Θ_s are calculated separately because of $\mathcal{C}$. This operation brings marginal computational burden since $\mathcal{C}$ and $\mathcal{R}$ are not depend on input samples. The fine-tuning process is very time-consuming. As a result, we use the performance of a sub-network within the pre-trained model to represent its quality. This setup is used in many pruning methods, like AMC [50], and we freeze weights $\mathcal{W}$ during pruning. When performing actual pruning, we round $C_l k_l$ to its nearest integer, and soft relaxation is not used. Instead, we use Eq. 6-1 to generate $\mathbf{a}$, which ensures that $\mathbf{a} \in \{0, 1\}$. Like previous differentiable pruning works, our method can be directly applied to pre-trained CNNs, which are flexible to use. The overall flowchart of our method is shown in Fig. 10.

4.4 Connections to Previous Works

In this section, we will discuss the differences and connections of our methods compared to previous works. To connect our method with previous work, we can use an equality constraint to replace the regularization term in Eq. 4-10:

$$\min_{\mathbf{k},\mathcal{S}} \mathcal{J}(\mathbf{k},\mathcal{S}) = \mathcal{L}(f(x;\mathbf{a},\mathcal{W}),y) + \lambda\mathcal{R}(T(\mathbf{k}),pT_{\text{total}}),$$
$$s.t.\ k_l = \frac{1}{C_l}\sum_{i=1}^{C_l} \underset{s_i^l>0.5}{1}(s_i^l),\ \ l=1,\ \cdots,\ L. \tag{4-11}$$

Here, we do not use g_k and g_s to simplify the analysis, and we also let $s_0^l = 0.5$ since we use sigmoid activation functions for g_s. Eq. 4-11 is closely related to Eq. 4-1. If we insert $\frac{1}{C_l}\sum_{i=1}^{C_l} \underset{s_i^l>0.5}{1}(s_i^l)$ to every k_l in $T(\mathbf{k})$, we almost recover Eq. 4-1. Compared to Eq. 4-10, Eq. 4-11 is more restrictive since it reduces the number of parameters for pruning one layer from $C_l + 1$ to C_l, which is equivalent to saying that disentangled pruning provides an extra degree of freedom compared to previous works. This may explain why using independent parameterization for importance and width achieves better empirical performance than previous works. Also note that Eq. 4-11 corresponds to set ρ to ∞ in Eq. 4-10, and $\mathbf{k}$ is no longer a validate variable. If we let $\rho = 0$, we have completely disentangled $\mathbf{k}$ and $\mathcal{S}$. But in this situation, the resulting $\mathbf{k}$ will be a trivial solution because it only depends on $\mathcal{R}$. From this perspective, the proposed method in Eq. 4-10 actually interpolates between previous differentiable pruning works and the complete disentangled formulation.

4.5 Experiments

4.5.1 Settings

Similar to many model compression works, CIFAR-10 [82] and ImageNet [15] are used to evaluate the performance of our method. Our method uses p to control the FLOPs budget. The detailed choices of p are listed in the supplementary materials. The architectures of g_s and g_k are

Table 7: Comparison results on CIFAR-10 dataset with ResNet-56 and MobileNetV2. Δ-Acc represents the performance changes before and after model pruning. +/- indicates increase or decrease compared to baseline results.

Method	Architecture	Baseline Acc	Pruned Acc	Δ-Acc	$\downarrow$ FLOPs
AMC [50]	ResNet-56	92.80%	91.90%	-0.90%	50.0%
DCP [154]		93.80%	93.81%	$+0.01\%$	47.0%
CCP [119]		93.50%	93.42%	-0.08%	52.6%
HRank [96]		93.26%	93.17%	-0.09%	50.0%
LeGR [9]		93.90%	93.70%	-0.20%	**53.0%**
DDNP (ours)		93.62%	93.83%	**+0.21%**	51.0%
Uniform [154]	MobileNetV2	94.47%	94.17%	-0.30%	26.0%
DCP [154]		94.47%	94.69%	$+0.22\%$	26.0%
MDP [39]		95.02%	95.14%	$+0.12\%$	28.7%
SCOP [132]		94.48%	94.24%	-0.24%	40.3%
DDNP (ours)		94.58%	94.81%	**+0.23%**	**43.0%**

also provided in supplementary materials. γ in Eq. 4-7 is chosen as round$(0.1C_l)$. γ then depends on layer width C_l, and it empirically works well.

Within the experiment section, our method is called DDNP (***D***isentangled ***D***ifferentiable for ***N***etwork ***P***runing). For CIFAR-10, we compare with other methods on ResNet-56 and MobileNetV2. For ImageNet, we select ResNet-34, ResNet-50, MobileNetV2 and MobileNetV3 small as our target models. The reason we choose these models is because that ResNet [46], MobileNetV2 [126] and MobileNetV3 [53] are much harder to prune than earlier models like AlexNet [83] and VGG [130].

λ decides the regularization strength in our method. We choose $\lambda = 2$ in all CIFAR-10 experiments and $\lambda = 4$ for all ImageNet experiments. We choose $\rho = 2$ and $t = 0.4$ for both datasets. For CIFAR-10 models, we train ResNet-56 and MobileNet-V2 from scratch following PyTorch examples. After pruning, we finetune the model for 160 epochs using SGD with a start learning rate of 0.1, weight decay 0.0001, and momentum 0.9. For ImageNet models, we directly use the pre-trained models released from pytorch [118]. After pruning, we finetune the model for 100 epochs using SGD with an initial learning rate of 0.1, weight decay 0.0001, and momentum 0.9.

For MobileNetV2 on ImageNet, we choose weight decay as 0.00004 and use an initial learning rate of 0.05 with cosine annealing learning rate scheduler, which is the same as the original paper [126]. Most settings for MobileNetV3 small are the same as MobileNetV2. The difference is that weight decay is reduced to 0.00001 following the original setting [53].

For training g_k and g_s, we use ADAM [79] optimizer with a constant learning rate of 0.001 and train them for 200 epochs. We start pruning from the whole network. To achieve this, we add a constant bias to the sigmoid function in g_k, and we set it to 3.0. We randomly sample $2,500$ and $25,000$ samples from CIFAR-10 and ImageNet, and they are used as the pruning subset D in Algorithm 3. In the experiments, we found that a separate validation set is not necessary. All samples in D come from the original training set. All codes are implemented with pytorch [118].

4.5.2 CIFAR-10

We present comparison results on CIFAR-10 in Tab. 7. On ResNet-56, our method achieves the largest performance gain ($+0.21\%$ Δ-Acc) compared to other baselines. All methods prune around 50% of FLOPs, and LeGR has the largest pruning rate. At this pruning rate, our method has obvious advantages compared to other methods. Specifically, our method is 0.20% better than DCP regarding Δ-Acc. Although DCP has the second best Δ-Acc, it has the lowest FLOPs reduction rate. CCP and HRank have similar pruning rates and performance, and our method outperforms them by around 0.30% in terms of Δ-Acc. LeGR prunes more FLOPs than our method, but it has a much lower Δ-Acc (-0.20% vs. $+0.21\%$).

For MobileNetV2, our method achieves the best Δ-Acc and prunes most FLOPs ($+0.23\%$ Δ-Acc and 43% FLOPs). SCOP prunes slightly less FLOPs, and the performance of SCOP is also lower than our method (-0.24% vs. $+0.23\%$ regarding Δ-Acc). Our method and DCP have similar performance, but our method prunes 17% more FLOPs. In summary, the CIFAR-10 results demonstrate that our method is an effective way for network pruning.

4.5.3 ImageNet Results

The ImageNet results are given in Tab. 8. We present both base and pruned Top-1/Top-5 accuracy in the table.

ResNet-34. Our method achieves the best Δ Top-1 and Δ Top-5 accuracy with ResNet-34. IE performs the second best regarding Δ Top-1/Δ Top-5, but it prunes much less FLOPs compared to other baselines. SCOP, FPGM, IE, and our method have similar pruning rates. SCOP has the largest FLOPs reduction rate, but the FLOPs gap between our method and SCOP is quite marginal (only 0.6%). Given similar pruning rates, our method outperforms other baselines by at least 0.41% in terms of Δ Top-1 accuracy.

ResNet-50. For ResNet-50, our method achieves the best pruned Top-1/Top-5 accuracy, and the reduction of FLOPs is also obvious. DCP prunes most FLOPs among all comparison baselines. Our method is 0.84% better than DCP regarding Δ Top-1 accuracy while only removing 0.6% less FLOPs than it. The gap between GBN and CC is around 0.09%, and they outperform other baselines. Our method further improves the result of GBN and CC by 0.43% and 0.32% with Δ Top-1 accuracy. CC has the second best performance, but our method prunes 2% more FLOPs than it. Notably, our method achieves no loss on Top-5 accuracy ($+0.02\%$). Also notice that CC considers both channel pruning and weight decomposition, introducing extra performance efficiency trade-off.

MobileNetV2. MoibleNetV2 is a computationally efficient model by design that is harder to prune than ResNets. With this architecture, all methods prune similar FLOPs within ranges between 29.5% to 30.7%. Our method obtains $72.20\%/90.51\%$ Top-1/Top-5 accuracy after pruning, and both of them are better than all the other methods. Compared to the second best method GFS [144], the Top-1/Δ Top-1 accuracy of our method is $0.60\%/0.65\%$ higher than it. MetaPruning prunes most FLOPs, but the performance is lower than our method by a large margin. AGMC improves the results of AMC, but the improvement is not very significant.

MobileNetV3 small. MobileNetV3 small is a tiny model with FLOPs of around 64M. The uniform baseline prunes most FLOPs which is 3.1% and 2.1% higher than GFS and our method, but the absolute FLOPs difference is small (Uniform: 47M; GFS: 49M; Ours: 48.3M). Our method has significant advantages on MobileNet-V3 small, and it is $1.23\%/1.06\%$ better than GFS on Top-1/Δ Top-1 accuracy. GFS greedily selects neurons with the largest impact on the loss starting from an empty set, and it performs well across multiple architectures. They argue that forward selection is better than backward elimination with greedy selection. On the contrary, in our setting, disentangled pruning can successfully discover good sub-networks starting from the whole model,

especially for compact architectures. The superior performance of our method demonstrates the advantage of disentangled pruning.

4.5.4 Impacts of Different Settings

In this section, we will demonstrate the effectiveness of different design choices.

We first build a differentiable pruning (DP) baseline by using the Gumbel-sigmoid function. We then compare DP with our method in Fig. 13 (a,e). Our method outperforms DP when $p = 0.5$ and $p = 0.35$. The advantage becomes obvious when the pruning rate is large ($p = 0.35$). This observation suggests that our method can discover better sub-networks than DP across different pruning rates. We also visualize the layer-wise width in Fig. 12. An interesting observation is that, with different p, the relative order of width changes (like the first and second blocks) with our method.

In Fig. 13 (b,f), we verify the effectiveness of soft top-k defined in section 4.3.3. The hard setting refers to set $\gamma = 0$ in Eq. 4-7. From the figure, we can see that soft top-k operation achieves better performance than the hard version. Moreover, when the pruning rate is large, the effect of soft top-k becomes more clear (gap around 5%). These results suggest soft top-k is preferred when disentangling the learning of width and importance.

In Fig. 13 (c,g), we present results by varying the architecture of g_s. We can see that full g_s (GRU+FC) has the best performance, followed by FC and channel-wise importance score. The learning of importance may become difficult when we use disentangled pruning (probably due to gradient calculations with STE), and naive parametrization (one parameter per channel) lacks enough capacity to efficiently capture inter-channel and inter-layer relationships. Using a model with a larger capacity enables fast learning.

Finally, we compare different output functions of g_s in Fig. 13. We compare three different output functions: sigmoid function, absolute function, and square function. Recall that we use s and $\bar{s}$ to represent the importance score and unnormalized importance score (outputs before the final activation of g_s). As a result, the importance score with sigmoid function, absolute function, and square function is defined as $s = \text{sigmoid}(\bar{s})$, $s^{AF} = \mid \bar{s} \mid$ and $s^{SF} = \bar{s}^2$. From Fig. 13, it is clear that sigmoid$(\cdot)$ has the best performance, which indicates that better alignment in the forward

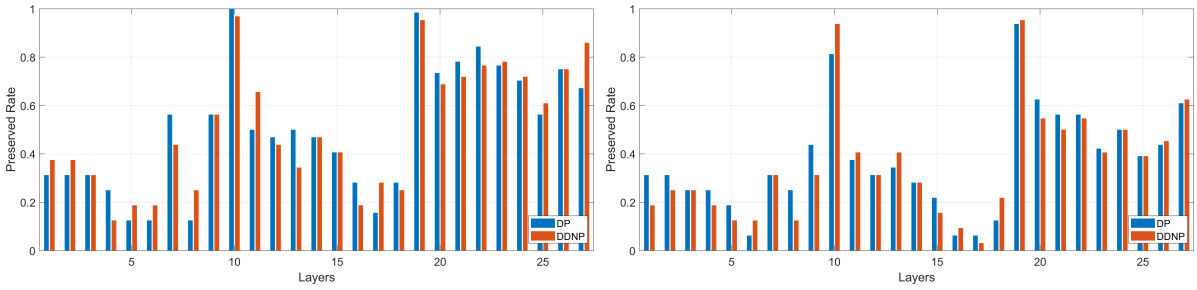
pass helps improve the quality of gradients when learning importance scores.

4.6 Conclusion

In previous differentiable pruning works, width and channel importance are entangled during the pruning process. Such a design is straightforward and easy to use, but it restricts the potential search space during the pruning process. To overcome this limitation, we propose to prune neural networks by disentangling width and importance. To achieve such a disentanglement, we propose to relax the hard constraint used in previous methods to a soft regularization term, allowing independent parametrization of width and importance. We also relax hard top-k to soft top-k with the smoothstep function. We further use an importance score generation network and a width network to facilitate the learning process. Moreover, the design choices are empirically verified for our method. The experimental results on CIFAR-10 and ImageNet demonstrate the strength of our method.

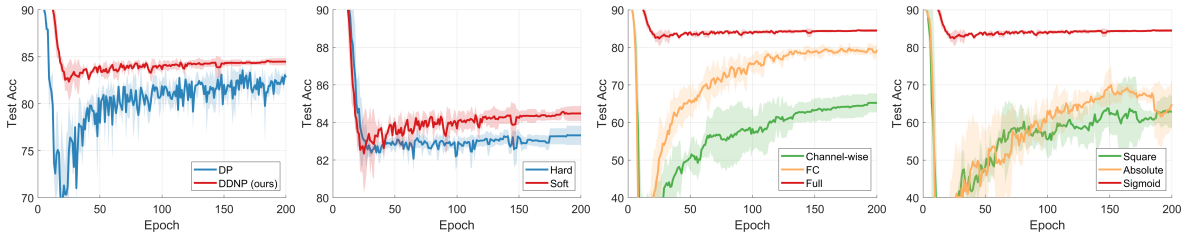
Table 8: Comparison results on ImageNet dataset with ResNet-34, ResNet-50, ResNet-101 and MobileNetV2. Δ-Acc represents the performance changes before and after model pruning. +/- indicates an increase or decrease compared to baseline results.

Method	Architecture	Base/Pruned Top-1	Base/Pruned Top-5	Δ Top-1	Δ Top-5	$\downarrow$ FLOPs
SFP [48]	ResNet-34	73.93%/71.84%	91.62%/89.70%	-2.09%	-1.92%	41.1%
IE [112]		73.31%/72.83%	-/-	-0.48%	-	24.2%
FPGM [49]		73.92%/72.63%	91.62%/91.08%	-1.29%	-0.54%	41.1%
SCOP [132]		73.31%/72.62%	91.42%/90.98%	-0.69%	-0.44%	**44.8%**
DDNP (ours)		73.31%/**73.03%**	91.42%/**91.23%**	**-0.28%**	**-0.19%**	44.2%
DCP [154]	ResNet-50	76.01%/74.95%	92.93%/92.32%	-1.06%	-0.61%	55.6%
CCP [119]		76.15%/75.21%	92.87%/92.42%	-0.94%	-0.45%	54.1%
MetaPruning [102]		76.60%/75.40%	-/-	-1.20%	-	51.2%
GBN [145]		75.85%/75.18%	92.67%/92.41%	-0.67%	-0.26%	55.1%
HRank [96]		76.15%/74.98%	92.87%/92.33%	-1.17%	-0.54%	43.8%
LeGR [9]		76.10%/75.30%	-/-	-0.80%	-	54.0%
SCOP [132]		76.15%/75.26%	92.87%/92.53%	-0.89%	-0.34%	54.6%
GReg [135]		76.13%/75.36%	-/-	-0.77%	-	**56.7%**
SRR [138]		76.13%/75.11%	92.86%/92.35%	-1.02%	-0.51%	55.1%
CC [92]		76.15%/75.59%	92.87%/92.64%	-0.56%	-0.13%	52.9%
DDNP (ours)		76.13%/**75.89%**	92.86%/**92.90%**	**-0.24%**	**$+$ 0.04%**	55.0%
Uniform [126]	MobileNetV2	71.80%/69.80%	91.00%/89.60%	-2.00%	-1.40%	30.0%
AMC [50]		71.80%/70.80%	-/-	-1.00%	-	30.0%
AGMC [146]		71.80%/70.87%	-/-	-0.93%	-	30.0%
MetaPruning [102]		72.00%/71.20%	-/-	-0.80%	-	**30.7%**
GFS [144]		72.00%/71.60%	-/-	-0.40%	-	30.0%
DDNP (ours)		72.05%/**72.20%**	90.39%/**90.51%**	**$+0.15\%$**	**$+0.12\%$**	29.5%
Uniform [53]	MobileNetV3 small	67.50%/65.40%	-/-	-2.10%	-	**26.6%**
GFS [144]		67.50%/65.80%	-/-	-1.70%	-	23.5%
DDNP (ours)		67.67%/**67.03%**	87.40%/**86.94%**	**-0.64%**	**-0.46%**	24.5%

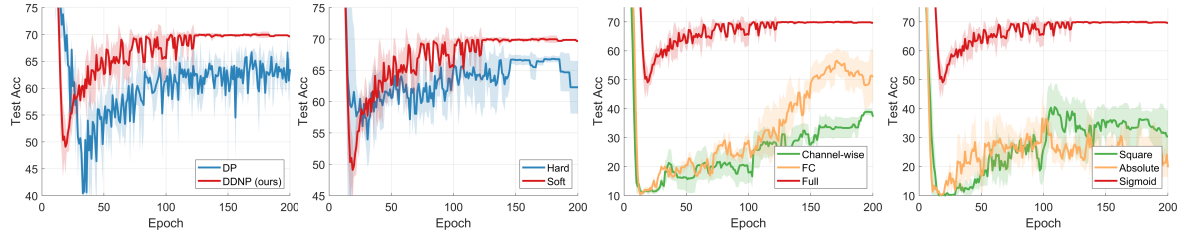


(a) (b)

Figure 12: (a) and (b): Layer-wise width for two different pruning rates: $p = 0.5/0.35$. We compare DDNP with differentiable pruning (DP) in both figures.



(a) DP baseline ($p = 0.5$) (b) Soft/Hard ($p = 0.5$) (c) g_s architecture ($p = 0.5$) (d) Output functions ($p = 0.5$)



(e) DP baseline ($p = 0.35$) (f) Soft/Hard ($p = 0.35$) (g) g_s architecture ($p = 0.35$) (h) Output functions ($p = 0.35$)

Figure 13: (a,e): Comparisons of our method and the diferentiable pruning (DP) baseline. (b,f): Comparisons of the soft and hard settings for the top-k operation. (c,g): Performance during pruning when using different architectures of g_s. (d,h): Performance during pruning when using different output functions of g_s. We run each setting three times and use shaded areas to represent variance. All experiments are done on CIFAR-10 with ResNet-56.

5.0 Structural Alignment for Network Pruning through Partial Regularization

5.1 Background

Convolutional Neural Networks (CNNs) have achieved many successes in different computer vision tasks [83, 122, 123, 129, 3, 19, 40]. To tackle real-world challenges, recent CNNs [83, 130, 46] become larger and larger regarding width, depth, etc. With such capacities, CNNs can obtain better performance on different benchmarks at the cost of computational and storage burdens. At the same time, with the recent developments of mobile and embedded devices, the demand for deploying CNNs on these devices has increased dramatically. However, there is a natural conflict between the size of CNNs and the hardware capability of these devices. To overcome these challenges, many works [43, 42] try to reduce the size of CNNs, and make them possible to be deployed on edge devices.

There are many directions to reduce the size of CNNs. Among them, weight pruning and structural pruning are two popular topics. Structural pruning, especially channel pruning, is more friendly to hardware than weight pruning since no post-processing steps are required to acquire savings in computational and storage costs. Thus, our paper focuses on channel pruning for CNNs. Many existing one-shot pruning works [119, 145, 32, 112, 50, 30] prune trained models directly. No matter what method is used, there will be a significant gap between the selected sub-network and the pruned model. Such a gap creates difficulties in regaining performance during the fine-tuning process. On the other hand, soft pruning methods [48, 75] softly remove structures during the training process, which can produce good results with a shorter fine-tuning process. However, soft pruning methods generally perform worse than typical one-shot pruning methods, probably because the weight space is restricted during the training process because of soft pruning.

To tackle the above problems, we introduce a novel partial regularization technique to align model weights and the discovered sub-network during the training process, which can produce a high performance sub-network and reduce the gap between the sub-network and the original model. In addition, unlike soft-pruning methods, all model structures are used for training. The partial regularization term contains a partial group lasso regularization on selected weights, and

other weights remain intact without modifications. An architecture generator is trained to select which weights should be aligned, and it is also updated during the training process. Our partial regularization formulation is related to partial regularization in lasso [106]. Inspired by the nonmonotone proximal gradient (NPG) method used in [106], we also use a proximal gradient method to solve the partial regularization problem in our setting. Note that our method dynamically changes which channels should be put in the partial regularization. As a result, we add a scalar to balance the regularization strength for different layers because the number of pruned channels is different for them. To maximally keep the capacity of the original model, we insert the partial regularization in the middle of the training process. This is because weights are vulnerable to pruning at the early training stage, and the FLOPs regularization will dominate updates of the architecture generator, which can create bad sub-networks and mislead the training process. Finally, we update model weights and the architecture generator periodically, and they are connected by the partial regularization term during training. To maintain similar training efficiency as the original model, we only use a small portion of samples to train the architecture generator. Thus, there is only a small overhead compared to the original training process. Our method successfully finds performant sub-networks from the original model with these techniques. In summary, the contributions of this paper can be summarized as follows:

1) We propose to align the sub-network in the original model with the final pruned model through partial regularization. By structural alignment, the gap between the selected sub-network and the pruned model is largely reduced, which naturally improves the performance of the pruned model.
2) We use an architecture generator parameterized by neural networks to select the proper sub-network structure and guide the partial regularization. Inspired by partial regularization in lasso [106], we propose to solve the partial regularization problem via proximal gradients, which facilitate the alignment process.
3) Empirical results show that the sub-network discovered by our method performs much better than the one-shot pruning setting. Extensive experiments on CIFAR-10 and ImageNet show that our method outperforms existing channel pruning methods on ResNets and MobileNet-V2.

5.2 Related Works

Weight-Level Pruning. Weight pruning removes redundant connections based on individual weights. High compression rates can be achieved by weight pruning, but they can not directly achieve acceleration, and specially designed sparse matrix libraries are required. One of the early works [43] proposes to measure the importance of weights with their L_1 or L_2 magnitude, and unimportant weights are removed. Instead of magnitude, SNIP [85] updates the importance of each weight by using gradients from the loss function. SNIP can be used at the initialization time. The assumption of the Lottery ticket hypothesis [22] suggests that there exist sparse sub-networks (winning tickets) that can achieve the performance of the full model. In addition, with repeated training and fine-tuning, it can achieve better results. On the other hand, rethinking network pruning [104] argues that the learned topology from pruning algorithms is the key to achieving better performance. In addition, weight rewinding [124] shows that resetting weights to values from the middle training process can also produce good results. Similar to weight rewinding, our method does not modify weight training at the beginning, the partial regularization is inserted in the middle training process.

Structural-Level Pruning. Different from weight pruning, structural pruning is more friendly to hardware since it requires little or no post-processing steps to achieve acceleration. Similar to weight pruning, one of the early structural pruning methods [88] measure the importance of filters by using the sum of the absolute value of kernel weights. Besides using the magnitude to measure channel or filter importance, other methods utilize the scaling factor of batchnorm [71] to indicate which channels are important because batchnorm [71] is popular for the design of recent CNNs [46, 126]. To prune channels, Liu *et al.* [103] apply the sparse regularization to the scaling factors of batchnorm, and the channel will be pruned if the corresponding scaling factor is small. Structure sparse selection [70] introduces scaling factor to specific structures, such as neurons, groups, or residual blocks, and the sparsity regularization is applied to these structures. Structures with small values will also be removed. Another line of research formulates channel pruning as a constrained optimization problem [77, 145, 32, 75, 150], and learnable parameters are used to control each channel. These parameters are end-to-end differentiable, which is amenable to gradient based optimization methods. Our method is also related to these researches. Different from these

methods, the learning of sub-networks accomplishes the training of model weights in our method. In addition, we use partial regularization to promote the selected sub-network, which reduces the gap between the pruned model and the sub-network. Besides using gates, Automatic Model Compression (AMC) [50] uses policy gradient to update the policy network. This policy network is then used to decide the left width of each layer. Collaborative channel pruning [119] prunes channels by exploiting inter-channel dependency. Greedy forward selection [144] starts from an empty network and greedily adds Important channels from the full model. MetaPruning [102, 91] uses a hypernet to generate parameters for sub-networks, and evolutionary algorithms are utilized to find the best sub-networks. MetaPruning shows that pruning should accomplish trained model weights.

Besides these pruning methods, regularization based pruning methods can also be applied to structural pruning. Previous works [140, 90] use group sparsity to prune different structures. In addition to structural sparsity, structural sharing is considered in other works [148, 35]. Our method relates to regularization based methods; however, the formulation of our partial regularization only aligns selected channels dynamically, and other weights are intact.

Other Related Works. Besides the above-mentioned methods, there are works from other perspectives, including bayesian pruning [111, 115], weight quantization [14, 121], and knowledge distillation [52].

5.3 Proposed Method

5.3.1 Overview

Before introducing our method, we first describe notations and provide an overview of our method. In a CNN, the feature map of lth layer can be represented by $\mathcal{F}_l \in \Re^{C_l \times W_l \times H_l}$, $l = 1, \ldots, L$, where C_l is the number of channels, H_l and W_l are height and width of the current feature map, L is the number of layers. Similarly, the weights of lth layer can be written as $\mathcal{W}_l \in \Re^{C_l \times C_{l-1} \times k_l \times k_l}$, and k_l is the kernel size of this layer. The mini-batch dimension of feature maps is ignored to simplify notations.

The core motivation of our proposed method is to reduce the gap between the selected sub-

network from the original model and the pruned model. To achieve this goal, we need two processes. First, we need to choose the desired sub-network from the original model, and this sub-network should also be updated when model weights are trained during the learning process. Secondly, we use the sub-network to guide the partial regularization term, which is used to decide which weights should be regularized. In addition, partial regularization should not be fixed to accommodate the changes in the selected sub-network.

5.3.2 Learning the Sub-network

We use an **A**rchitecture **G**enerator **N**etwork (AGN) to generate the desired sub-network architecture $\mathbf{v} \in \{0,1\}^N$, where 0 or 1 is used to depict the removal or keep of a channel, and N is the total number of channels from all layers. The large parameterization space of AGN can facilitate the learning of sub-network structures. To generate $\mathbf{v}$, the following equation is used:

$$\mathbf{v} = \text{AGN}(\Theta), \tag{5-1}$$

and AGN is composed of gated recurrent unit (GRU) [11] and dense layers. In addition, Gumbel-Sigmoid [73] with STE [2] are used to produce the final binary vector $\mathbf{v}$, and they are placed after the output of dense layers. More details of AGN are provided in the supplementary materials.

Once we have $\mathbf{v}$, we can apply it to the feature maps to produce a sub-network. The feature map of the lth layer is then modified as follows:

$$\mathcal{F}\widehat{\mathcal{F}}_l = \mathbf{v}_l \odot \mathcal{F}_l, \tag{5-2}$$

where $\odot$ is element-wise multiplication, $\mathbf{v}_l$ is the architecture vector of lth layer, and $\mathbf{v}_l$ is resized to have the same size of $\mathcal{F}_l$. The feature map $\mathcal{F}_l$ is from the output of the activation function. The overall loss function for generating the desired sub-network is as follows:

$$\min_{\Theta} \mathcal{J}_\theta(\Theta) := \mathcal{L}(f(x; \mathcal{W}, \mathbf{v}), y) + \lambda \mathcal{R}_{\text{FLOPs}}(T(\mathbf{v}), pT_{\text{total}}) \tag{5-3}$$

where $T(\mathbf{v})$ is the current FLOPs decided by the vector $\mathbf{v}$, T_{total} is the total FLOPs of the original model, $p \in (0,1]$ is a hyperparameter deciding the remaining fraction of FLOPs, λ is the hyperparameter controlling the strength of FLOPs regularization, $f(x; \mathcal{W}, \mathbf{v})$ is a CNN parameterized

by $\mathcal{W}$ and the sub-network structure is determined by the architecture vector $\mathbf{v}$, $\mathcal{L}$ is the task loss function and $\mathcal{R}_{\text{FLOPs}}$ is the regularization term for FLOPs. The regularization term $\mathcal{R}_{\text{FLOPs}}$ is $\mathcal{R}_{\text{FLOPs}}(x, y) = \log(\max(x, y)/y)$. With Eq. 6-3, we can find promising sub-networks when training the AGN and model weights periodically.

5.3.3 Partial Regularization

Given a sub-network $\mathbf{v}$ obtained from AGN, we can then reduce the gap between the sub-network and the pruned model and thus increase its performance. We can formulate the optimization problem with partial regularization as follows:

$$\min_{\mathcal{W}} \ \mathcal{J}_w(\mathcal{W}) := \mathcal{L}(f(x; \mathcal{W}), y) + \gamma \mathcal{R}_w(\mathcal{W}), \tag{5-4}$$

where $\mathcal{R}_w$ is the partial regularization term, and γ controls the strength of the partial regularization. $\mathcal{R}_w$ has the following form:

$$R_w(\mathcal{W}) = \sum_{l=1}^{L} \sum_{i \in S_l} \frac{\hat{N}_l}{\hat{N}} \|\mathcal{W}_{l[i,:,:,:]}\|_{\text{GL}}, \tag{5-5}$$

where $\hat{N}_l = \sum 1 - \mathbf{v}_l$, $\hat{N} = \sum 1 - \mathbf{v}$ and $S_l = \{i \mid \text{if } \mathbf{v}_{l[i]} = 0\}$. S_l contains the indices of pruned channels which are decided by AGN. $\frac{\hat{N}_l}{\hat{N}}$ is a scalar to adjust the regularization strength given different layers. The numerator $\hat{N}_l$ is the number of pruned channels of the lth layer, and the denominator $\hat{N}$ is the number of pruned channels from all layers. It is easy to see that $\sum_{l=1}^{L} \frac{\hat{N}_l}{\hat{N}} = 1$. $\|x\|_{\text{GL}}$ is the norm of grouped weights for Group Lasso, and $\|x\|_{\text{GL}} = \sqrt{\sum_{i=1}^{|x|} x_i^2}$ where $|x|$ represents the number of elements in x. In Eq. 6-5, we assume the corresponding layer is pruned across the output dimension. If it is pruned across the input dimension, we have $\sum_{i \in S_l} \|\mathcal{W}_{l[:,i,:,:]}\|_{\text{GL}}$.

The goal of using Group Lasso for our partial regularization is to reduce the distance between the dense model and the pruned model, other suitable functions may also be useful, but we found that the partial regularization with Group Lasso already produces good results. Another benefit of the partial regularization $\mathcal{R}_w$ is that it will not penalize weights that are not pruned. By doing so, we can avoid the problem of overpenalizing all weights' magnitude. In addition, $\mathbf{v}$ is updated during the training process, and $\mathcal{R}_w$ will dynamically regularize weights. As a result, $\mathbf{v}$ can flexibly change instead of falling into a fixed sub-network during the optimization process.

Algorithm 4: Structural Alignment for Network Pruning

Input: D, D_{AGN} ,p, λ, γ, E, E_{start},
Initialization: initialize $\mathcal{W}$ and θ.
for $e := 1$ *to* E **do**
  /* Optimizing model weights. Freeze Θ of the AGN. */
  for *a mini-batch* (x, y) *in* D **do**
    1. calculate the gradients w.r.t model weights: $\nabla_{\mathcal{W}}\mathcal{L}$.
    2. update model weights using any stochastic optimizer.
    3. **if** *if* $e \geq E_{start}$ **then**
      generate $\mathbf{v}$ from the AGN by using Eq. 6-1.
      apply the proximal gradient step following Eq. 6-8.
  end
  /* Optimizing Θ of the AGN. Freeze model weights $\mathcal{W}$. */
  if *if* $e \geq E_{start}$ **then**
    for *a mini-batch* (x, y) *in* D_{AGN} **do**
      1. generate $\mathbf{v}$ from the AGN by using Eq. 6-1 and apply it to the model.
      2. calculate gradients w.r.t to $\mathcal{J}$ in Eq. 6-3: $\nabla_{\Theta}\mathcal{J}_{\theta}$
      3. update the AGN with ADAM
    end
end
Pruning the model with resulting $\mathbf{v}$, and fine-tuning it.

5.3.4 Proximal Gradients for Partial Regularization

Eq. 6-5 has a similar formulation of the partial regularization of lasso [106]. In [106], the related optimization problem is solved via a nonmonotone proximal gradient (NPG) method. However, NPG requires frequent evaluation of the loss function to ensure the loss value after proximal gradients is less or equal to the loss value before the update. With CNNs, the costs of NPG are too large due to frequent loss evaluations. As a result, we use a one-step proximal gradient update to solve the problem defined in Eq. 6-4.

The proximal operator of $\mathcal{R}_w$ is defined as:

$$\text{prox}_{\gamma\mathcal{R}_w}(x) = \underset{y}{arg\,min}\ \gamma\mathcal{R}_w(y) + \frac{1}{2}\|x - y\|^2. \tag{5-6}$$

In Eq. 6-6, $\|x - y\|^2$ defines the sum of the square difference between x and y. We denote model weights after tth update as $\mathcal{W}^t$, and $\mathcal{W}^{t+1}$ can be obtained by:

$$\mathcal{W}^{t+1} = \text{prox}_{\alpha^{t+1}\gamma\mathcal{R}_w}(u(\mathcal{W}^t, \alpha^{t+1})), \tag{5-7}$$

where $u(\mathcal{W}^t, \alpha^{t+1})$ is an update rule that can be applied for many algorithms, and α^{t+1} is the learning rate at the current step. Take SGD as an example, $u(\mathcal{W}^t, \alpha^{t+1}) = \mathcal{W}^t - \alpha^{t+1}\nabla_{\mathcal{W}^t}\mathcal{L}$. With Eq. 6-6 and Eq. 5-7, model weights $\mathcal{W}$ can then be updated with proximal gradients.

We now present the analytical solution to Eq. 6-6, so that model weights can be efficiently updated. From the structure of $\mathcal{R}_w$, we know that channels with indices $i \in S_l$ will be regularized. This is equivalent to not regularizing channels if $i \notin S_l$. Consequently, we have the following form of the proximal gradient operator:

$$\text{prox}_{\alpha\gamma\mathcal{R}_w}(\mathcal{W}_l) = \begin{cases} \frac{\mathcal{W}_{l[i,:,:,:]}}{\|(1-\mathbf{v}_l)\odot\mathcal{W}_l\|_2} \max\Big(0, -\frac{\hat{N}_l}{\hat{N}}\alpha\gamma \\ \quad +\|(1-\mathbf{v}_l)\odot\mathcal{W}_l\|_2\Big), \text{ if } i \in S_l, \\ \mathcal{W}_{l[i,:,:,:]}, \text{ if } i \notin S_l. \end{cases} \tag{5-8}$$

In Eq. 6-8, we omit the step notation t to simplify the notations, and we still assume W_l is pruned along the output dimension. In this equation, it is easier to see that $\frac{\hat{N}_l}{\hat{N}}$ can balance the regularization strength given different layers. Due to the property of our proposed partial regularization, the term $\|(1-\mathbf{v}_l)\odot\mathcal{W}_l\|_2$ also changes dynamically, since $\hat{N}_l$ is changed after updates of the AGN. As a result, if no adjustment is applied, no matter how large or small $\hat{N}_l$ is, only a constant value $\frac{1}{L}\alpha\gamma$ will be used for the soft-thresholding, $\max(0, \|(1-\mathbf{v}_l)\odot\mathcal{W}_l\|_2 - \frac{1}{L}\alpha\gamma)$, in all circumstances, which is not reasonable. To accompany the changes of v_l, and consequently $\hat{N}_l$, we use $\frac{\hat{N}_l}{\hat{N}}$ to dynamically balance the soft-thresholding parameter between different layers. With Eq. 6-8, we can efficiently update $\mathcal{W}$ with our proposed partial regularization.

5.3.5 Network Pruning via Structural Alignment

We present the algorithm of our method in Algorithm. 4. In Algorithm. 4, D is the training dataset; D_{AGN} is a sub-dataset within D and it is used to train AGN; p decides how much FLOPs is preserved and it described in section 5.3.2; γ and λ are hyperparameters to control the strength of $\mathcal{R}_{\text{FLOPs}}$ and $\mathcal{R}_w$; E is the total number of epochs; E_{start} is the start epoch to train AGN and apply $\mathcal{R}_w$ when optimizing model weights. Note that, to reduce the overhead brought by training AGN, we only use a small sub-set sampled from D. As we discussed in section 5.1, we need to set a start epoch for AGN and $\mathcal{R}_w$. If we apply $\mathcal{R}_w$ when training starts ($E_{\text{start}} = 0$), it will largely

restrict the final performance of the whole model before pruning. Instead, we can have deserved results if we start partial regularization in the middle of the training process. The reason for this problem can come from several perspectives. For example, at the beginning of the training, the classification loss can not produce accurate guidance for pruning since weights are not trained properly. Consequently, it will produce a bad sub-network, and following this sub-network only gives even worse results.

We summarize our method in Fig. 14. The inference path when training model weights and the AGN are different, and they are connected by partial regularization, which is different from current one-shot pruning and soft pruning methods. With this design, model weights are not directly affected by the sub-network architecture, which creates a smooth transition for the selected sub-network before and after pruning.

5.4 Experiments

5.4.1 Settings

We use CIFAR-10 [82] and ImageNet [15] to evaluate the performance of our method. Our method requires one hyper-parameter p to control the FLOPs budget. The detailed choices of p are listed in supplementary materials. We choose ResNets [46] and MobileNet-V2 [126] for comparison. For CIFAR-10, we compare our method with other methods on ResNet-56 and MobileNetV2. For ImageNet, we select ResNet-34, ResNet-50, ResNet-101, and MobileNetV2 as our target models.

We set λ in Eq. 6-3 to 4.0 for all models and datasets. Similarly, we set γ to 0.0005 for all settings. We set E_{start} at 20% of the total training epochs. Detailed numbers of E_{start} are listed in supplementary materials. The range of E_{start} is quite large, which is hard to be explored thoroughly. The current setting already provides good results, but better settings may also exist. To reduce the training costs of the AGN, we random sample 5% of samples from the original dataset for constructing D_{AGN}. With this setup, the additional costs are less than 5% of the original training costs. We train the parameters Θ of the AGN using ADAM [79] with a start learning rate 0.001. Besides the training of the AGN, we follow the standard training recipe of ResNets for both CIFAR-

10 and ImageNet. For MobileNet-V2, We follow the training settings in their original paper [126]. After training, we prune the model by using the sub-network generated from the AGN. The fine-tuning settings are similar to the training setting. Due to space limitations, details of training and fine-tuning are also presented in supplementary materials. In the experimental section, our method is abbreviated as **SANP**: **S**tructural **A**lignment for **N**etwork **P**runing.

5.4.2 CIFAR-10 Results

The results of CIFAR-10 are presented in Tab. 9. For ResNet-56, our method achieves the best performance (in terms of Δ-Acc) compared to other methods. Specifically, our method outperforms the second best method GNN-RL by 0.22% regarding Δ-Acc (SANP $+0.32\%$ vs. GNN-RL $+0.10\%$) when pruning similar FLOPs (SANP 52.0% vs. GNN-RL 54.0%). Our method also outperforms HRank and DMC by 0.25% and 0.41% separately (DMC $+0.07\%$ and HRank -0.09%). The gap between other methods and our method is even larger. For MobileNet-V2, our method prunes most FLOPs (46.0%) and achieves the best performance (Δ-Acc: $+0.45\%$). Compared to the second best method, DMC, our method prunes 6% more FLOPs and outperforms it by 0.19%. SCOP also prunes 40% FLOPs, and the gap between our method and SCOP is even larger (0.69% better in terms of Δ-Acc). The uniform setting and DCP prunes around 26% FLOPs, but the performance is still worse than our method.

5.4.3 ImageNet Results

All results for the ImageNet dataset are shown in Tab. 10.

ResNet-34. Our method achieves 73.43% Top-1 accuracy and 91.48% Top-5 accuracy, which is better than the baseline by 0.19% and 0.16% for Top-1/Top-5 accuracy separately. At the same time, our method removes 44.1% FLOPs, which is on par with other methods. It is obvious that the advantage of our method is clear compared to other methods. Our method prunes similar FLOPs to SCOP and DMC. However, the Δ Top-1 Acc of our method is 0.88% and 0.92% better than SCOP and DMC, respectively, and we have similar observations for Δ Top-5 Acc (0.60% and 0.47% better than SCOP and DMC). Taylor has the second best Δ Top-1 Acc, but the pruned FLOPs is much lower than our method (ours: 44.1% vs. Taylor: 24.2%). The advantage of our

method compared to FPGM is more obvious.

ResNet-50. Our method achieves 76.47% Top-1 accuracy and 93.00% Top-5 accuracy, which is also better than the baseline Top-1/Top-5 accuracy. The second best method, CHIP, removes 48.7% FLOPs while maintaining the original performance. Our method outperforms CHIP by 0.41% in terms of the Δ Top-1 accuracy while pruning near 8% more FLOPs. NPPM is a strong baseline for ResNet-50, our method outperforms by 0.60% in Δ -Top-1 Acc. GNN-RL and Random-Pruning are two recent pruning works. Our method outperforms them by 2.23% and 1.16% separately, while our method prunes more FLOPs. PHP, DCP, CCP, and DMC have similar Δ-Top 1 accuracy. The gap between our method and these methods ranges from 1.21% to 1.47% regarding Δ-Top 1 accuracy. The advantage of our method compared to the rest methods is more apparent.

ResNet-101. Our method achieves 78.14% Top-1 accuracy and 94.00% Top-5 accuracy, which is 0.61% and 0.29% better than the baseline Top-1 and Top-5 accuracy. Although DMC prunes a little bit more FLOPs, it only achieves 77.41% Top-1 accuracy after pruning which is 0.73% lower than our method. FPGM, Taylor, and PFP remove less than 50% FLOPs. Our method prunes at least 10% more FLOPs and still has an advantage in terms of Δ-Top 1 accuracy (from 0.66% to 1.55%).

MobileNet-V2. MobileNet-V2 is generally harder to prune compared to ResNets. All comparison methods on MobileNet-V2 remove around 30% FLOPs. Our method achieves 72.05% Top-1 accuracy and 90.37% Top-5 accuracy, which is 0.14% and 0.07% better than the baseline Top-1 and Top-5 accuracy. Given the similar pruning rate, our method is 1.15%, 0.94%, 1.11% and 1.14% higher than Random-Pruning, MetaPruning CC, and AMC separately regarding Δ-Top-1 Acc. MetaPruning prunes most FLOPs, but the performance is much lower than our method. In short, our method can also be applied to lightweight CNNs, like MobileNet-V2.

5.4.4 Analysis of Our Method

The effectiveness of partial regularization. To verify whether our proposed partial regularization is effective, we plot the average channel norm of different groups within each block for ResNet-56 and MobileNet-V2 on CIFAR-10. The results are shown in Fig. 15. The average channel norm for the group with partial regularization and without partial regularization are obtained by

$\frac{1}{\hat{N}_l}\sum_{i\in S_l}\|\mathcal{W}_{l[i,:,:,:]}\|_{\text{GL}}$ and $\frac{1}{C_l-\hat{N}_l}\sum_{i\notin S_l}\|\mathcal{W}_{l[i,:,:,:]}\|_{\text{GL}}$ separately. Weights with partial regularization are effectively aligned. On the contrary, weights without partial regularization are lightly affected, which justifies the strength of our proposed partial regularization.

The impact of λ. We study the impact of λ when training the AGN, and we plot the test accuracy and $\mathcal{R}_{\text{FLOPs}}$ in Fig. 17a and Fig. 17e. From the figures, we can see that if λ is too large, it will have negative impacts on learned sub-networks. Otherwise, our method is robust to λ,

The effect of AGN. In Fig. 17b and Fig. 17f, we plot the test accuracy when learning the AGN given different pruning rates. We construct a **Simple** baseline, which parameterizes each channel by using one learnable parameter. We can see that when the parameterization space shrinks, the performance of learned sub-networks will be affected severely. In addition, the learning is slower, and the best sub-network performance is also much worse than using AGN.

The effect of $\mathbf{E_{start}}$. In the method section, we argue that inserting partial regularization in the middle training process is beneficial. We plot the results of $E_{\text{start}} = 0$ and $E_{\text{start}} = 40$ in Fig. 17c and Fig. 17g. In general, they can find sub-networks with similar performance, but $E_{\text{start}} = 0$ is worse than $E_{\text{start}} = 40$ when the pruning rate is large ($p = 0.35$). More importantly, the full model accuracy of $E_{\text{start}} = 0$ is 92.96% (average across different runs), which is around 0.50% worse than $E_{\text{start}} = 40$, which suggests that using partial regularization at the beginning will limit the capacity of the full model.

The effect of different setups. We construct two additional baselines to see how partial regularization helps to produce a better sub-network within the full model. The **One-Shot** baseline directly trains the AGN on the pre-trained model. The **w/o Partial Regularization** baseline set $\gamma = 0$, and the rest settings are the same as our method. The related results are shown in Fig. 17d and Fig. 17h. From these figures, we can see that partial regularization always produces the best sub-network from the full model. In addition, 'w/o partial regularization' is worse than the one-shot setting. This is probably because it is hard to capture the changes of model weights without partial regularization, which makes the training of AGN much harder than the rest settings. **More comparisons on the ImageNet dataset.** To further show how our method improves the one-shot setting, we present the sub-network performance before and after fine-tuning for one-shot and partial regularization settings in Fig. 16 and Tab. 11. From Fig. 16, we can see that partial regularization still produces better sub-networks on ImageNet, and it is around 10% better in terms of Top-1

accuracy than the one-shot setting for different models. The advantage of partial regularization naturally extends to results after fine-tuning, as shown in Tab. 11.

5.5 Conclusion

In this paper, we investigate how partial regularization helps to produce a better sub-network for network pruning. Specifically, our method uses AGN to guide partial regularization across the training process. We further provide an efficient way to update model weights through proximal gradients. With these designs, partial regularization effectively reduces the gap between the sub-network within the full model and the pruned model. Our method then starts from a better sub-network, thus resulting in a better final pruned model. Extensive experimental results on CIFAR-10 and ImageNet show the effectiveness of our method.

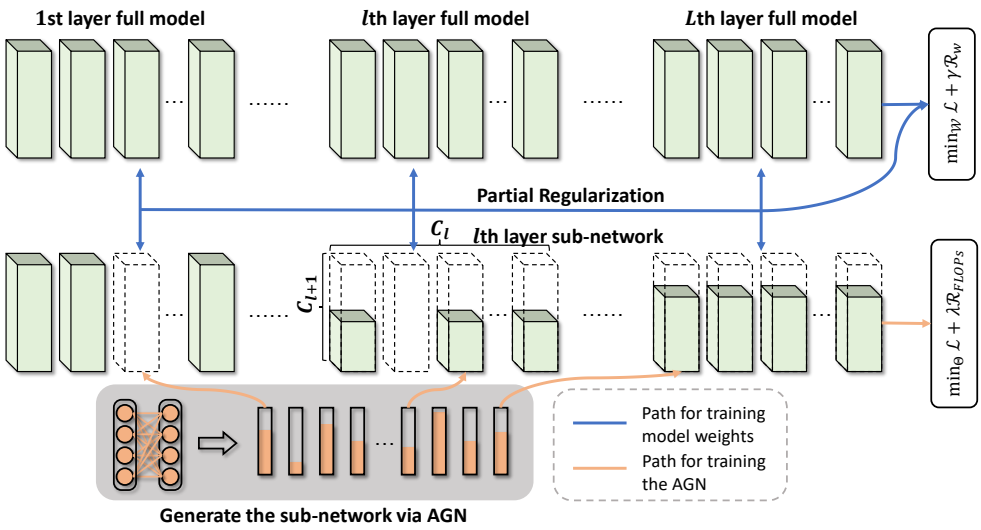
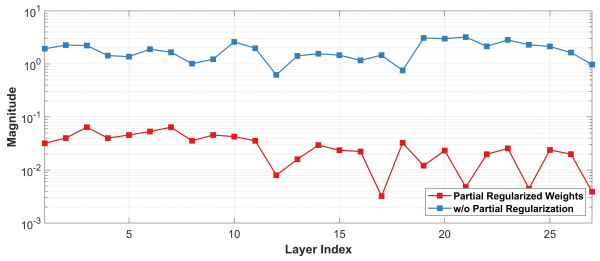


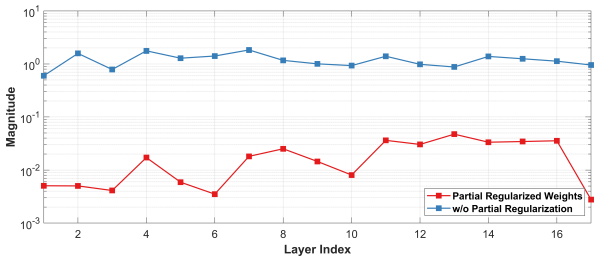
Figure 14: Overview of the proposed method. The AGN generates the sub-network to guide the partial regularization during the training process of model weights. The AGN is trained by evaluating sub-networks on a sub-set from the whole dataset. Both model weights and AGN are trained in an end-to-end differentiable manner.

Table 9: Comparison of results on CIFAR-10. Δ-Acc represents the performance changes relative to the baseline, and $+/-$ indicates an increase/decrease, respectively.

Architecture	Method	Baseline Acc	Pruned Acc	Δ-Acc	Pruned FLOPs
ResNet-56	DCP-Adapt [154]	93.80%	93.81%	+0.01%	47.0%
	SCP [75]	93.69%	93.23%	−0.46%	51.5%
	FPGM [49]	93.59%	92.93%	−0.66%	52.6%
	SFP [48]	93.59%	92.26%	−1.33%	52.6%
	FPC [47]	93.59%	93.24%	−0.25%	52.9%
	HRank [96]	93.26%	92.17%	−0.09%	50.0%
	DMC [32]	93.62%	92.69%	+0.07%	50.0%
	GNN-RL [147]	93.49%	93.59%	+0.10%	**54.0%**
	SANP (ours)	93.49%	**93.81%**	**+ 0.32%**	52.0%
MobileNetV2	Uniform [154]	94.47%	94.17%	−0.30%	26.0%
	DCP [154]	94.47%	94.69%	+0.22%	26.0%
	DMC [32]	94.23%	94.49%	+0.26%	40.0%
	SCOP [132]	94.48%	94.24%	−0.24%	40.3%
	SANP (ours)	94.52%	**94.97%**	**+0.45%**	**46.0%**



(a) ResNet-56.



(b) MoibleNet-V2.

Figure 15: (a) and (b): the average norm of channels with or without partial regularization for ResNet-56 and MobileNet-V2.

Table 10: Comparison results on ImageNet with ResNet-34/50/101 and MobileNet-V2.

Architecture	Method	Baseline Top-1 Acc	Baseline Top-5 Acc	Δ Top-1 Acc	Δ Top-5 Acc	Pruned FLOPs
ResNet-34	FPGM [49]	73.92%	91.62%	−1.29%	−0.54%	41.1%
	Taylor [112]	73.31%	-	−0.48%	-	24.2%
	DMC [32]	73.30%	**91.42%**	−0.73%	−0.31%	43.4%
	SCOP [132]	**73.31%**	**91.42%**	**−0.69%**	**−0.44%**	**44.8%**
	SANP (ours)	73.24%	91.32%	**+0.19%**	**+0.16%**	**44.1%**
ResNet-50	DCP [154]	76.01%	92.93%	−1.06%	−0.61%	55.6%
	CCP [119]	76.15%	92.87%	−0.94%	−0.45%	54.1%
	FPGM [49]	76.15%	92.87%	−1.32%	−0.55%	53.5%
	ABCP [97]	76.01%	92.96%	−2.15%	−1.27%	54.3%
	DMC [32]	76.15%	92.87%	−0.80%	−0.38%	55.0%
	SCOP [132]	**76.15%**	**92.87%**	**−0.89%**	**−0.34%**	**54.6%**
	PFP [94]	76.13%	92.86%	−0.92%	−0.45%	44.0%
	CHIP [131]	**76.15%**	**92.87%**	**+0.00%**	**+0.04%**	**48.7%**
	NPPM [30]	**76.15%**	**92.87%**	**−0.19%**	**+0.12%**	**56.0%**
	Random-Pruning [89]	75.83%	92.92%	−0.75%	−0.40%	51.0%
	GNN-RL [147]	76.10%	-	−1.82%	−	53.0%
	SANP (ours)	76.06%	92.86%	**+ 0.41%**	**+ 0.17%**	**56.2%**
ResNet-101	FPGM [49]	77.37%	93.56%	−0.05%	0.00%	41.1%
	Taylor [112]	77.37%	-	−0.02%	-	39.8%
	DMC [32]	77.37%	**93.56%**	+0.04%	+0.03%	**56.0%**
	PFP [94]	77.37%	93.56%	−0.94%	−0.44%	45.1%
	SANP (ours)	77.53%	93.71%	**+ 0.61%**	**+ 0.29%**	55.4%
MobileNet-V2	Uniform [126]	71.80%	91.00%	−2.00%	−1.40%	30.0%
	AMC [50]	71.80%	-	−1.00%	-	30.0%
	CC [92]	71.88%	-	−0.97%	-	28.3%
	MetaPruning [102]	72.00%	-	−0.80%	-	**30.7%**
	Random-Pruning [89]	71.88%	-	−1.01%	-	29.1%
	SANP (ours)	71.91%	90.30%	**+ 0.14%**	**+ 0.07%**	29.1%

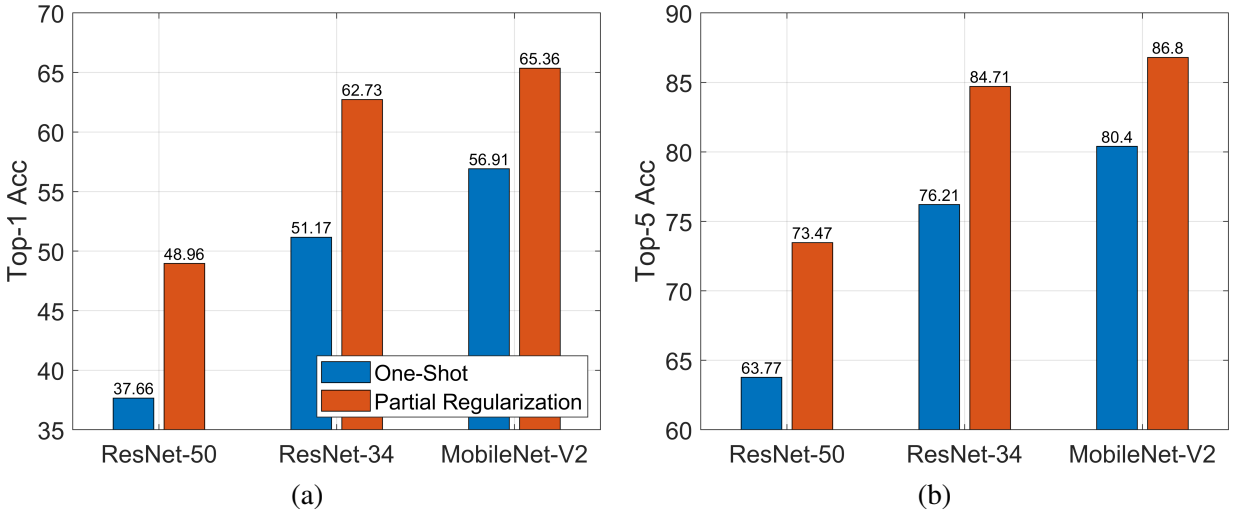
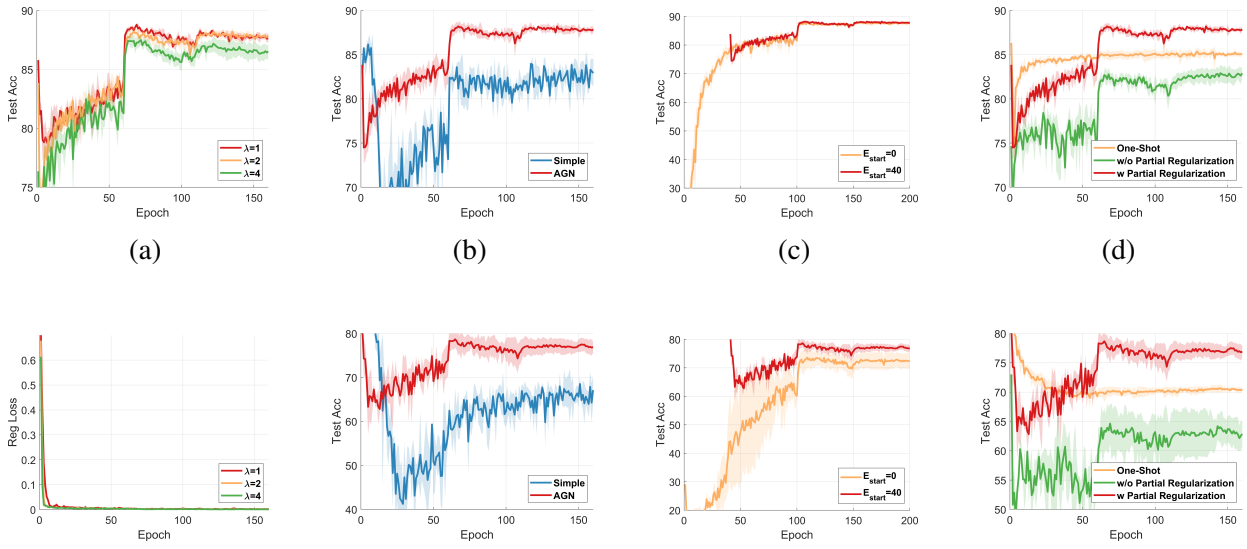


Figure 16: Top-1 and Top-5 accuracy after pruning given different settings.

Table 11: Performance of pruned models given different pruning settings on ImageNet.

Settings	Architecture	Baseline Top-1 Acc	Δ Top-1 Acc	Pruned FLOPs
One-Shot	ResNet-34	73.31%	-0.50%	44.0%
SANP		73.24%	$+0.19\%$	44.1%
One-Shot	ResNet-50	76.13%	-0.57%	56.0%
SANP		76.06%	$+0.41\%$	55.2%
One-Shot	MobileNetV2	71.88%	-0.42%	29.3%
SANP		71.91%	$+0.07\%$	29.1%



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 17: (a, e): the impact of λ in $\mathcal{R}_{\text{FLOPs}}$. (b, f): the effect of the architecture of AGN. (c, g): the effect of E_{start}. (d, h): the effect of different setups. Experiments are conducted on CIFAR-10 with ResNet-56 and $p = 0.5$ (a,b,c,d,e) and $p = 0.35$ (f,g,h).

6.0 BilevelPruning: Unified Dynamic and Static Channel Pruning for Convolutional Neural Networks

6.1 Background

Convolutional neural networks (CNNs) have recently achieved great successes in many machine learning and computer vision tasks [83, 122, 123, 129, 3]. Despite the remarkable performance, the computational and storage costs of most CNNs are quite expensive due to their complex architectures. Such costs have become the major bottleneck to deploying CNNs on portable devices with limited resources (*e.g.*, memory, CPU, energy). To solve this problem, many researchers focus on how to truncate the costs of deep models effectively. These researches can be summarized into several directions, such as weight pruning [43], weight quantization [7], structural pruning [88], matrix decomposition [16] and so on. Among these approaches, channel pruning, which belongs to structural pruning, is a promising way to effectively reduce computational and storage costs since other methods often require additional post-processing steps to acquire actual compression. Thus, this work focuses on investigating the channel pruning technique.

A series of channel pruning approaches [51, 108, 154] use different criteria to evaluate the importance of each channel, and the redundant (less important) channels are pruned. These approaches are also called static channel pruning. The benefit of static channel pruning is that unessential channels are permanently removed, which saves both storage and computational costs. However, the model capacity of static pruning is restricted by using a fixed sub-network. Some more recent works [37, 100] try to select important channels based on inputs and intermediate feature maps at inference time, and they belong to dynamic channel pruning. Given different inputs, different sub-networks are dynamically selected, which largely improves the model capacity. Most existing dynamic pruning methods preserve all channels to ensure the model has the largest capacity. Compared to static pruning, dynamic pruning methods often achieve better performance but at the cost of requiring extra storage space.

As mentioned in recent storage efficient dynamic pruning work [6], the large storage costs of most dynamic pruning methods prohibit them from being deployed in resource-limited portable

devices. To save storage costs for dynamic pruning, storage efficient pruning [6] heuristically combines static and dynamic channel pruning by using reinforcement learning. The final pruned model is obtained by combining the outputs of both static and dynamic pruning through a hand-designed function. Channels with low importance are permanently removed. Although this approach achieves good results, there are several drawbacks. First, the sub-networks from dynamic and static pruning in their method are treated separately. In their work, static sub-networks are not considered when conducting dynamic pruning and vice versa, which generally hurts the performance. Moreover, the learning of position and importance of remaining channels are also separated. Second, they use a hand-designed function to fuse dynamic and static pruning results, leading to sub-optimal performance due to the lack of the learning process.

To tackle the aforementioned problems, we propose a new model to integrate static and dynamic pruning. To naturally form relationships between static and dynamic sub-networks, we look for the best static sub-network by evaluating dynamic sub-networks. We then integrate the learning of static and dynamic sub-networks by using bi-level optimization. Moreover, the static sub-network is never evaluated directly, and it's only implicitly trained through dynamic sub-networks. Such a setup ensures that dynamic sub-networks fully utilize their static counterpart. Our new formulation integrates dynamic and static channel pruning, leading to a better trade-off between storage costs and dynamic flexibility. Specifically, the limited model capacity in static pruning is compensated by dynamic pruning, and the extra storage costs in dynamic pruning are also reduced by static pruning. As a result, our model enjoys the benefits of both static and dynamic pruning, and their shortcomings are compensated by each other. The final pruning results are also learned in an end-to-end fashion without handcrafted functions.

In our method, the selection of channels for both dynamic and static pruning is based on differentiable gates, and they can be optimized through backpropagation. Under this setting, we can apply parameter constraints on the static sub-network and FLOPs constraints on dynamic sub-networks. Previous dynamic pruning works [100, 6] often require hyper-parameters to implicitly specify the computational budget and/or the trade-off between dynamic and static pruning. But our method can set them directly, which is an additional benefit of our method.

In summary, the major contributions of our method can be summarized as follows:

- We propose a novel channel pruning method, which unifies both dynamic and static pruning.

Dynamic and static sub-networks are connected by evaluating the static sub-network through dynamic sub-networks instead of training them in parallel.

- We integrate static and dynamic pruning by formulating them as a bi-level optimization problem. By doing so, our method enjoys benefits from both static and dynamic pruning. In addition, we present an efficient method for optimizing the matrix-vector product in bi-level optimization.
- The experimental results on CIFAR-10 and ImageNet datasets suggest that our method achieves state-of-the-art performance compared to existing dynamic and static pruning methods.

6.2 Related Works

6.2.1 Regular Pruning

Weight pruning. Weight pruning aims to eliminate redundant parameters. An early work [139] prunes model weights based on minimum description length. Optimal brain damage [84] and surgeon [44] utilize second-order information to remove connections. The drawback is that the computation of second-order derivatives is expensive. More recently, *Han et al.* [43] propose to prune weights based on their magnitude. Magnitude pruning is very efficient, and the cost of computing L_1 or L_2 magnitude is negligible. Regular network pruning approaches follow a three-stage pipeline: training, pruning, and fine-tuning. *Zhang et al.* [104] raise questions about such standard procedure and argue that the sub-network architecture obtained by pruning is more valuable than the remaining weights. They also show that retraining sub-networks from scratch is enough to recover the performance. On the other hand, the lottery ticket hypothesis (LTH) [22] shows that good sub-networks exist at the initialization stage. A series of works [124, 113] related to LTH extend this work to larger datasets and more complicated architectures. Another line of research [153, 120] shows that training masks on top of untrained models can also lead to ideal performance. The model after weight pruning has much less parameters but it requires sparse matrix libraries or specific hardware to achieve actual savings in storage and computational costs.

Structural Pruning. Structural pruning tries to remove certain structures in a deep model,

such as kernels, channels, layers, and so on. In contrast to weight pruning, structural pruning can accelerate inference speed and save storage costs without additional effort. Filter pruning [88] tries to prune filters from CNNs that are having small effects on the outputs. Similar to magnitude pruning, the importance of each filter is measured by L_1 or L_2 norm of the filter, and L_1 norm performs better in their settings. Unlike filter pruning, soft filter pruning [48] does not remove filters during training, and they instead reset these filters and put them into training again. Network slimming [**?**] uses L_1 sparsity regularization on scaling factors of channels from batch normalization layers, and channels with small scaling factors are removed. Sparse structural selection [70] extends network slimming to more structures, including layers. Discrimination-aware pruning [154] not only considers the norms of channels but also uses classification loss to identify unimportant channels. Automatic model compression [50] applies reinforcement learning (RL) for structural pruning. RL is used since it can better cooperate with the discrete nature of structural pruning. Greedy pruning [144] starts from an empty model and adds connections that reduce the loss value most. Static pruning methods directly reduce storage costs, but the pruned model is fixed leading to limited model capacity.

The static pruning part of our method has a close connection to structural pruning approaches. However, unlike the aforementioned approaches, the static sub-network is not directly evaluated; instead, the static sub-network can be viewed as the backbone model for dynamic sub-networks.

6.2.2 Dynamic Pruning

Regular pruning methods are designed to find a fixed sub-network for all inputs. On the other hand, dynamic pruning aims to provide different sub-networks for different inputs, which increases the model capacity given the same inference budget. Runtime neural pruning [95] treats dynamic pruning for different layers as a Markov decision process and uses reinforcement learning for training. SkipNet [137] uses a gating module to skip convolution blocks based on previous feature maps dynamically. The dynamic skipping problem is formulated as a sequential decision-making problem, which is jointly solved by reinforcement and supervised learning. Adaptive neural networks [4] adaptively select the components of a deep model based on the input examples. They also introduce an early exit mechanism to further reduce computational costs. In feature boosting

and suppression [37], they propose to skip unimportant input and output channels dynamically. They use Lasso regularization to introduce sparsity on the runtime channel importance.

Besides pruning, some works utilize the power of dynamic computation to improve the design of CNNs. CondConv [142] replace traditional convolutions with learned specialized convolutional kernels for each input. Dynamic convolution [8] applies input-dependent attention on multiple convolution kernels, which drastically improves the model capacity.

Most aforementioned works need to keep the full model to achieve the best performance. To reduce storage costs, storage efficient dynamic pruning [6] introduces static pruning along with dynamic pruning to reduce storage costs. Our method also aims to save storage costs while maintaining dynamic flexibility. But unlike storage-efficient pruning, we explicitly consider integrating static and dynamic pruning, which leads to a better trade-off. Our method is also closely related to bi-level optimization methods and other optimization techniques [68, 67, 61, 66, 60, 38, 62].

6.3 Proposed Method

6.3.1 Notations

To better illustrate our method, we first introduce some necessary notations. In a CNN, the feature map of i-th layer can be represented by $\mathcal{F}_i \in \mathcal{R}^{B\times C_i\times W_i\times H_i}$, $i = 1, \ldots, L$, where B is the mini-batch size, C_i is the number of channels, W_i and H_i are the width and height of the current feature map, L is the number of layers. $\odot$ is the element-wise product. We use $\sigma(x) = \frac{1}{1+e^{-x}}$ to represent the sigmoid function. $\lfloor\cdot\rceil$ is used to represent rounding to the nearest integer.

6.3.2 Static and Dynamic Settings

For static pruning, we can use a 0-1 vector to indicate whether to prune a channel or not. To produce such vectors, we use the following function:

$$g_s = \lfloor v_s \rceil,\ v_s = \sigma((\theta_s + \mu)/\tau), \tag{6-1}$$

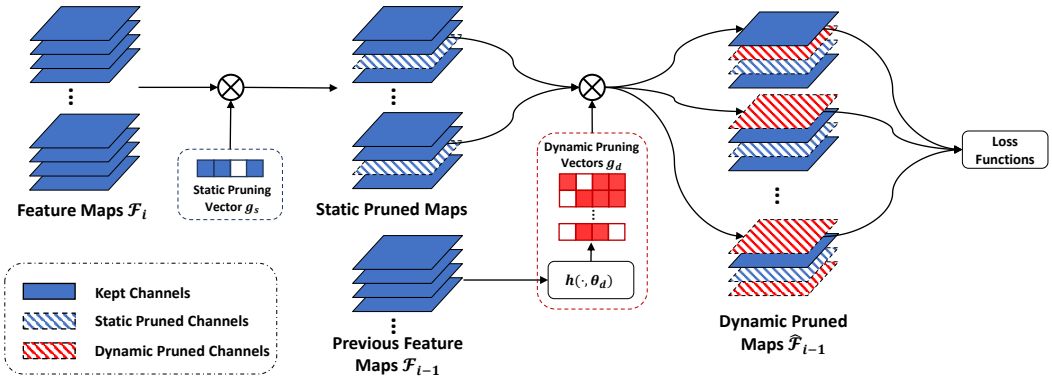


Figure 18: The flowchart of the proposed method. In the figure, we first conduct static pruning followed by dynamic pruning. Instead of naively combining static pruning and dynamic pruning, we formulate the pruning problem as a bi-level optimization problem to unify static and dynamic pruning. The whole process is differentiable, which allows efficient gradient based optimization.

where $g_s \in R^{C_i}$ is the static pruning vector, $\mu \sim \text{Gumbel}(0, 1)$, τ is the temperature hyper-parameter, and $\theta_s \in R^{C_i}$ are learnable parameters for static pruning. v_s is a continuous vector, we further round it to its nearest neighbor g_s. The rounding function is not differentiable, we solve this problem by using the straight-through estimator [2] to calculate gradients. Now we have the binary vector g_s for static pruning. The generation of g_s can be seen as using the straight-through Gumbel-sigmoid [73] trick to approximate Bernoulli distribution.

We can use similar formulations for dynamic pruning and consider feature maps from the $i-1$th layer. The detailed formulation can be written as:

$$g_d = \lfloor v_d \rceil, \; v_d = \sigma((h(\mathcal{F}_{i-1}; \theta_d) + \mu)/\tau), \tag{6-2}$$

where $g_d \in R^{B \times C_i}$ is the dynamic pruning vector, and $h(\cdot; \theta_d)$ is a routing function parameterized by θ_d to dynamically select channels, the rest settings are the same as static pruning. The routing function $h(\cdot; \theta_d)$ is composed of global average pooling followed by squeeze and excitation (SE) [55], which is suggested by FBS [37]. By using SE, we can save parameters when some layers in a model are too wide (like later layers of MobileNet-V2).

After we have g_s, the resulting feature map obtained by static pruning can be represented as:

$$\tilde{\mathcal{F}}_i = g_s \odot \mathcal{F}_i \tag{6-3}$$

where $\tilde{\mathcal{F}}_i$ is the pruned feature map by applying g_s, and g_s is first expanded to have the same dimension of $\mathcal{F}_i$. After having $\tilde{\mathcal{F}}_i$, we can regard it as the new base feature map, and apply dynamic pruning on it:

$$\hat{\mathcal{F}}_i = g_d \odot \tilde{\mathcal{F}}_i. \tag{6-4}$$

where $\hat{\mathcal{F}}_i$ is the dynamically pruned feature map, g_d is also first expanded to have the same dimension of $\tilde{\mathcal{F}}_i$ and conduct element-wise product. We can then remove channels from $\mathcal{W}_i$ based on $\hat{\mathcal{F}}_i$. One can also use a more sophisticated method to specify the relationships between static and dynamic pruning. For example, one can directly multiply g_s along with the output dimension of the weight matrix θ_d of the routing function. However, we found that such modifications do not provide any benefits.

6.3.3 Unified Dynamic and Static Pruning

Since all operations of static and dynamic pruning are differentiable, we can formulate the static pruning problem as follows:

$$\min_{\Theta_s} \mathcal{L}(f(x; \Theta_s, \Theta_d), y) + \lambda \mathcal{R}_p(T_p(\Theta_s), p_p \hat{T}_p), \tag{6-5}$$

where Θ_s is the collection of all learnable parameters θ_s for static pruning, $\hat{T}_p$ is the number of all prunable parameters, $T_p(\Theta_s)$ is the remained number of parameters decided by the static sub-network, $\mathcal{R}_p$ is the regularization term to reduce the number of parameters to a predefined threshold p_p, x, y are input samples and their labels, $f(\cdot; \Theta_s, \Theta_d)$ is a sub-network from the whole network and it is parameterized by Θ_s and Θ_d, and $\mathcal{L}$ is the cross-entropy loss for classification. We omit the model weights $\mathcal{W}$, since we fix $\mathcal{W}$ in $f(\cdot; \Theta_s, \Theta_d)$ during the pruning stage. Similarly, the dynamic pruning problem can be defined as:

$$\min_{\Theta_d} \mathcal{L}(f(x; \Theta_s, \Theta_d), y) + \lambda \mathcal{R}_r(T_r(\Theta_d), p_r \hat{T}_r) + \gamma R_d(v_d), \tag{6-6}$$

where Θ_d is the collection of all θ_d, $\hat{T}_r$ is the total prunable FLOPs of the model, $T_r(\Theta_d)$ is the average FLOPs of B dynamic sub-networks, $\mathcal{R}_r$ is the regularization term to push average FLOPs of dynamic sub-networks to the corresponding threshold p_r, γ is the hyper-parameter for $\mathcal{R}_d$, and

$\mathcal{R}_d$ is a regularization term to prevent dynamic sub-networks from collapsing to a single trivial solution. We define $R_d(v_d)$ as follows:

$$R_d(v_d) = \frac{1}{L}\sum_{i=1}^{L}\|v_d^i - \bar{v}_d^i\|_2^{-1}, \tag{6-7}$$

where v_d^i is the continuous dynamic vector of ith layer, and $\bar{v}_d^i = \frac{1}{B}\sum v_d^i$ is the average of dynamic vectors from the current layer. We use continuous v_d^i instead of discrete g_d^i. Because the variance of g_d^i could be very small for early layers (also pointed out in [142]), which results in instability and difficulty when optimizing Eq. 6-7.

To reduce the number of hyper-parameters, we use the same λ for both $\mathcal{R}_r$ and $\mathcal{R}_p$. Given the objective function in Eq. 6-4 and Eq. 6-5, we can see that the static sub-network is not directly evaluated, and it is used as the new backbone model for dynamic pruning.

We have the objective functions to conduct static and dynamic pruning; a natural question is how to train them together? We can simply put Eq. 6-5 and Eq. 6-6 together and optimize them using gradient descent. However, such a process will make the training of static and dynamic pruning interfere with each other, which will hurt the pruning result (shown in supplementary materials). Alternatively, we can optimize Eq. 6-5 and Eq. 6-6 iteratively, but doing so can not integrate static and dynamic pruning, and training of static and dynamic pruning are separated.

To unify the training of dynamic and static sub-networks, we can consider the following bi-level optimization [13] problem:

$$\begin{aligned} &\min_{\Theta_d} \mathcal{L}(f(x;\Theta_s^*,\Theta_d),y) + \lambda\mathcal{R}_r(T_r(\Theta_d),p_r\hat{T}_r) + \gamma R_d(v_d) \\ &s.t.\ \Theta_s^* = \arg\min_{\Theta_s} \mathcal{L}(f(x;\Theta_s,\Theta_d),y) + \lambda\mathcal{R}_p(T_p(\Theta_s),p_p\hat{T}_p), \end{aligned} \tag{6-8}$$

where the outer problem is to find good dynamic sub-networks based on the optimal static sub-network, and the inner problem is getting the optimal static sub-network. The formulation of the problem in Eq. 6-8 also appeared in gradient-based hyperparameter optimization [21, 1] and differentiable neural architecture search [99].

The inner problem in Eq. 6-8 is not easy to solve since how to generate dynamic sub-networks without learning is unclear. A naive random sampling of dynamic sub-networks only produces trivial results. We then approximate Θ_s^* by training one step, and it has been proven effective in previous works [1, 99]. Multi-step approximation can also be used, but it will dramatically

increase the computational costs since one has to perform backpropagation through multiple time steps. As a result, we chose to use the one-step approximation. To simplify notations, we use $\mathcal{L}(\Theta_s, \Theta_d) = \mathcal{L}(f(x; \Theta_s, \Theta_d), y)$ for the following derivations. Let us first define the update rule u for Θ_s: $\Theta_s^{'} = u(\Theta_s, \eta)$, and η is the learning rate. Take SGD as an example, the update rule of Θ_s is $\Theta_s^{'} = u(\Theta_s, \eta) = \Theta_s - \eta(\nabla_{\Theta_s}\mathcal{L}(\Theta_s, \Theta_d) + \lambda\nabla_{\Theta_s}\mathcal{R}_p)$. We then approximate Θ_s^* with $\Theta_s^{'}$, and the gradient with respect to Θ_d is:

$$
\begin{aligned}
&\nabla_{\Theta_d}\mathcal{L}(\Theta_s^*, \Theta_d) + \lambda\nabla_{\Theta_d}\mathcal{R}_r + \gamma\nabla_{\Theta_d}R_d \\
\approx\ &\nabla_{\Theta_d}\mathcal{L}(\Theta_s - \eta(\nabla_{\Theta_s}\mathcal{L}(\Theta_s, \Theta_d) + \lambda\nabla_{\Theta_s}\mathcal{R}_p), \Theta_d) \\
&\quad + \lambda\nabla_{\Theta_d}\mathcal{R}_r + \gamma\nabla_{\Theta_d}R_d \\
=\ &\nabla_{\Theta_d}\mathcal{L}(\Theta_s^{'}, \Theta_d) - \eta\nabla^2_{\Theta_d,\Theta_s}\mathcal{L}(\Theta_s, \Theta_d)\nabla_{\Theta_s^{'}}\mathcal{L}(\Theta_s^{'}, \Theta_d) \\
&\quad + \lambda\nabla_{\Theta_d}\mathcal{R}_r + \gamma\nabla_{\Theta_d}R_d.
\end{aligned}
\tag{6-9}
$$

where the last equality is obtained by applying the chain rule. Note that, we use $\Theta_s^{'}$ to approximate Θ_s^*, and the second line of Eq. 6-9 is an approximated solution instead of an analytical solution. At first glance, the final gradient contains a costly matrix-vector product. However, we will show that the second-order derivative is just the multiplication of two first-order terms. Let's take a specific layer i as an example, we first rearrange Eq. 6-3 and Eq. 6-4: $\hat{\mathcal{F}}_i = g^i \odot \mathcal{F}_i$, and $g^i = g_s^i \odot g_d^i$, we have:

$$
\begin{aligned}
\nabla^2_{\theta_d^i,\theta_s^i}\mathcal{L}(\theta_s^i, \theta_d^i) =&\nabla_{\theta_d^i}(\nabla_{g^i}\mathcal{L}(\theta_s^i, \theta_d^i)\nabla_{\theta_s^i}g^i) \\
=&\nabla_{\theta_d^i}(\nabla_{g^i}\mathcal{L}(\theta_s^i, \theta_d^i)((\nabla_{\theta_s^i}g_s^i) \odot g_d^i)) \\
=&\nabla_{g^i}\mathcal{L}(\theta_s^i, \theta_d^i)(\nabla_{\theta_s^i}g_s^i \cdot \nabla_{\theta_d^i}g_d^i).
\end{aligned}
\tag{6-10}
$$

To simplify derivation, θ_d^i is flattened as a vector, and we omit all transpose notations. The result of Eq. 6-10 indicates that the second-order term is just the multiplication of two Jacobians matrices followed by $\nabla_{g^i}\mathcal{L}(\theta_s^i, \theta_d^i)$, which is more efficient than using finite difference approximation for the matrix-vector product [128, 99]. The calculation of the Jacobians matrix is also simple since the computation of g_s^i and g_d^i only includes simple operations like matrix multiplications and element-wise functions. With Eq. 6-9 and Eq, 6-10, we always update Θ_d by taking Θ_s into consideration, which is ignored in storage efficient pruning [6]. The calculation of the Jacobians matrix is listed in the supplementary materials. In practice, we use the Adam optimizer [79] instead of SGD. The

Algorithm 5: Unified Dynamic and Static Channel Pruning

Input: dataset for pruning: D_{prune}; remained rate of FLOPs and parameters: p_r and p_p; hyper-parameter: λ and γ; training epochs for pruning: E_{prune}; pre-trained CNN: f.

Initialization: initialize Θ_d randomly; initialize Θ_s uniformly; freeze $\mathcal{W}$ in f.

for $e := 1$ *to* E_{prune} **do**

  for *a mini-batch* (x, y) *in* D_{prune} **do**

    1. produce static and dynamic vectors: g_s and g_d. (Eq. 6-1 and 6-2)

    2. calculate gradients w.r.t Θ_s from Eq. 6-5.

    3. update Θ_s by Adam optimizer.

    4. calculate gradients w.r.t Θ_d (Eq. 6-9 and Eq. 6-10).

    5. update Θ_d by Adam optimizer.

  end

end

Get f' by pruning f based on g_s.

Return f' for fine-tuning.

derivation of gradients w.r.t Θ_d given the Adam optimizer is also provided in the supplementary materials.

6.3.4 The Overall Algorithm

We follow the three-stage procedure of regular pruning methods: training, pruning, and fine-tuning. During pruning, Θ_s and Θ_d are learned; pruned Θ_d and $\mathcal{W}$ are trained during fine-tuning. After we obtain static and dynamic sub-networks by solving the problem in Eq. 6-8, we permanently remove channels with 0 in g_s, which also saves costs for fine-tuning. As a result, only a static sub-network that is important to dynamic sub-networks is persevered for finetuning. The rest parts of the model are removed to achieve the goal of saving memory costs. The corresponding channels in the dynamic routing function $h(\cdot)$ are also removed. The fine-tuning loss can be written as:

$$\min_{\Theta_d, \mathcal{W}} \mathcal{L}(f'(x; \Theta_d, \mathcal{W}), y) + \lambda \mathcal{R}_r(T_r(\Theta_d), p_r \hat{T}_r) + \gamma R_d(v_d), \tag{6-11}$$

Table 12: Comparison of the accuracy changes (Δ-Acc), reduction in FLOPs, and the number of parameters of various channel pruning algorithms on CIFAR-10. '+/-' of Δ-Acc indicates increase/decrease compared to baselines. '-' in '$\downarrow$ #Params' indicates increase of parameters.

Method	Architectures	Dynamic	Base Acc	Acc	Δ-Acc	$\downarrow$FLOPs	$\downarrow$ #Params
FBS [37]	CifarNet	✓	91.37%	89.88%	-1.49%	74.6%	-11.0%
SEP-A [6]		✓	92.07%	91.23%	-0.84%	74.5%	**22.0%**
SEP-B [6]		✓	92.07%	91.42%	-0.65%	74.5%	-31.0%
UDSP (ours)		✓	92.36%	**91.89%**	**-0.47%**	**75.1%**	20.1%
AMC [50]	ResNet-56	✗	92.80%	91.90%	-0.90%	50.0%	-
FPGM [49]		✗	93.59%	92.93%	-0.66%	**52.6%**	-
HRank [96]		✗	93.26%	93.17%	-0.21%	50.6%	**42.4%**
DSA [116]		✗	93.13%	92.91%	-0.22%	52.2%	-
SEP [6]		✓	93.12%	93.44%	+0.32%	50.0%	19.8%
UDSP (ours)		✓	93.12%	**93.78%**	**+0.66%**	50.1%	20.0%

where f' is the pruned model with around $p_p\hat{T}_p$ parameters. Here, we abuse notations Θ_d and $\mathcal{W}$ to represent weights after static pruning, and they are different from the original weights. We only modify the feature maps during fine-tuning, which takes advantage of mini-batch training. During the evaluation, we dynamically prune the channels. For both pruning and fine-tuning, we choose $R_r(x, y) = R_p(x, y) = \log(\max(x, y)/y)$. Typically, regular regression loss functions, like MAE and MSE, can be used for R_r and R_p, but they can hardly achieve target values for some architectures like MobileNet-V2. We insert g_s and g_d after the Conv-Bn-ReLU block and before the next convolution layer for pruning, which can accurately reflect the pruned model. The overall algorithm of our method is provided in Alg. 5. The whole process of our method is summarized in Fig. 18.

6.4 Experiments

6.4.1 Settings

In the experiment section, we call our method UDSP (**U**nified **D**ynamic and **S**tatic channel **P**runing). We use CIFAR-10 [82] and ImageNet [15] to verify the performance of our method, as

most previous pruning works use these datasets.

On CIFAR-10, we use CifarNet following several dynamic pruning works [6, 37]. Besides CifarNet, we also test our method on ResNet-56. For ImageNet, we evaluate our method on ResNets [46] and MobileNet-V2 [126]. p_p and p_r are used to decide how much FLOPs and parameters to be pruned. Detailed settings of p_p and p_r are provided in the supplementary materials. λ and γ in Eq. 6-5 and Eq. 6-6 are set to 2.0 and 0.1 separately for all models and datasets. τ in Eq. 6-1 and Eq. 6-2 is set to 0.4. Other implementation details are given in the supplementary materials.

6.4.2 CIFAR-10 Results

We present CIFAR-10 results in Tab. 24. For CifarNet, all comparison methods are dynamic. From Tab. 24, we can see that our method can outperform other comparison methods with similar pruned FLOPs. Compared to FBS, our method saves 27.9% of parameters (79.9% vs. 111% #Params compared to the original model) while achieving 1.02% improvements with Δ-Acc. SEP-A has similar parameter savings as our method, but the Δ-Acc is 0.37% lower than our method. SEP-B keeps all channels, and our method still outperforms it by 0.18% with Δ-Acc. Moreover, our method only uses 60.9% parameters of SEP-B.

We compare our method with both static and dynamic pruning methods on ResNet-56. All comparison methods reduce around 50% FLOPs. Our approach has similar pruning rates of FLOPs and parameters as SEP, but our method performs better than SEP by 0.34%. HRank achieves the best performance among static pruning methods. Static pruning methods prune more parameters compared to dynamic pruning methods, but the performance of our method is 0.87% higher than HRank in terms of Δ-Acc. In summary, our method achieves a better trade-off between storage costs and performance than SEP [6].

6.4.3 ImageNet Results

On the ImageNet dataset, we use ResNet-18, ResNet-34, ResNet-50, and MobileNet-V2 to evaluate the performance of different methods. All results are shown in Tab. 13. The results of other comparison baselines are directly adapted from their original paper following the common

practice.

ResNet-18. For static pruning methods, DSA [116] achieves the best performance. The Δ Top-1 accuracy of our method is 0.83% higher than DSA, and our method prunes 10.2% more FLOPs. This result suggests that dynamic pruning still has advantages when the model capacity is reduced to some extent. FBS and CGNN use additional parameters for dynamic pruning. Our method outperforms FBS and CGNN by 2.26% and 0.79% in terms of Δ Top-1 accuracy separately. In addition, our method prunes 11.5% more FLOPs than CGNN and saves 20% of parameters. Finally, our method is better than SEP by 0.75% in terms of Δ Top-1 accuracy, while both methods prune similar FLOPs and parameters.

ResNet-34. IE [112] performs better than other static pruning methods, but it prunes less FLOPs and parameters. Our method has similar parameters and performance as IE, but we can prune 27.7% more FLOPs than IE. Our method saves 20% parameters and performs better than CGNN by 0.71% in terms of Δ Top-1 accuracy, and both methods prune similar FLOPs.

ResNet-50. For ResNet-50, We compare several recent state-of-the-art pruning methods. Our method outperforms ResRe by 0.54% and 0.50% in terms of Top-1 and Δ Top-1 accuracy. The gap between other methods and our method is more obvious. 3DP explores pruning in 3 dimensions, which allows a more flexible trade-off. Our method is better than 3DP by 0.61% regarding Top-1 accuracy, indicating that our method can achieve similar flexibility. In addition, our method prunes most FLOPs, and we can also reduce storage costs to some extent (25.0% reduction). DepGrah and DTP are recently proposed static pruning methods, our UDSP still has a clear advantage when it comes to these baselines.

MobileNet-V2. AMC, MetaPruning and MobileNet-V2 0.75 all remove around 30% FLOPs. MetaPruning achieves the lowest accuracy lost. Our method prunes around 6% more FLOPs than MetaPruning, and performs better (0.52% and 0.44% higher with Top-1 and Δ Top-1 accuracy). Our method and GSS prune a similar amount of FLOPs, and the Δ Top-1 accuracy of our method is higher than GSS by 0.64%. In addition to FLOPs reduction, our method can also remove around 15.3% of parameters.

In summary, our method provides a larger model capacity compared to static pruning methods, and the storage costs are reduced compared to dynamic pruning methods. Moreover, our method achieves a better trade-off between storage costs and performance than SEP, indicating that inte-

grating dynamic and static pruning is important for pruning.

6.4.4 Analysis of Different Settings

To understand different design choices and hyper-parameter settings, we provide additional analysis in this section. In Fig. 19(a,b), we plot the loss value and model accuracy given different pruning settings. We can see that bi-level optimization outperforms iterative training with both accuracy and loss values, which suggests that integrating dynamic and static pruning is beneficial. We also show the difference between the finetuned model in Tab. 14, and we can draw similar conclusions.

In Fig. 19(c), we provide the accuracy after pruning (before fine-tuning) given different γ. A too-large γ usually hurts the performance, and γ around 0.1 provides relatively good results.

In Fig. 19(d), we fix $p_r = 0.5$, and plot the accuracy after pruning given different percentages of remaining parameters (p_p). We can see that the performance does not decrease a lot when we keep more than 75% of parameters. We further present results when pruning more parameters after finetuning in Tab 15. When pruning 30% of parameters (UDSP^2), the performance of our method does not decrease too much. However, there is a large performance drop when pruning 40% of parameters. Under this setting (UDSP^1), the parameter reduction of our method is similar to the static pruned model from HRank, and the dynamic flexibility is largely restricted. These observations suggest that, under the same FLOPs pruning rate, our method can maintain a good trade-off between dynamic flexibility and storage costs until the pruning rate for parameters is similar to static pruning methods.

In Fig. 20, we plot the value of regularization losses and model accuracy given different choices of λ. From the figure, it can be seen that our method is robust to different choices of λ. A lower λ can lead to a little better final performance, but the difference is small.

In Fig. 21, we plot the final architectures of ResNet-56 and CifarNet for our method and SEP. Our method tends to preserve more channels when the width of the original model changes. Later layers often have more dynamic flexibility, probably because they are less penalized by the FLOPs constraint $\mathcal{R}_r$. This figure also suggests that our method does not collapse into a single static solution. For both CifarNet and ResNet-56, SEP does not fully utilize the capacity of early layers, especially on ResNet-56. These results justify why our method can outperform SEP. This also

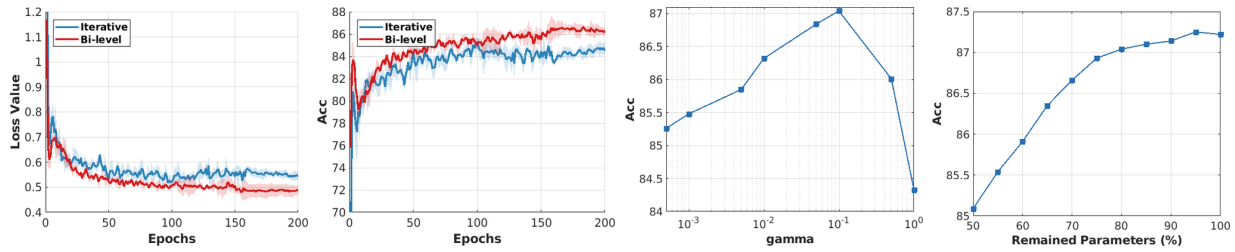
suggests that a more sophisticated interaction (bi-level optimization) between static and dynamic sub-networks is crucial to achieving good results.

6.5 Conclusion

In this paper, we study the problem of how to integrate dynamic and static pruning. We explicitly formulate the static and dynamic pruning problems as a new bi-level optimization task such that two types of models can complement each other. We further improve the efficiency of the cost matrix-vector product in the bi-level pruning problem. The superior performance of our method on CIFAR-10 and ImageNet datasets suggests that our method is a promising solution for integrating dynamic and static channel pruning.

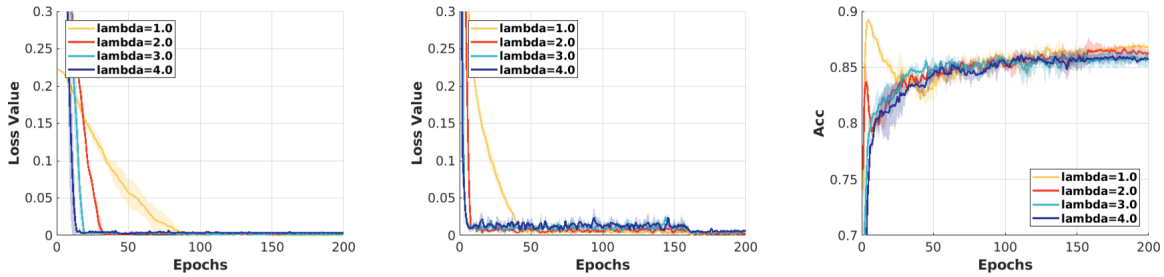
Table 13: Comparison of the accuracy changes (Δ Top-1), reduction in FLOPs, and the number of parameters of various channel pruning algorithms on ImageNet.

Method	Architectures	Dynamic	Pruned Top-1	Δ Top-1	$\downarrow$ FLOPs	$\downarrow$ #Params
AMC [50]	ResNet-18	✗	66.63%	-3.13%	50.0%	24.0%
FPGM [49]		✗	68.41%	-1.87%	41.5%	**28.0%**
DSA [116]		✗	68.61%	-1.11%	40.0%	-
FBS [37]		✓	68.17%	-2.54%	49.5%	-12.0%
CGNN [57]		✓	67.95%	-1.07%	38.7%	-
SEP [6]		✓	68.73%	-1.03%	48.5%	19.0%
UDSP (ours)		✓	**69.48%**	**-0.28%**	**50.2%**	20.0%
SFP [48]	ResNet-34	✗	71.84%	-2.09%	41.1%	-
FPGM [49]		✗	72.63%	-1.29%	41.5%	**28.9%**
IE [112]		✗	72.83%	-0.48%	22.3%	21.1%
CGNN[57]		✓	72.40%	-1.10%	**50.4%**	-
UDSP (ours)		✓	**72.91%**	**-0.39%**	50.0%	20.0%
SCOP[132]	ResNet-50	✗	75.26%	-0.89%	54.6%	51.8%
GFP[101]		✗	76.42%	-0.37%	51.0%	55.8%
3DP[136]		✗	75.90%	-0.25%	53.0%	50.0%
ResRe[17]		✗	75.97%	-0.12%	56.1%	-
DepGraph [20]		✗	75.97%	-0.12%	51.18%	-
DTP[93]		✗	75.55%	-0.58%	56.7%	-
UDSP (ours)		✓	**76.51%**	**+0.38%**	**58.4%**	25.0%
MobileNet-V2 0.75 [126]	MobileNet-V2	✗	69.80%	-2.00%	30.0%	**24.8%**
AMC [50]		✗	70.80%	-1.10%	30.0%	17.2%
MetaPruning [102]		✗	71.20%	-0.60%	30.9%	-
GSS [144]		✗	71.20%	-0.80%	36.0%	22.9%
UDSP (ours)		✓	**71.72%**	**-0.16%**	**36.6%**	15.3%



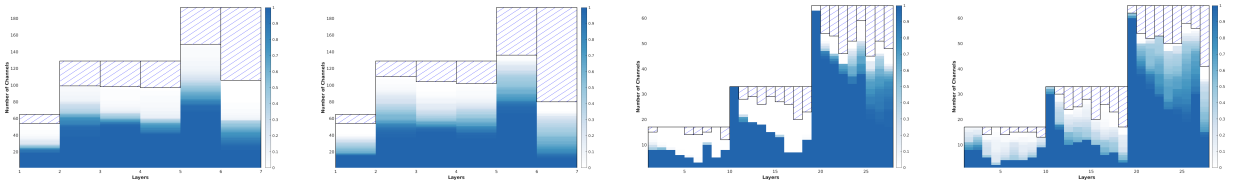
(a) Loss (b) Accuracy (c) Impact of γ (d) Impact of p_p

Figure 19: (a,b): Comparison of loss and accuracy given different training settings. Mean and variance are provided by running the experiment 3 times. (c) Impact of γ during the pruning process. (d) Impact of p_s during the pruning process. All results are obtained with ResNet-56 on CIFAR-10.



(a) $\mathcal{R}_p$ Loss (b) $\mathcal{R}_r$ Loss (c) Accuracy

Figure 20: The regularization losses and model accuracy given different choices of λ. Mean and variance are provided by running the experiment 3 times.



(a) CifarNet (SEP) (b) CifarNet (Ours) (c) ResNet-56 (SEP) (d) ResNet-56 (Ours)

Figure 21: The resulting architectures of ResNet-56 and CifarNet on CIFAR-10 with our method and SEP. We plot the probability of using each channel, and the probability is calculated on the whole test dataset. Channels with dashed lines are permanently removed.

Table 14: Comparisons between different pruning settings of our algorithm on ResNet-56 for the CIFAR-10 dataset.

Bi-level	Acc	Δ-Acc	↓ FLOPs	↓ #Params
✗	93.55%	+0.43%	50.0%	20.3%
✓	93.78%	+0.66%	50.1%	20.0%

Table 15: Comparisons given different pruning rates for #Params with ResNet-56 on CIFAR-10.

Settings	Base Acc	Acc	Δ-Acc	↓FLOPs	↓ #Params
UDSP[1]	93.12%	93.32%	+0.20%	50.0%	40.0%
UDSP[2]	93.12%	93.64%	+0.52%	50.1%	30.0%
UDSP[3]	93.12%	**93.78%**	**+0.66%**	50.1%	20.0%

7.0 Conclusion

In this dissertation, we propose several new methods to compress CNNs. We first build a differentiable optimization framework for pruning CNNs given certain resource constraints. By turning the deterministic discrete gate into the stochastic discrete gate, moreover, our method can explore a larger search space of sub-networks. To further enlarge the space, we introduced the symmetric weight decay on the gate parameters inspired by the fact that regularization loss can be regarded as weight decay. Our method also benefits from the exact estimation of sub-networks' outputs because of a combination of the precise placement of gates and the discrete setting. We then incorporated accuracy as the additional supervision signal where we studied how to simultaneously achieve low loss value and high accuracy when searching for sub-networks. By using an episodic memory module and re-sampling techniques, we are able to train a performance prediction network in-place during pruning, which also saves computational resources. By utilizing information from the classification loss and performance maximization, our method is able to find good sub-networks during pruning.

In previous works, width and channel importance are entangled during the pruning process. Such a design is straightforward and easy to use, but it restricts the potential search space during the pruning process. To overcome this limitation, we propose to prune neural networks by disentangling width and importance. To achieve such a disentanglement, we propose to relax the hard constraint used in previous methods to a soft regularization term, allowing independent parametrization of width and importance. We also relax hard top-k to soft top-k with the smooth-step function. We further use an importance score generation network and a width network to facilitate the learning process. Furthermore, based on the previous observation that there is a huge gap between models before and after model compression, we proposed partial regularization to produce a better sub-network for network pruning. Specifically, our method uses AGN to guide partial regularization across the training process. We further provide an efficient way to update model weights through proximal gradients. With these designs, partial regularization effectively reduces the gap between the sub-network within the full model and the pruned model. Our method then starts from a better sub-network, thus resulting in a better final pruned model. On top of

the previous static pruning settings, We propose a new bi-level optimization based model to naturally integrate the static and dynamic channel pruning. By doing so, our method enjoys benefits from both sides, and the disadvantages of dynamic and static pruning are reduced. Our method can be further applied to tasks like learning kernel sizes [26], generative models [27], federated learning [34], training sparse model from scratch [141], reinforcement learning with partial regularization [28], *etc.*

All these methods achieved or surpassed the state-of-art result at the current time, showing the incredible strength of our novel pruning algorithms. In addition, all these methods require minimal additional computational resources to find the proper sub-network, which shows the efficiency of our methods. The efficiency and effectiveness of our methods show great potential to be used on large foundational vision, vision-language, and language models.

A.1 DMC: Derivation of Regularization Gradient

The detail derivation of regularization gradient is given below if $\hat{T} \neq pT$:

$$\begin{aligned}
\frac{\partial \mathcal{R}_{\log}}{\partial \theta_{l,c}} &= \frac{1}{|\hat{T}-pT|+1} \cdot \frac{\hat{T}-pT}{|\hat{T}-pT|} \cdot \frac{\partial \sum_{l=1}^{L} (\text{FLOPs})(\widehat{\text{FLOPs}})_l}{\partial \theta_{l,c}} \\
&= \frac{1}{|\hat{T}-pT|+1} \cdot \frac{\hat{T}-pT}{|\hat{T}-pT|} \\
&\cdot k_l^2 \cdot \frac{\mathbf{1}^T \mathbf{g}_{l-1}}{\mathcal{G}_l} \cdot w_l \cdot h_l \cdot \frac{\partial \mathbf{1}^T \mathbf{g}_l}{\partial \theta_{l,c}} \\
&= \eta_l \cdot \frac{1}{|\hat{T}-pT|+1} \cdot \frac{\hat{T}-pT}{|\hat{T}-pT|} \cdot \frac{\partial g(\theta_{l,c})}{\partial \theta_{l,c}} \\
&= \eta_l \cdot \frac{1}{|\hat{T}-pT|+1} \cdot \frac{\hat{T}-pT}{|\hat{T}-pT|},
\end{aligned}$$

where $\eta_l = k_l^2 \cdot \frac{\mathbf{1}^T \mathbf{g}_{l-1}}{\mathcal{G}_l} \cdot w_l \cdot h_l$ .The result of the second line is due to the definition of $(\text{FLOPs})(\widehat{\text{FLOPs}})_l$ and $(\text{FLOPs})(\widehat{\text{FLOPs}})_l = k_l \cdot k_l \cdot \frac{\mathbf{1}^T \mathbf{g}_{l-1}}{\mathcal{G}_l} \cdot \mathbf{1}^T \mathbf{g}_l \cdot w_l \cdot h_l$. The result of the fourth line is because of STE: $\frac{\partial g(\theta_{l,c})}{\theta_{l,c}} = 1$, if $\theta_{l,c} \in [0,1]$. If $\hat{T} = pT$, then $\mathcal{R}_{\log} = 0$, and 0 can be used as the subgrident of this point. Thus, we have the sub-gradient given in the paper:

$$\frac{\partial \mathcal{R}_{\log}}{\partial \theta_{l,c}} = \begin{cases} \eta_l \cdot \frac{1}{|\hat{T}-pT|+1} \cdot \frac{\hat{T}-pT}{|\hat{T}-pT|}, & \text{if } \hat{T} \neq pT \\ 0, & \text{if } \hat{T} = pT \end{cases}$$

A.2 Choice of p for DMC

Table 16: Choice of p for ImageNet models. p is the **remained** FLOPs divided by the total FLOPs

Architecture	ResNet-34	ResNet-50	ResNet-101	MobileNetV2
p	0.55	0.38	0.42	0.50

In this section, we will give the detailed number of p. In a CNN, we do not prune the first layer, the last layer, and residual connections in ResNet. As a result, the actual remaining FLOPs may not equal to p. We list the choice of p for ImageNet models in Tab. 16. For CIFAR-10 models, the unpruned FLOPs are quite small, thus, p is the same as the remained fraction of FLOPs.

A.3 DMC: Acceleration

The CPU run time of different models is shown in Tab. 17. The input is a mini-batch of 4 images.

Table 17: CPU time for different ImageNet Models. The time is measured in milliseconds.

Architecture	ResNet-34	ResNet-50	ResNet-101	MobileNetV2
Original Time (ms/batch)	113.5	195.7	331.5	106.8
Pruned Time (ms/batch)	81.4	126.2	205.4	67.5
Improvement (%)	28.3%	35.5%	38.0%	36.8%
Pruned FLOPs (%)	43.4%	55.0%	56.0%	46.0%

A.4 DMC: Discussion of Difference between Proposed Method and Trainable Gate [77]

In trainable gate (TG) [77], they propose to turn the non-differentiable gate into a differentiable gate inspired by [41]. They add a perturbation to the gate function to make it differentiable. Our method, on the other hand, does not modify the gate function and uses STE to handle gradient calculation. Thus, the approach of making the gate differentiable is different. Moreover, in their framework, the gate calculation is deterministic. Consequently, they can not sample sub-networks as we do. Their work also applies a form of constraint to limit the resources of the pruned neural network.

A.5 DMC: More Results

In Fig. 23, we plot the accuracy and classification loss on the test dataset given two settings: using performance maximization (PN) and differential gates (DG). From this figure, we can see that PN can outperform DG on test accuracy during pruning, but the difference in classification loss is much smaller. In summary, our method can achieve lower classification loss and higher

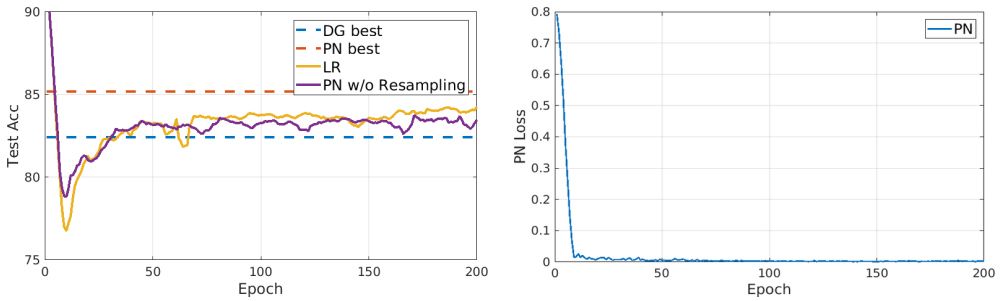


Figure 22: *Left*: More Pruning Settings. *Right*: PN Loss. Results are from CIFAR-10 with ResNet-56.

accuracy. At some points, even if the classification losses are close, the difference in accuracy can be larger than 1%, indicating that the classification loss is not always a good proxy for accuracy.

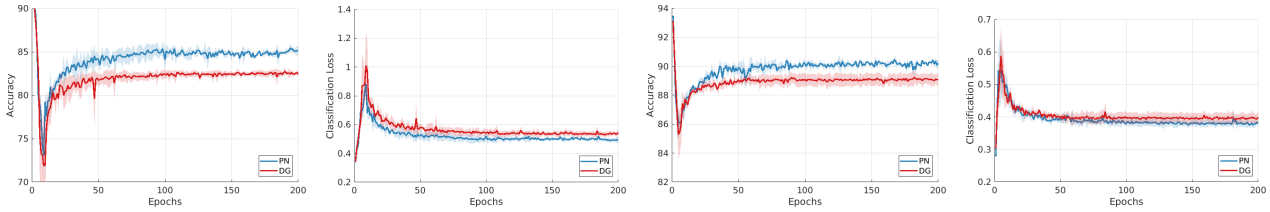
To verify the effectiveness of GRU, we add a comparison baseline for the linear regression (LR) model in Fig. 22-*Left*. Using PN is better than LR, we hypothesize that LR may omit hierarchical information of different layers while PN can capture it using GRU. The training loss of the PN is shown in Fig. 22-*Right*, which is quite stable.

A.6 Choice of p for NPPM

The choice of p can be analytically calculated. Let T_{all} be the overall FLOPs of a CNN, and T_{total} is the total prunable FLOPs. Suppose we want to remove 50% of FLOPs, then $pT_{\text{total}} = 0.5T_{\text{all}}$, and $p = 0.5\frac{T_{\text{all}}}{T_{\text{total}}}$. The detailed p is listed in Tab. 26.

A.7 NPPM: Orthogonal Projection of Gradients

Recall that $g^i_{\mathcal{L}} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}_i}$ represents the gradient vector from the classification loss of ith layer, and let $g^i_{\text{P}} = \frac{\partial \log(\frac{1}{\text{PN}_{(\mathbf{a})}})}{\partial \mathbf{w}_i}$ be the gradient vector from performance maximization. The projection of g^i_{P}



(a) (b) (c) (d)

Figure 23: (a, b): Accuracy and classification loss given different settings with ResNet-56. (c, d): Accuracy and classification loss given different settings with MobileNetV2. Both experiments are done on CIFAR-10. Shaded areas represent variance from 5 runs.

onto $g_{\mathcal{L}}^i$ is $\bar{g}_{\mathrm{P}}^i = \frac{(g_{\mathcal{L}}^i)^T(g_{\mathrm{P}}^i)}{\|g_{\mathcal{L}}^i\|^2} g_{\mathcal{L}}^i$, and

$$\hat{g}_{\mathrm{P}}^i = g_{\mathrm{P}}^i - \frac{(g_{\mathcal{L}}^i)^T(g_{\mathrm{P}}^i)}{\|g_{\mathcal{L}}^i\|^2} g_{\mathcal{L}}^i, \tag{0-1}$$

is orthogonal to $g_{\mathcal{L}}^i$. Thus, we have $g_{\mathrm{P}}^i = \bar{g}_{\mathrm{P}}^i + \hat{g}_{\mathrm{P}}^i$.

A.8 NPPM: Structure of the Performance Prediction Network

The structure of the performance prediction network is shown in Tab. 21, where FC is a fully-connected layer, and Avg averages of the outputs of all steps of GRU and $i = 1, \cdots, L$.

A.9 DDNP: Choice of p Given Different Datasets and Architectures.

We present the choice of p for all experiments in Tab 26. For ResNet-50, p is set to 0.38 when we prune 55.0% FLOPs, and p is set to 0.31 when we prune 62.0% FLOPs (results in Tab. 23)

Table 18: Choice of p.

Dataset	Architecture	p
CIFAR-10	ResNet-56	0.50
	MobileNetV2	0.55
ImageNet	ResNet-34	0.50
	ResNet-50	0.38
	ResNet-101	0.42
	MobileNetV2	0.63
	ShuffleNetV2+	0.70

A.10 DDNP: Architectures of g_s and g_k.

The architecture of g_s is shown in Tab. 21. The forward calculation of g_s is:

$$\begin{aligned} o_l, h_l &= \text{GRU}(x_{sl}, h_{l-1}) \\ \bar{s}^l &= \text{FC}_l(o_l), \end{aligned} \tag{0-2}$$

$\bar{s}^l$ is the importance score before the final output function. In most experiments, we use the sigmoid function to produce the final importance score: $s^l = \text{sigmoid}(\bar{s}^l)$. The experiments in Figure 4 (d,b) of the main text employees two other types of output functions, and they are absolute value functions (used in weight pruning papers [127]): $s^{AF} =\mid \bar{s} \mid$ and square function: $s^{SF} = \bar{s}^2$.

We also present the architecture of g_k in Tab. 22. Here, we use $\bar{k}$ to represent outputs before normalization. Given a certain layer l, the final k_l is obtained by:

$$k_l = \text{sigmoid}(\bar{k}_l + \text{b}), \tag{0-3}$$

where b is a positive constant to ensure we start pruning from a whole network, and $b = 3.0$ for all experiments. This is also discussed in section 5.1 of the main text.

Table 19: The structure of PN used in our method.

Inputs a_i, i=1,$\cdots$, L
FC$_i$(C_i, 16), BN,ReLU
GRU(16, 16),Avg
FC(16, 1), sigmoid

Table 20: Choice of p for different models. p is the **remained** FLOPs divided by the total FLOPs.

Dataset	CIFAR-10		ImageNet			
Architecture	ResNet-56	MobileNet-V2	ResNet-34	ResNet-50	MobileNet-V2	MobileNet-V3 small
p	0.48	0.56	0.55	0.38/0.31	0.67	0.75

A.11 DDNP: Additional Experiments

The test accuracy and normalized resource loss given different λ during training are shown in Fig. 24 (a,b). These figures show that a small λ may hinder the pruning process since the pruned model can not reach target FLOPs. Using a larger λ can solve this problem.

We also study the impact of ρ in Fig. 24 (c). We can see that the test accuracy of the pruned model is pretty low when $\rho = 0$, since learning of importance and width are completely disentangled. The performance of pruning also increases when we increase ρ. But if we use a too large ρ, the performance decreases, indicating that entangling width and importance reduce pruning flexibility.

In Fig. 24 (d), we present more settings for pruning. "Fix S" means that importance s is fixed, and only the second term and third term of Eq. (9) are used. In this setting, data is no longer used for pruning, and L_1 norm is used as the channel importance. "Fix k" represents that width k is fixed, and all k_l are set to 0.5 since they are no longer learnable. "Channel-wise-ks" means that we use scalar parameterization (one variable per-channel and per-layer) for both width and importance. From the figure, we can see that parameterization for both channel importance and

Table 21: The architecture of g_s used in our method.

Inputs x_{sl}, $l = 1, \cdots, L$
GRU(64,128), WeightNorm, ReLU
FC$_l$(128,C_l), WeightNorm, $l = 1, \cdots, L$
Outputs $\bar{s}^l$, $l = 1, \cdots, L$

Table 22: The architecture of g_k used in our method.

Inputs x_k,
FC$_l$(32, 64), WeightNorm, ReLU
FC$_l$(64,L), WeightNorm
Outputs $\bar{\mathbf{k}} = [\bar{k}_1, \cdots, \bar{k}_L]$

width can impact the pruning results, and the parameterization of channel importance has a larger impact. Moreover, "Fix S" or "Fix k" largely restrict the learning flexibility.

In Fig. 24 (e), we present the difference between learnable inputs and fixed inputs for g_k and g_s. For fixed inputs, we randomly generate x_s and x_k (in Tab. 21 and Tab. 22) before pruning and fix them during pruning. For learnedable inputs, we just randomly initialize x_s and x_k and include them as learnable parameters during pruning. Learnable inputs and fixed inputs do not have large differences.

In Tab. 23, we present additional results for ResNet-50 when the FLOPs pruning rate is larger (around 62%). Compared with state-of-the-art methods like CHIP [131] and CC [92], our method outperforms them with similar FLOPs pruning rates. The advantage of our method compared to HRank [96] is more obvious.

At last, we measure the additional costs brought by the importance and width generation networks with ResNet-56 on CIFAR-10 and ResNet-50 on ImageNet. On CIFAR-10, we measure the running time by averaging the time costs of the last 10 epochs. For ResNet-56, with scalar parameterization, the running time is 0.136 second/iteration. With importance and width generation networks, the time the running time is 0.143 second/iteration. On ImageNet, we measure the running time by averaging the time costs for the last 5 epochs. with scalar parameterization, the running time is 1.08 second/iteration. With importance and width generation networks, the running time is 1.15 second/iteration. From these results, we can see that the additional costs are trivial.

A.12 SANP: Implementation Details

We list more details of the implementation in this section. To train the base model on CIFAR-10, we follow the standard training PyTorch training examples. The models are trained for 200 epochs, and the learning rates are 0.1, 0.01, and 0.001 for the first 100 epochs, 100 to 150 epochs, and 150 to 200 epochs. We select SGD as the optimizer with a momentum of 0.9 and weight decay of 10^{-4}. After pruning, we finetune the model for 160 epochs using similar optimization hyperparameters. The learning rates for finetuning are 0.1, 0.01, and 0.001 for the first 80 epochs, 80 to 120 epochs, and 120 to 160 epochs.

For ResNet models on ImageNet, we train the model for 90 epochs following the standard PyTorch ImageNet training script. The learning rates are 0.1, 0.01, and 0.001 for the first 30 epochs, 30 to 60 epochs, and 60 to 90 epochs. Still, SGD is selected as the optimizer with momentum of 0.9 and weight decay of 10^{-4}. For MobileNet-V2, we train the model for 150 epochs with the cos-annealing learning rate and a start learning rate of 0.045, and weight decay 4×10^{-5} as mentioned in their original paper [126]. For each model on ImageNet, we fine-tune them for 100 epochs. The learning rate schedule and the start learning rate are the same as the training of the based model. Except that when fine-tuning ResNets, we set the learning rate to 0.0001 for 90 to 100 epochs. We also list p and E_{start} in Tab. 24. As mentioned in the paper, we set E_{start} to around 20% of the total training time.

As we discussed in the paper, AGN is composed of dense layers and GRUs, and now we present the architecture of AGN in Tab. 25. In Tab. 25, $z \in \Re^{L \times 32}$ is the input to the AGN, and z is initially sampled from a normal distribution, and it is then fixed during training. Outputs o_l are continuous values. We use the following equation to covert it into $\mathbf{v}_l$:

$$\mathbf{v}_l = \text{round}(\text{sigmoid}((o_l + g + b)/\tau)), \tag{0-4}$$

where sigmoid$(\cdot)$ is the sigmoid function, round$(\cdot)$ is the rounding function, g is sampled from Gumbel distribution ($g \sim$ Gumbel$(0, 1)$), b is a constant value to make sure pruning starts from the whole model, and τ is the temperature hyper-parameter. As shown in Eq. 0-4, straight-through Gumbel-Sigmoid [73] are used to produce the final binary vector $\mathbf{v}$. The function of AGN can

also be understood as translating z into the final architecture vector $\mathbf{v}$. For all experiments, we set $\tau = 0.4$ and $b = 3.0$. Note that the additional training costs of our method are trivial compared to the original training process, mainly because we only train AGN in a small sub-dataset of the whole dataset. Take MobileNet-V2 as an example, its average model training time is 976 seconds per epoch, and the average AGN training time is 95 seconds per epoch. The overhead is **around 10%** of the original training time.

A.13 SANP: Regularization with Blocks

Recent CNN designs often use a block as a building block. In the paper, we do not explicitly talk about this setup to simplify notations. Usually, we group $\mathbf{v}_l$ based on the definition of blocks, and L is the total number of unique $\mathbf{v}_l$, and it could be different from the actual number of layers. To avoid conflicts, we rewrite $\mathbf{v}_l$ as $\mathbf{v}_k$, and $k = 1, \cdots, K$, and K is the total number of unique $\mathbf{v}_k$ (a single $\mathbf{v}_k$ can be used for multiple layers along different dimension). Let us use the basic block from ResNets as an example. With this setting, $\mathcal{R}_w$ can be written as:

$$R_w(\mathcal{W}) = \sum_{j=1}^{B} \sum_{i \in S_k} \frac{\hat{N}_k}{\hat{N}} (\|\mathcal{W}_j^{\text{upper}}{}_{[i,:,:,:]}\|_{\text{GL}} + \|\mathcal{W}_j^{\text{lower}}{}_{[:,i,:,:]}\|_{\text{GL}}), \tag{0-5}$$

where k is the corresponding index of $\mathbf{v}_k$ for the jth block, and B is the total number of blocks. For the upper layer in the jth block, the regularization is applied on the output dimension, and the regularization is applied on the input dimension for the lower layer in the jth block. The formulation of Eq. 0-5 can be easily extended to other block types, such as the bottleneck block in ResNets and the inverted residual block in MobileNet-V2.

A.14 SANP: Different Choices of γ

In this section, we want to discuss the impact of γ and how it affects the performance of the whole model. We plot the related results in Fig. 25. From the figure, we can see that when we increase γ, we can get a better sub-network during the training process. However, it also negatively

affects the full model performance, as shown in Fig. 25b. The full model accuracy provides the baseline before pruning. Δ-Acc often increases when we have a better sub-network, but if the baseline accuracy is too low, the final fine-tuned accuracy will also be worse. On CIFAR-10, this happens when we use $\gamma \geq 1 \times 10^{-3}$ (a better Δ-Acc but a worse final accuracy). On the ImageNet dataset, we also find that when $\gamma = 5 \times 10^{-4}$, we can get a similar baseline accuracy with better sub-network accuracy. We can probably use a larger γ to get better Δ Top-1 accuracy, but it will create weaker baselines, which is not a good practice for fair comparisons.

A.15 SANP: Stability of Sub-network Architectures

In our paper, we apply soft regularization when training model weights. One problem is that if the sub-network architecture changes frequently, then most weights will be penalized, which brings a trivial difference between penalizing all weights. To investigate this problem, we plot the hamming distance between the sub-network architecture of two consecutive epochs in Fig. 26. We can see that the hamming distance will be smaller than 0.05 after around 10 epochs, and the hamming distance will continuously decrease after 20 epochs. This observation suggests that the sub-network architecture becomes more and more stable after 20 epochs. Even if some weights are wrongly penalized at the beginning, they still have enough time to recover their magnitudes.

A.16 UDSP: Implementation Details

We train ResNet-56 and CifarNet on CIFAR-10 from scratch following pytorch [118] examples. For ImageNet models, we can directly use pre-trained models from Pytorch, since our method is able to prune any pre-trained models.

To improve efficiency, we only use part of the dataset for pruning. We randomly sample 5,000 and 50,000 images from CIFAR-10 and ImageNet as D_{prune} in Alg. 5. Adam [79] optimizer is used to train both Θ_s and Θ_d, and the training lasts for 200 epochs with mini-batchsize 256. The start learning rate and weight decay are set to 10^{-3} and 10^{-4}, and the learning rate is decayed to 10^{-4}

at epoch 160. We initialize all Θ_s to 3.0 to ensure most initial g_s are 1.0, which means that static pruning starts from the whole network.

On CIFAR-10, we finetune the model for 160 epochs by using the Adam optimizer with a start learning rate 10^{-3}. The learning rate is changed to 10^{-4} at epoch 80, and it is further reduced to 10^{-5} at epoch 120. Following storage efficient pruning [6], we continue to use the Adam optimizer on ImageNet models. After pruning, we finetune ResNet models for 100 epochs with a start learning rate 10^{-3}. The learning rate is then decayed to 10^{-4} at epoch 30, and it is further decayed to 10^{-5} and 10^{-6} at epoch 60 and epoch 90. For MobileNet-V2, we also use the Adam optimizer and finetune it for 100 epochs. We use the cos-annealing learning rate scheduler following their original setting [126]. The mini-batch size and weight decay are 256 and 10^{-4} for both CIFAR-10 and ImageNet models. All codes are implemented with pytorch [118]. The experiments are conducted on a machine with 4 Nvidia Tesla P40 GPUs.

A.17 UDSP: Negative Impacts of Joint Training

In section 3.3 of the main text, we argue that joint training of dynamic and static pruning will interfere with each other. In Fig. 27, we present the comparison results between joint training and iterative training. We can see that joint training lacks exploration during learning and its performance is lower than the iterative baseline.

A.18 UDSP: Derivation of Gradients w.r.t Θ_d from Adam

In Eq. 9 of our paper, we provide the gradients w.r.t Θ_d when updating Θ_s with SGD, since it's simple and easy to follow. In practice, SGD can hardly achieve satisfactory performance when dealing with discrete values. As a result, we use Adam to update Θ_s. As a result, we will show how to calculate the gradients of Θ_d under the Adam optimizer. We show the update rule of Θ_s in Al. 6, we omit timestep t of Θ_s to simplify notations. We focus on the first term in Eq. 9, and the

Algorithm 6: Update Θ_s with Adam

Input: $\eta, \beta_1, \beta_2 \in (0, 1]$, $\epsilon \geq 0$: learning rate and decay rate for ADAM.

Initialize $m_0, n_0, t = 0$

Update rule at step t:

$m_t = \beta_1 m_{t-1} + (1 - \beta_1)(\nabla_{\Theta_s} \mathcal{L}(\Theta_s, \Theta_d) + \lambda \nabla_{\Theta_s} \mathcal{R}_p)$

$n_t = \beta_2 n_{t-1} + (1 - \beta_2)(\nabla_{\Theta_s} \mathcal{L}(\Theta_s, \Theta_d) + \lambda \nabla_{\Theta_s} \mathcal{R}_p)^2$

$\hat{m}_t = m_t / (1 - \beta_1^t)$

$\hat{n}_t = n_t / (1 - \beta_2^t)$

$\Theta_s' = u(\Theta_s, \eta) = \Theta_s - \eta \hat{m}_t / (\sqrt{\hat{n}_t} + \epsilon)$

gradient w.r.t Θ_d is:

$$
\begin{aligned}
& \nabla_{\Theta_d} \mathcal{L}(\Theta_s^*, \Theta_d) \\
\approx & \nabla_{\Theta_d} \mathcal{L}(\Theta_s - \eta \hat{m}_t / (\sqrt{\hat{n}_t} + \epsilon), \Theta_d) \\
= & \nabla_{\Theta_d} \mathcal{L}(\Theta_s', \Theta_d) - \eta \nabla^2_{\Theta_d, \Theta_s} \mathcal{L}(\Theta_s, \Theta_d)((\frac{\hat{\beta}_1}{2(\sqrt{\hat{n}_t} + \epsilon)} \\
& - \frac{\hat{\beta}_2 \hat{m}_t (\nabla_{\Theta_s} \mathcal{L}(\Theta_s, \Theta_d) + \lambda \nabla_{\Theta_s} \mathcal{R}_p)}{2(\hat{n}_t + \epsilon \sqrt{\hat{n}_t})^2}) \nabla_{\Theta_s'} \mathcal{L}(\Theta_s', \Theta_d)),
\end{aligned}
\tag{0-6}
$$

where $\hat{\beta}_1 = \frac{1-\beta_1}{1-\beta_1^t}$, $\hat{\beta}_2 = \frac{1-\beta_2}{1-\beta_2^t}$. The derivation in Eq. 0-6 is a little bit complicated compared to the SGD update, but it is still the result of the chain rule.

A.19 UDSP: Calculation of Jacobian Matrix

Recall that we need to calculate $\nabla_{\theta_s^i} g_s^i \cdot \nabla_{\theta_d^i} g_d^i$ in Eq. 10. Let us first focus on the element-wise function $g_s^i = \lfloor v_s^i \rceil$ and $v_s^i = \sigma((\theta_s^i + \mu)/\tau)$, and we have:

$$
\nabla_{\theta_s^i} g_s^i = \mathbf{I} \nabla_{\theta_s^i} v_s^i = \nabla_{\theta_s^i} v_s^i, \tag{0-7}
$$

where the identity matrix $\mathbf{I}$ comes from using straight-through estimator, $\nabla_{\theta_s^i} v_s^i = \text{diag}(a_s^i)$, and $a_s^i = \frac{1}{\tau} \sigma((\theta_s^i + \mu)/\tau)(1 - \sigma((\theta_s^i + \mu)/\tau))$.

To simplify derivation, we assume linear transformation is used in the routing function $h(\cdot)$ instead of SE, and $\theta_d^i \in R^{C_i \times C_{i-1}}$. Under this setting, we have $h(\mathcal{F}_{i-1}; \theta_d) = \theta_d \bar{\mathcal{F}}_{i-1}$, and $\bar{\mathcal{F}}_{i-1} = \text{GAP}(\mathcal{F}_{i-1})$ is the result of global average pooling (GAP). We also let $z_i = h(\mathcal{F}_{i-1}; \theta_d)$ Similarly, $\nabla_{\theta_d^i} g_d^i$ can be calculated as follows:

$$\nabla_{\theta_d^i} g_d^i = \nabla_{z_i} v_s^i \nabla_{\theta_d^i} z_i, \tag{0-8}$$

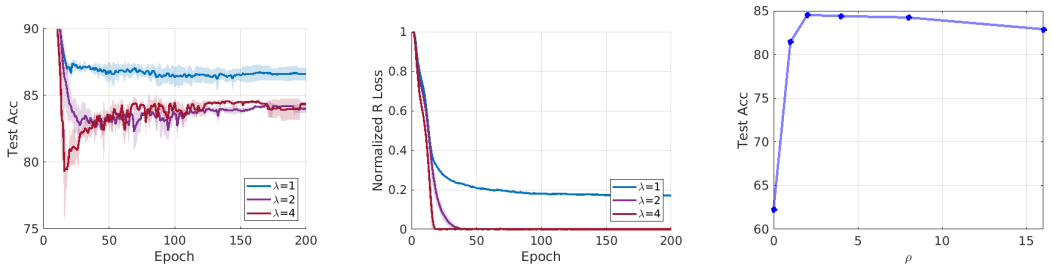
where $\nabla_{z_i} v_s^i = \text{diag}(a_d^i)$, and $a_d^i = \frac{1}{\tau}\sigma((z_i + \mu)/\tau)(1 - \sigma((z_i + \mu)/\tau))$. The last term $\nabla_{\theta_d^i} z_i \in R^{C_i \times C_i \times C_{i-1}}$ (mini-batch dimension is omitted) is the Jacobin matrix of matrix-vector product w.t.r to θ_d^i. $(\nabla_{\theta_d^i} z_i)_{[:,j,k]} = \left[0 \cdots (\bar{\mathcal{F}}_{i-1})_k \cdots 0\right]^\intercal$, and $(\bar{\mathcal{F}}_{i-1})_k$ is at the jth element of the vector. The above result is obtained by applying the chain rule on the matrix-vector product. As a result, the calculation of $\nabla_{\theta_d^i} z_i$ is just rearranging $\bar{\mathcal{F}}_{i-1}$ to the right position, which is not expansive.

We first vectorize $\nabla_{\theta_d^i} g_d^i \in R^{C_i \times C_i C_{i-1}}$, and we have $\nabla_{\theta_s^i} g_s^i \in R^{C_i \times C_i}$. The computation of $(\nabla_{\theta_s^i} g_s^i \cdot \nabla_{\theta_d^i} g_d^i) \in R^{C_i \times C_i C_{i-1} \times C_i}$ can be written as:

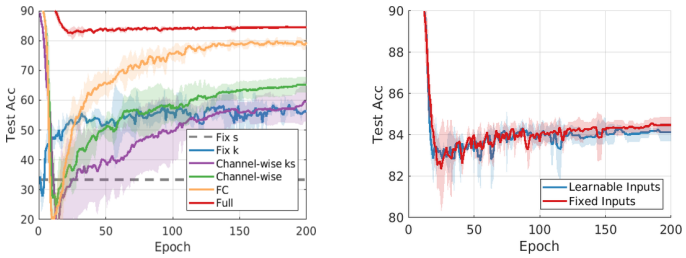
$$(\nabla_{\theta_s^i} g_s^i \cdot \nabla_{\theta_d^i} g_d^i)_{[p,:,:]} = ((\nabla_{\theta_s^i} g_s^i)_{[p,:]}^\intercal (\nabla_{\theta_d^i} g_d^i)_{[p,:]})^\intercal. \tag{0-9}$$

A.20 UDSP: Selections of p_r and p_p

We present the choices of p_r and p_p in Tab. 26. The choices of p_r and p_p are not hard to calculate. Let T_p^{all} be the number of all parameters given a CNN, and $\hat{T}_p$ is the total number of prunable parameters. If we want to remove 20% of parameters, then $p_p \hat{T}_p = 0.8 T_p^{\text{all}}$. Finally, $p_p = 0.8 T_p^{\text{all}} / \hat{T}_p$. Similar calculations can be applied on p_r.



(a) Test Acc given λ (b) $\mathcal{R}$ Loss given λ (c) Effect of ρ



(d) More settings (e) Different inputs

Figure 24: (a,b): Test accuracy and normalized $\mathcal{R}$ loss during training given different λ. (c): Test accuracy given different slection of ρ. (d): Impact to pruning with additional settings. (e): Learnable inputs vs. fixed inputs. We run each setting three times and use shaded areas to represent variance. All experiments are done on CIFAR-10 with ResNet-56.

Method	Base/Pruned Top-1	Base/Pruned Top-5	Δ Top-1	Δ Top-5	$\downarrow$ FLOPs
HRank [96]	76.15%/ 71.98%	92.87%/91.01%	-4.17%	-1.86%	62.1%
CC [92]	76.15%/ 74.54%	92.87%/92.25%	-1.61%	-0.62%	62.7%
CHIP [131]	76.15%/ 75.26%	92.87%/92.53%	-0.89%	-0.34%	62.8%
DDNP (ours)	76.13%/**75.56%**	92.86%/**92.68%**	$\mathbf{-0.57\%}$	$\mathbf{-0.18\%}$	62.0%

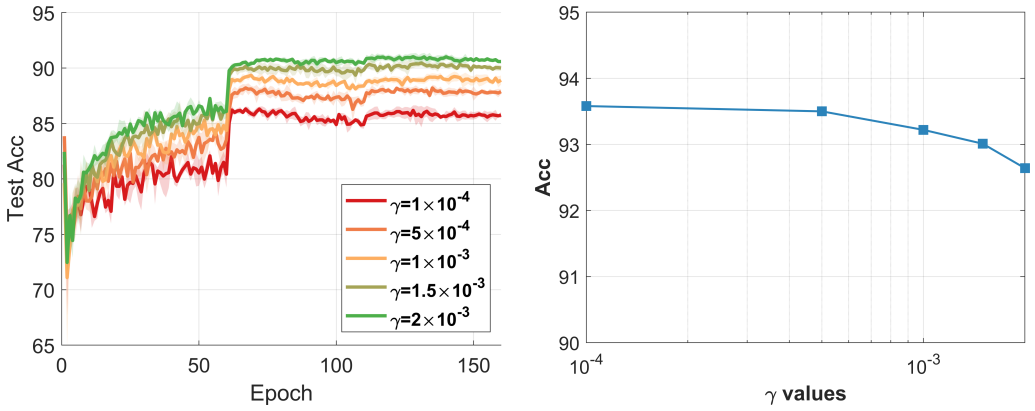
Table 23: Additional results with ResNet-50 on ImageNet when pruning more FLOPs.

Table 24: Choices of p and E_{start} for different models.

Architecture	Dataset	p	E_{start}
ResNet-56	CIFAR-10	0.48	40
MobileNet-V2		0.54	40
ResNet-34	ImageNet	0.54	18
ResNet-50		0.37	18
ResNet-101		0.41	18
MobileNet-V2		0.65	30

Table 25: The architecture of AGN.

Input z
GRU(32,64)$\rightarrow$ LayerNorm$\rightarrow$ GeLU
Dense$_l$(64, C_l)$\rightarrow$Outputs o_l, $l = 1, \cdots, L$



(a) (b)

Figure 25: (a) The sub-network performance during training with different γ. (b) Performance of the whole model given γ. All experiments are conducted on CIFAR-10 with ResNet-56.

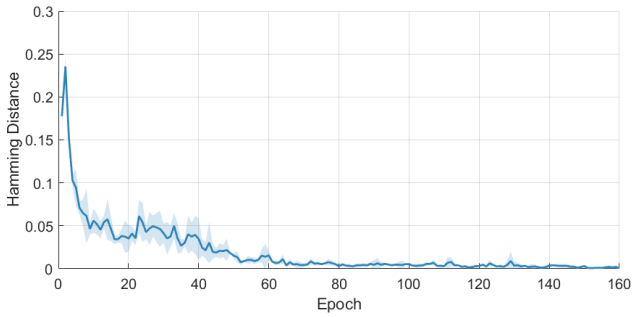
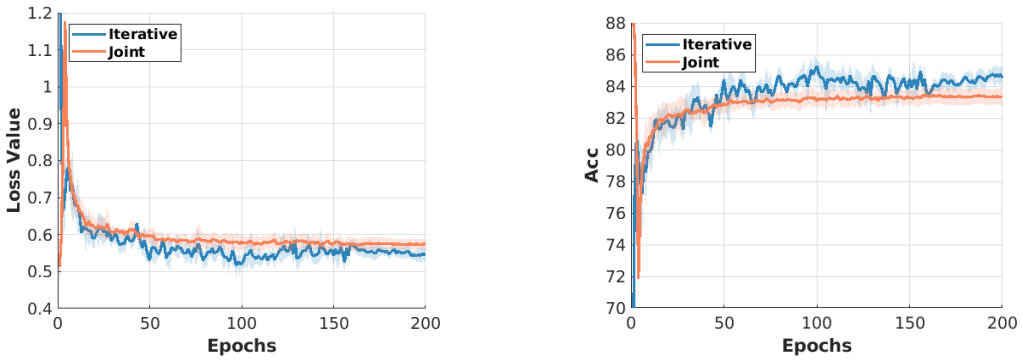


Figure 26: Hamming Distance between sub-network architecture during the training process. This experiment is conducted on CIFAR-10 with ResNet-56.



(a) Loss

(b) Accuracy

Figure 27: (a,b): Comparison of loss and accuracy given joint and iterative training. Mean and variance are provided by running the experiment 3 times.

Table 26: Choice of p_r and p_p.

Dataset	Architecture	p_r	p_p
CIFAR-10	ResNet-56	0.50	0.80
	CifarNet	0.75	0.50
ImageNet	ResNet-18	0.45	0.75
	ResNet-34	0.45	0.75
	ResNet-50	0.36	0.70
	MobileNet-V2	0.60	0.80

Bibliography

[1] Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*, 2018.

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[4] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*, pages 527–536. PMLR, 2017.

[5] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[6] Jianda Chen, Shangyu Chen, and Sinno Jialin Pan. Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 14747–14758. Curran Associates, Inc., 2020.

[7] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294, 2015.

[8] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11030–11039, 2020.

[9] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1518–1528, 2020.

[10] Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

[11] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[12] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[13] Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007.

[14] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[16] Misha Denil, Babak Shakibi, Laurent Dinh, Marc' Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26, 2013.

[17] Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4510–4520, 2021.

[18] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017.

[19] Xin Dong, Junfeng Guo, Ang Li, Wei-Te Ting, Cong Liu, and H.T. Kung. Neural mean discrepancy for efficient out-of-distribution detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19217–19227, June 2022.

[20] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16091–16101, 2023.

[21] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR, 2018.

[22] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

[23] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *arXiv preprint arXiv:1903.01611*, 2019.

[24] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[25] Alireza Ganjdanesh*, Shangqian Gao*, and Heng Huang. Interpretations steered network pruning via amortized inferred saliency maps. In *European Conference on Computer Vision (ECCV)*, pages 278–296. Springer, 2022.

[26] Alireza Ganjdanesh, Shangqian Gao, and Heng Huang. Effconv: Efficient learning of kernel sizes for convolution layers of cnns. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

[27] Alireza Ganjdanesh*, Shangqian Gao*, and Heng Huang. Compressing image-to-image translation gans using local density structures on their learned manifold. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (to appear)*, 2024.

[28] Alireza Ganjdanesh*, Shangqian Gao*, and Heng Huang. Jointly training and pruning cnns via learnable agent guidance and alignment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (to appear)*, 2024.

[29] Shangqian Gao, Cheng Deng, and Heng Huang. Cross domain model compression by structurally weight sharing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8973–8982, 2019.

[30] Shangqian Gao, Feihu Huang, Weidong Cai, and Heng Huang. Network pruning via performance maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9270–9280, 2021.

[31] Shangqian Gao, Feihu Huang, Jian Pei, and Heng Huang. Discrete model compression with resource constraint for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[32] Shangqian Gao, Feihu Huang, Jian Pei, and Heng Huang. Discrete model compression with resource constraint for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1899–1908, 2020.

[33] Shangqian Gao, Feihu Huang, Yanfu Zhang, and Heng Huang. Disentangled differentiable network pruning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[34] Shangqian Gao*, Junyi Li*, Zeyu Zhang*, Yanfu Zhang, Weidong Cai, and Heng Huang. Device-wise federated network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (to appear)*, 2024.

[35] Shangqian Gao, Burak Uzkent, Yilin Shen, Heng Huang, and Hongxia Jin. Learning to jointly share and prune weights for grounding based vision and language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[36] Shangqian Gao, Zeyu Zhang, Yanfu Zhang, Feihu Huang, and Heng Huang. Structural alignment for network pruning through partial regularization. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[37] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng zhong Xu. Dynamic channel pruning: Feature boosting and suppression. In *International Conference on Learning Representations*, 2019.

[38] Bin Gu, Xiyuan Wei, Shangqian Gao, Ziran Xiong, Cheng Deng, and Heng Huang. Black-box reductions for zeroth-order gradient algorithms to achieve lower query complexity. *The Journal of Machine Learning Research (JMLR)*, 22(1):7685–7731, 2021.

[39] Jinyang Guo, Wanli Ouyang, and Dong Xu. Multi-dimensional pruning: A unified framework for model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1508–1517, 2020.

[40] Junfeng Guo, Yiming Li, Xun Chen, Hanqing Guo, Lichao Sun, and Cong Liu. SCALE-UP: An efficient black-box input-level backdoor detection via analyzing scaled prediction consistency. In *The Eleventh International Conference on Learning Representations*, 2023.

[41] Sangchul Hahn and Heeyoul Choi. Gradient acceleration in activation functions. 2018.

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

[43] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[44] Babak Hassibi and David G Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

[45] Hussein Hazimeh, Natalia Ponomareva, Petros Mol, Zhenyu Tan, and Rahul Mazumder. The tree ensemble layer: Differentiability meets conditional computation. In *International Conference on Machine Learning*, pages 4138–4148. PMLR, 2020.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[47] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2009–2018, 2020.

[48] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2234–2240, 2018.

[49] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

[50] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[51] 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*, pages 1389–1397, 2017.

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

[53] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[54] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[55] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[56] Ting Hua, Xiao Li, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Dynamic low-rank estimation for transformer-based language models. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023.

[57] Weizhe Hua, Yuan Zhou, Christopher M De Sa, Zhiru Zhang, and G. Edward Suh. Channel gating neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[58] Chen Huang, Shuangfei Zhai, Walter Talbott, Miguel Ángel Bautista, Shih-Yu Sun, Carlos Guestrin, and Joshua M. Susskind. Addressing the loss-metric mismatch with adaptive loss alignment. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2019.

[59] Feihu Huang, Songcan Chen, and Heng Huang. Faster stochastic alternating direction method of multipliers for nonconvex optimization. In *International Conference on Machine Learning*, pages 2839–2848, 2019.

[60] Feihu Huang and Shangqian Gao. Riemannian gradient methods for stochastic composition problems. *Neural Networks*, 153:224–234, 2022.

[61] Feihu Huang and Shangqian Gao. Gradient descent ascent for minimax problems on riemannian manifolds. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2023.

[62] Feihu Huang, Shangqian Gao, Songcan Chen, and Heng Huang. Zeroth-order stochastic alternating direction method of multipliers for nonconvex nonsmooth optimization. *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 2549–2555, 2019.

[63] Feihu Huang*, Shangqian Gao*, and Heng Huang. Bregman gradient policy optimization. In *International Conference on Learning Representations (ICLR)*, 2022.

[64] Feihu Huang, Shangqian Gao, Jian Pei, and Heng Huang. Nonconvex zeroth-order stochastic admm methods with lower function query complexity. *arXiv preprint arXiv:1907.13463*, 2019.

[65] Feihu Huang, Shangqian Gao, Jian Pei, and Heng Huang. Momentum-based policy gradient methods. In *International conference on machine learning (ICML)*, pages 4422–4433. PMLR, 2020.

[66] Feihu Huang, Shangqian Gao, Jian Pei, and Heng Huang. Accelerated zeroth-order and first-order momentum methods from mini to minimax optimization. *The Journal of Machine Learning Research (JMLR)*, 23(1):1616–1685, 2022.

[67] Feihu Huang, Shangqian Gao, Jian Pei, and Heng Huang. Nonconvex zeroth-order stochastic admm methods with lower function query complexity. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) (to appear)*, 2023.

[68] Feihu Huang, Junyi Li, Shangqian Gao, and Heng Huang. Enhanced bilevel optimization via bregman distance. *Advances in Neural Information Processing Systems*, 35:28928–28939, 2022.

[69] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

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

[71] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on Machine Learning - Volume 37*, ICML, pages 448–456. JMLR.org, 2015.

[72] Roxana Istrate, Florian Scheidegger, Giovanni Mariani, Dimitrios Nikolopoulos, Constantine Bekas, and Adelmo Cristiano Innocenza Malossi. Tapas: Train-less accuracy predictor for architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3927–3934, 2019.

[73] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[74] Yiding Jiang, Dilip Krishnan, Hossein Mobahi, and Samy Bengio. Predicting the generalization gap in deep networks with margin distributions. In *International Conference on Learning Representations*, 2019.

[75] Minsoo Kang and Bohyung Han. Operation-aware soft channel pruning using differentiable masks. *International Conference on Machine Learning*, 2020.

[76] Ehud D Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks*, 1(2):239–242, 1990.

[77] Jaedeok Kim, Chiyoun Park, Hyun-Joo Jung, and Yoonsuck Choe. Plug-in, trainable gate for streamlining arbitrary neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[78] Minchul Kim, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Token fusion: Bridging the gap between token pruning and token merging. In *Winter Conference on Computer Vision (WACV)*, 2024.

[79] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[80] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014. Citeseer, 2000.

[81] Yu Kong*, Shangqian Gao*, Bin Sun, and Yun Fu. Action prediction from videos via memorizing hard-to-predict samples. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, volume 32, 2018.

[82] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

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

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

[85] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *ICLR*, 2019.

[86] Chao Li*, Shangqian Gao*, Cheng Deng, Wei Liu, and Heng Huang. Adversarial attack on deep cross-modal hamming retrieval. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2218–2227, 2021.

[87] Chao Li, Shangqian Gao, Cheng Deng, De Xie, and Wei Liu. Cross-modal learning with adversarial samples. *Advances in neural information processing systems (NeurIPS)*, 32, 2019.

[88] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ICLR*, 2017.

[89] Yawei Li, Kamil Adamczewski, Wen Li, Shuhang Gu, Radu Timofte, and Luc Van Gool. Revisiting random channel pruning for neural network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 191–201, 2022.

[90] Yawei Li, Shuhang Gu, Christoph Mayer, Luc Van Gool, and Radu 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*, pages 8018–8027, 2020.

[91] Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. Dhp: Differentiable meta pruning via hypernetworks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII 16*, pages 608–624. Springer, 2020.

[92] Yuchao Li, Shaohui Lin, Jianzhuang Liu, Qixiang Ye, Mengdi Wang, Fei Chao, Fan Yang, Jincheng Ma, Qi Tian, and Rongrong Ji. Towards compact cnns via collaborative compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6438–6447, 2021.

[93] Yunqiang Li, Jan C van Gemert, Torsten Hoefler, Bert Moons, Evangelos Eleftheriou, and Bram-Ernst Verhoef. Differentiable transportation pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16957–16967, 2023.

[94] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020.

[95] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[96] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[97] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. Channel pruning via automatic structure search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 673 – 679, 2020.

[98] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2019.

[99] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[100] Liu Liu, Lei Deng, Xing Hu, Maohua Zhu, Guoqi Li, Yufei Ding, and Yuan Xie. Dynamic sparse graph for efficient deep learning. In *International Conference on Learning Representations*, 2019.

[101] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021.

[102] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3296–3305, 2019.

[103] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[104] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

[105] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

[106] Zhaosong Lu and Xiaorui Li. Sparse recovery via partial regularization: Models, theory, and algorithms. *Mathematics of Operations Research*, 43(4):1290–1316, 2018.

[107] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*, 2018.

[108] Jian-Hao Luo, Hao Zhang, Hong-Yu Zhou, Chen-Wei Xie, Jianxin Wu, and Weiyao Lin. Thinet: pruning cnn filters for a thinner net. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[109] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[110] Zelda Mariet and Suvrit Sra. Diversity networks: neural network compression using determinantal point processes. *arXiv preprint arXiv:1511.05077*, 2015.

[111] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2498–2507. JMLR. org, 2017.

[112] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[113] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 4932–4942. Curran Associates, Inc., 2019.

[114] Ari S Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *NeurIPS*, 2019.

[115] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pages 6775–6784, 2017.

[116] Xuefei Ning, Tianchen Zhao, Wenshuo Li, Peng Lei, Yu Wang, and Huazhong Yang. Dsa: More efficient budgeted pruning via differentiable sparsity allocation. *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[117] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[118] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[119] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. Collaborative channel pruning for deep networks. In *International Conference on Machine Learning*, pages 5113–5122, 2019.

[120] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 11893–11902, 2020.

[121] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[122] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[123] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[124] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2020.

[125] Megvii Research. Shufflenetv2+. `https://github.com/megvii-model/ShuffleNet-Series/tree/master/ShuffleNetV2%2B`.

[126] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[127] Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. Hydra: Pruning adversarially robust neural networks. In *NeurIPS*, 2020.

[128] Zebang Shen, Alejandro Ribeiro, Hamed Hassani, Hui Qian, and Chao Mi. Hessian aided policy gradient. In *International Conference on Machine Learning*, pages 5729–5738. PMLR, 2019.

[129] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[130] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[131] Yang Sui, Miao Yin, Yi Xie, Huy Phan, Saman Aliari Zonouz, and Bo Yuan. Chip: Channel independence-based pruning for compact neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[132] Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chunjing Xu, Chao Xu, and Chang Xu. Scop: Scientific control for reliable neural network pruning. *Advances in Neural Information Processing Systems*, 33, 2020.

[133] Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020.

[134] Chaoqi Wang, Roger B. Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, pages 6566–6575, 2019.

[135] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. In *International Conference on Learning Representations*, 2021.

[136] Wenxiao Wang, Minghao Chen, Shuai Zhao, Long Chen, Jinming Hu, Haifeng Liu, Deng Cai, Xiaofei He, and Wei Liu. Accelerate cnns from three dimensions: A comprehensive pruning framework. In *International Conference on Machine Learning*, pages 10717–10726. PMLR, 2021.

[137] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[138] Zi Wang, Chengcheng Li, and Xiangyang Wang. Convolutional neural network pruning with structural redundancy reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14913–14922, 2021.

[139] Andreas S Weigend, David E Rumelhart, and Bernardo A Huberman. Generalization by weight-elimination with application to forecasting. In *Advances in neural information processing systems*, pages 875–882, 1991.

[140] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

[141] Xidong Wu*, Shangqian Gao*, Zeyu Zhang, Zhenzhen Li, Runxue Bao, Yanfu Zhang, Xiaoqian Wang, and Heng Huang. Auto-train-once: Controller network guided automatic network pruning from scratch. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (to appear)*, 2024.

[142] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. *NeurIPS*, 2019.

[143] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*, 2018.

[144] Mao Ye, Chengyue Gong, Lizhen Nie, Denny Zhou, Adam Klivans, and Qiang Liu. Good subnetworks provably exist: Pruning via greedy forward selection. *International Conference on Machine Learning*, 2020.

[145] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2130–2141, 2019.

[146] Sixing Yu, Arya Mazaheri, and Ali Jannesari. Auto graph encoder-decoder for neural network pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6362–6372, 2021.

[147] Sixing Yu, Arya Mazaheri, and Ali Jannesari. Topology-aware network pruning using multi-stage graph embedding and reinforcement learning. In *International Conference on Machine Learning*, pages 25656–25667. PMLR, 2022.

[148] Dejiao Zhang, Haozhu Wang, Mario Figueiredo, and Laura Balzano. Learning to share: Simultaneous parameter tying and sparsification in deep learning. 2018.

[149] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 184–199, 2018.

[150] Yanfu Zhang*, Shangqian Gao*, and Heng Huang. Exploration and estimation for model compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 487–496, 2021.

[151] Yanfu Zhang, Shangqian Gao, and Heng Huang. Recover fair deep classification models via altering pre-trained structure. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[152] Yanfu Zhang, Shangqian Gao, Jian Pei, and Heng Huang. Improving social network embedding via new second-order continuous graph neural networks. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining (KDD)*, pages 2515–2523, 2022.

[153] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, 2019.

[154] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 875–886, 2018.

[155] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. 2017.