Network Pruning via Performance Maximization

Shangqian Gao[1], Feihu Huang[1], Weidong Cai[2], and Heng Huang*[1,3]

[1]Electrical and Computer Engineering Department, University of Pittsburgh, PA, USA
[2]School of Computer Science, The University of Sydney, NSW 2006, Australia
[3]JD Finance America Corporation, Mountain View, CA, USA
shg84@pitt.edu, huangfeihu2018@gmail.com, tom.cai@sydney.edu.au, heng.huang@pitt.edu

Abstract

Channel pruning is a class of powerful methods for model compression. When pruning a neural network, it's ideal to obtain a sub-network with higher accuracy. However, a sub-network does not necessarily have high accuracy with low classification loss (loss-metric mismatch). In the paper, we first consider the loss-metric mismatch problem for pruning and propose a novel channel pruning method for Convolutional Neural Networks (CNNs) by directly maximizing the performance (i.e., accuracy) of sub-networks. Specifically, we train a stand-alone neural network to predict sub-networks' performance and then maximize the output of the network as a proxy of accuracy to guide pruning. Training such a performance prediction network efficiently is not an easy task, and it may potentially suffer from the problem of catastrophic forgetting and the imbalance distribution of sub-networks. To deal with this challenge, we introduce a corresponding episodic memory to update and collect sub-networks during the pruning process. In the experiment section, we further demonstrate that the gradients from the performance prediction network and the classification loss have different directions. Extensive experimental results show that the proposed method can achieve state-of-the-art performance with ResNet, MobileNetV2, and ShuffleNetV2+ on ImageNet and CIFAR-10.

1. Introduction

Convolutional Neural Networks (CNNs) have demonstrated great successes in many computer vision and machine learning applications, like classification [32], detection [47, 48], action recognition [51] and self-driving cars [2]. To achieve better performances on these tasks, the design of CNNs becomes more and more complex in terms of depth and width [52, 13, 21] since AlexNet [32]. 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 [12], quantization [3], structural pruning [34] 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 [16, 37]. To find such a sub-network, many existing channel pruning approaches [28, 57, 58] 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 [20]. 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. 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 triv-

*This work was partially supported by NSF IIS 1845666, 1852606, 1838627, 1837956, 1956002, 2040588.

ial 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 [10], 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.

2. Related Works

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 [54] or remove weights based on the sensitivity [33, 27]. A more recent weight pruning work [12] prunes weights according to their L_2 or L_1 norms. Another work by *Dong et al.* [7] explored weight pruning based on second derivative information. Other weight pruning methods include data-driven pruning [19], variational dropout [41], utilizing determinantal point process [40] and so on. Different from the above works, the lottery ticket hypothesis [8] argues that there exist good sub-networks within a randomly initialized large network. Follow up works [9, 43] 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 [34] removes filters based on their L_1 norms. Soft filter pruning [15] 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 [22] adds learnable scaling factors to prune neurons or layers with the sparsity regularization. Discrimination-aware pruning [58] considers both classification loss and norms to guide the pruning of the model. Gate decorator [57] inserts learnable factors for each channel and then uses Taylor expansion of the loss to estimate the global importance. Operation-aware pruning [26] 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) [16] adapts reinforcement learning for structural pruning. With reinforcement learning, model accuracy can be directly used in reward function to guide the pruning process. Metapruning [37] utilizes a hypernet to predict the weights of sub-networks, and it then applies the evolutionary search for pruning. With an evolutionary search, accuracy is directly used as guidance.

Differentiable pruning approaches [26, 57, 28, 11] 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.

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 [53] 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 [25], they use margin distributions of multiple layers to predict the generalization gaps of neural networks. A more related work [23] trains an accuracy predictor for neural architecture search (NAS)

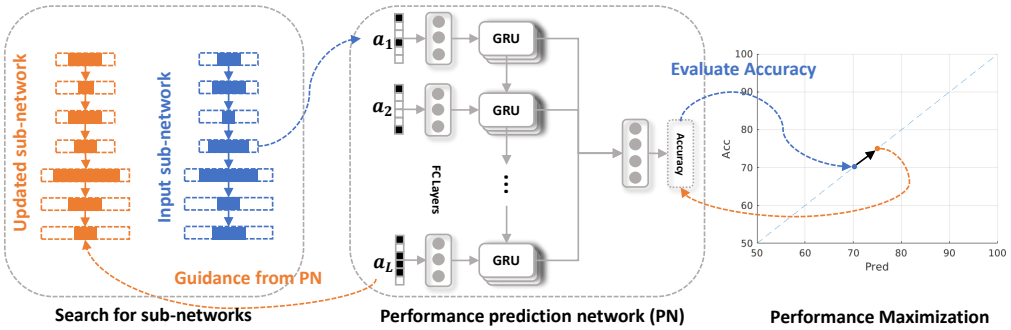


Figure 1: The flowchart of performance maximization process. A sub-network is first sampled from differentiable gates. The performance prediction network is then used as a proxy to maximize the accuracy of sub-networks.

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. Network Pruning via Performance Maximization

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.

3.2. 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:

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

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:

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

where $a_i \in \{0, 1\}^{C_i}$. Since the indicator function $\mathbb{1}(\cdot)$ is not differentiable, we use straight-through estimator [1] to calculate gradients. The differentiable gate in Eq. 1 and Eq. 2 uses Gumbel-Solftmax [24] 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 $\mathcal{F}_i$:

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

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

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

where $\mathbf{w}$ contains all learnable weights of gates defined in Eq. (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. 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. 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 [5]; 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:

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

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.

3.4. 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 [10]. 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:

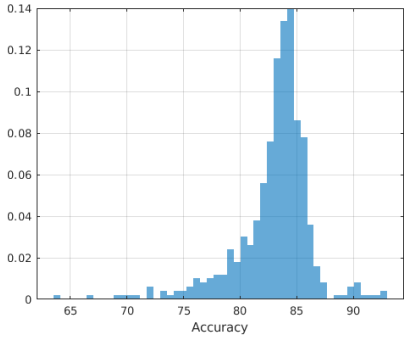


Figure 2: Empirical distribution of the accuracy from sub-networks collected during the pruning process. The results are based on ResNet-56 on CIFAR-10.

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

Within certain iterations, sub-networks produced by Eq. 1 and Eq. 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:

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

$\bar{\mathbf{a}}$ is the sub-network defined in Eq. 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.

3.5. Imbalanced Accuracy Distribution

In Fig. 2, 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

Algorithm 1: 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. 1 and Eq. 2 and its accuracy q
    2. update the sub-network according to Eq. 6.
    3. update **EM** with Eq. 7 and K every c iterations.
    if $K > b_P$ **then**
      4. update $\mathbf{w}_p$ with Eq. 5 with a mini-batch sampled from **EM**.
    end
    5. calculate gradients for $\mathbf{w}$ by backpropagation through Eq. 8.
    6. modify gradients according to Eq. 9.
    7. update $\mathbf{w}$ with ADAM.
  end
end
return $\mathbf{w}$.

of their count. This is equivalent to create N pseudo-classes and conduct re-sampling.

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 $\mathrm{PN}(\mathbf{a})$, thus we can maximize $\mathrm{PN}(\mathbf{a})$ as a proxy of accuracy. $\max_{\mathbf{w}} \mathrm{PN}(\mathbf{a})$ is equivalent to $\min_{\mathbf{w}} \frac{1}{\mathrm{PN}(\mathbf{a})}$. To stabilize the training, we optimize the follwing problem instead: $\min_{\mathbf{w}} \log(\frac{1}{\mathrm{PN}(\mathbf{a})})$. The overall optimization problem is shown in the following equation:

$$\min_{\mathbf{w}} \mathcal{J}(\mathbf{w}) = \mathcal{L}(f(x; \mathbf{a}, \Theta), y) + \gamma_{(K,\mathcal{L}_{\mathrm{P}})} \cdot \log(\frac{1}{\mathrm{PN}(\mathbf{a})}) + \lambda \mathcal{R}(T(\mathbf{a}), pT_{\mathrm{total}}), \tag{8}$$

where $\gamma_{(K,\mathcal{L}_{\mathrm{P}})}$ is a function to reflect the confidence of the performance prediction network and it is used to automatically control the magnitude of $\log(\frac{1}{\mathrm{PN}(\mathbf{a})})$, λ is used to control the magnitude of the regularization, and the other terms are introduced in Eq. 4. $\gamma_{(K,\mathcal{L}_{\mathrm{P}})}$ is defined as: $\gamma_{(K,\mathcal{L}_{\mathrm{P}})} = \mathbb{1}_{K \geq \frac{K_{\max}}{4}}(K) \cdot (1 - \mathcal{L}_p)^2$, and the range of $\gamma_{(K,\mathcal{L}_{\mathrm{P}})}$ is $[0, 1]$. Usually, lower $\mathcal{L}_{\mathrm{P}}$ indicates higher confidence of $\mathrm{PN}(\cdot)$. However, the training of PN is an incremental learning task, $\mathcal{L}_{\mathrm{P}}$ 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 the gradients orthogonal to each other. Let $g^i_{\mathcal{L}} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}_i}$ represents the gradient from the classification loss of ith layer, and $g^i_{\mathrm{P}} = \frac{\partial \log(\frac{1}{\mathrm{PN}(\mathbf{a})})}{\partial \mathbf{w}_i}$ be the gradient from performance maximization. The modified gradients from these two terms are:

$$g^i = g^i_{\mathcal{L}} + \hat{g}^i_{\mathrm{P}}, \tag{9}$$

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

The overall algorithm is shown in Alg. 1. The process of performance maximization is shown in Fig. 1. The explanation of inputs is listed here: D: dataset, p: pruning rate of FLOPs, λ is introduced in Eq. 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. 1, 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. 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.

The proposed PN shares certain properties of the value function [30]. 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.

4. Experiments

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 [31] and ImageNet [6] 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 [13], MobileNetV2 [50] and ShuffleNetV2+ [39, 49] are used for evaluation. ShuffleNetV2+ is an improved version of ShuffleNetV2 which has similar performance with MobileNetV3 [18]. These models are generally harder to prune

Method	Architecture	Baseline Acc	Pruned Acc	Δ-Acc	↓ FLOPs
AMC [16]	ResNet-56	92.80%	91.90%	-0.90%	50.0%
SFP [15]		93.59%	93.35%	-0.24%	52.6%
DCP [58]		93.80%	93.81%	+0.01%	47.0%
CCP [46]		93.50%	93.42%	-0.08%	**52.6%**
HRank [36]		93.26%	92.17%	-0.09%	50.0%
Pruning Criterion [14]		93.59%	93.24%	-0.35%	52.6%
NPPM(ours)		93.04%	93.40%	**+0.36%**	50.0%
WM [58]	MobileNetV2	94.47%	94.17%	-0.30%	26.0%
DCP [58]		94.47%	94.69%	+0.22%	26.0%
NPPM(ours)		94.23%	94.75%	**+0.52%**	**47.0%**

Table 1: 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.

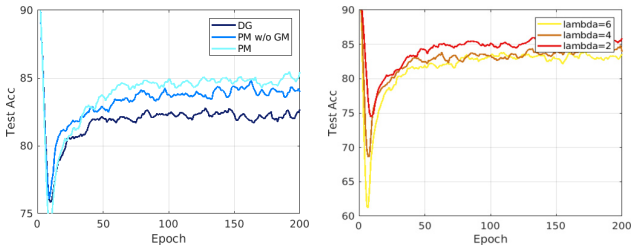
compared with AlexNet or VGG. We use p to decide how much FLOPs should be removed, the detailed choices of p are listed in supplementary materials. We choose $\lambda = 2$ used in Eq. 8 for all experiments.

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 [50]. 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 [39, 49] too.

When training $\mathbf{w}$ and $\mathbf{w}_P$, we use ADAM [29] 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 [45]. The experiments are conducted on a machine with 4 Nvidia Tesla P40 GPUs.

4.2. CIFAR-10 Results

In Tab. 1, 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



(a) (b)

Figure 3: (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

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.

4.3. ImageNet Results

We present all results for ImageNet in Tab. 2. 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

Method	Architecture	Pruned Top-1	Pruned Top-5	Δ Top-1	Δ Top-5	↓ FLOPs	FLOPs
Pruning Filters [34]	ResNet-34	72.17%	-	-1.06%	-	24.8%	2.77G
SFP [15]		71.84%	89.70%	-2.09%	-1.92%	41.1%	2.17G
IE [42]		72.83%	-	-0.48%	-	24.2%	2.79G
FPGM [17]		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 [58]	ResNet-50	74.95%	92.32%	-1.06%	-0.61%	55.6%	1.83G
CCP [46]		75.21%	92.42%	-0.94%	-0.45%	54.1%	1.89G
MetaPruning [37]		75.40%	-	-1.20%	-	51.2%	2.01G
GBN [57]		75.18%	92.41%	-0.67%	-0.26%	55.1%	1.85G
HRank [36]		74.98%	92.33%	-1.17%	-0.54%	43.8%	2.32G
Hinge [35]		74.70%	-	-1.40%	-	54.4%	1,88G
DSA [44]		74.69%	92.45%	-1.33%	-0.80%	50.0%	2.06G
SCP [26]		75.27%	92.30%	-0.62%	-0.68%	54.3%	1.88G
LeGR [4]		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 [55]	ResNet-101	77.37%	-	-2.10%	-	47.0%	4.16G
IE [42]		77.35%	-	-0.02%	-	39.8%	4.72G
FPGM [17]		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 [50]	MobileNetV2	69.80%	89.60%	-2.00%	-1.40%	30.0%	220M
AMC [16]		70.80%	-	-1.00%	-	30.0%	220M
MetaPruning [37]		71.20%	-	-0.80%	-	**30.7%**	217M
LeGR [4]		71.40%	-	-0.40%	-	30.0%	220M
Greedy Pruning [56]		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

Table 2: 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.

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%

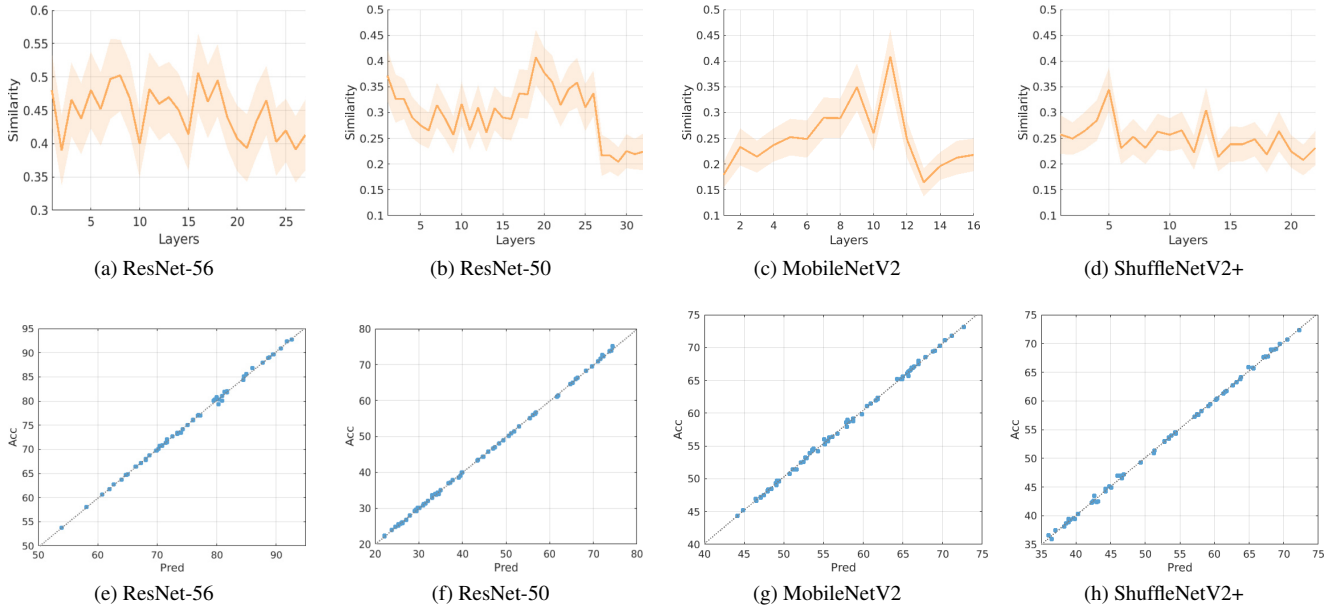
Table 3: Difference between finetuning and training from scratch for our method.

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 [57, 26, 58] 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



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

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

Figure 4: (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.

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. 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.

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. 4. To plot predictions from PN, we sample 100 sub-networks in **EM** and calculate the accuracy on the subset. From Fig. 4 (e)∼(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. 4 (a)∼(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 3 (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 [38]. Our results (Tab. 3) 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.

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.

References

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 3

[2] 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. 1

[3] 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. 1

[4] 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. 7

[5] 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. 4

[6] 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. 5

[7] 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. 2

[8] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. 2

[9] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *arXiv preprint arXiv:1903.01611*, 2019. 2

[10] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999. 2, 4

[11] 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*, pages 1899–1908, 2020. 2

[12] 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. 1, 2

[13] 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. 1, 5

[14] 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. 6

[15] 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. 2, 6, 7

[16] 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. 1, 2, 6, 7

[17] 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. 7

[18] 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. 5

[19] 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. 2

[20] 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. 1

[21] 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. 1

[22] 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. 2

[23] 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. 2

[24] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 3

[25] 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. 2

[26] Minsoo Kang and Bohyung Han. Operation-aware soft channel pruning using differentiable masks. *International Conference on Machine Learning*, 2020. 2, 7

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

[28] 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. 1, 2

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[30] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014. Citeseer, 2000. 5

[31] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 5

[32] 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. 1

[33] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990. 2

[34] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ICLR*, 2017. 1, 2, 7

[35] 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. 7

[36] 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. 6, 7

[37] 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. 1, 2, 7

[38] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019. 8

[39] 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. 5, 6

[40] Zelda Mariet and Suvrit Sra. Diversity networks: neural network compression using determinantal point processes. *arXiv preprint arXiv:1511.05077*, 2015. 2

[41] 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. 2

[42] 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. 7

[43] 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. 2

[44] Xuefei Ning, Tianchen Zhao, Wenshuo Li, Peng Lei, Yu Wang, and Huazhong Yang. Dsa: More efficient budgeted pruning via differentiable sparsity allocation. *European Conference on Computer Vision*, 2020. 7

[45] 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. 6

[46] 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. 6, 7

[47] 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. 1

[48] 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. 1

[49] Megvii Research. Shufflenetv2+. https://github.com/megvii-model/ShuffleNet-Series/tree/master/ShuffleNetV2%2B. 5, 6

[50] 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. 5, 6, 7

[51] 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. 1

[52] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[53] Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020. 2

[54] 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. 2

[55] 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. 7

[56] 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. 7

[57] 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. 1, 2, 7

[58] 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. 1, 2, 6, 7