# Disentangled Differentiable Network Pruning

Shangqian Gao, Feihu Huang, Yanfu Zhang, and Heng Huang

Department of Electrical and Computer Engineering, University of Pittsburgh

{shg84,feh23,yaz91,heng.huang}@pitt.edu

**Abstract.** In this paper, we propose a novel channel pruning method for compression and acceleration of Convolutional Neural Networks (CNNs). Many existing channel pruning works try to discover compact sub-networks by optimizing a regularized loss function through differentiable operations. Usually, a learnable parameter is used to characterize each channel, which entangles the width and channel importance. In this setting, the FLOPs or parameter constraints implicitly restrict the search space of the pruned model. To solve the aforementioned problems, we propose optimizing each layer's width by relaxing the hard equality constraint used in previous works. The relaxation is inspired by the definition of the top-$k$ operation. By doing so, we partially disentangle the learning of width and channel importance, which enables independent parametrization for width and importance and makes pruning more flexible. We also introduce soft top-$k$ to improve the learning of width. Moreover, to make pruning more efficient, we use two neural networks to parameterize the importance and width. The importance generation network considers both inter-channel and inter-layer relationships. The width generation network has similar functions. In addition, our method can be easily optimized by popular SGD methods, which enjoys the benefits of differentiable pruning. Extensive experiments on CIFAR-10 and ImageNet show that our method is competitive with state-of-the-art methods.

**Keywords:** Model Compression, Channel Pruning, Differentiable Pruning

## 1 Introduction

Convolutional Neural Networks (CNNs) have accomplished tremendous success in various computer vision tasks [2, 28, 43, 44, 47]. To deal with real-world applications, recently, the design of CNNs has become more and more complicated in terms of width, depth, etc. [14,20,28,48]. 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 [11, 12] 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 [9,24,25,54] 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 [17], evolutionary algorithms [33] 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 [9, 24, 25, 54] 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 [9, 24, 25, 54], our method disentangles the learning of width and channel importance, which potentially enlarge the search space. Compared to search based algorithms [17, 33], 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.

## 2 Related Works

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

### 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, [12] proposes to use $L_1$ or $L_2$ magnitude as the criterion to prune weights and connections. SNIP [29] updates the importance of each weight by using gradients from the loss function. Weights with lower importance will be pruned. Lottery ticket hypothesis [7] 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 [35], 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.

### 2.2 Structural Pruning

One of the previous works [30] 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 [52] 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 [56]. 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, [36] 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

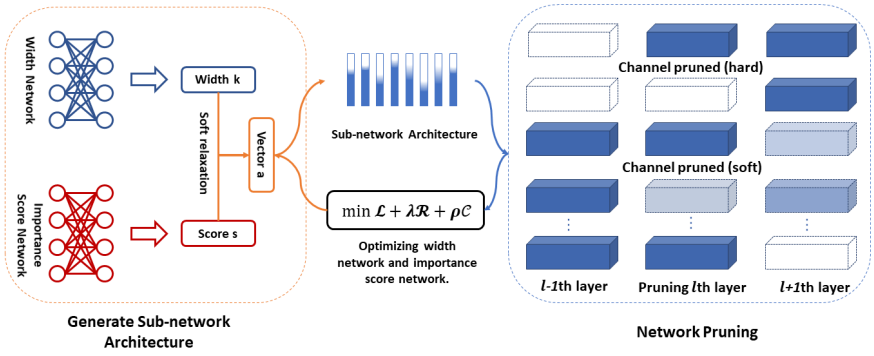



**Fig. 1.** Flowchart of our proposed method. Importance score generator $g_s$ and width generator $g_k$ are used to generate the importance score and width. We then use them to generate the mask vector $\mathbf{a}$, and it is used to produce the sub-network architecture. The network is pruned according to $\mathbf{a}$. Finally, $g_k$ and $g_s$ are optimized by minimizing the loss function.

structure pruning, since batchnorm [22] is widely used in recent neural network designs [14, 20]. A straightforward way to achieve channel pruning is to make the scaling factor of batchnorm to be sparse [34]. If the scaling factor falls below a certain threshold, the channel will be removed. Structure sparse selection [21] 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 [8, 9, 24, 25, 54, 57], 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 [41] tries to prune channels by using Taylor expansion. Greedy forward selection [53] 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) [17], they use policy gradient to update the policy network. This policy network is then used to generate the width of each layer. MetaPruning [33] 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.

### 2.3 Other Methods

Besides weight and channel pruning methods, there are works from other perspectives, including bayesian pruning [37,39], weight quantization [5,42], pruning for fairness [58], and knowledge distillation [18].

# 3 Proposed Method

## 3.1 Preliminary

To better describe our proposed approach, necessary notations are introduced first. In a CNN, the feature map of $l$th 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. 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:

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

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. $\Theta$ are learnable parameters to decide whether to prune the channel. There are many ways to characterize a channel, such as Gumbel-sigmoid approximation [23], shape function [25] 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, $\Theta$ will be reparameterized into two variables: importance score $\mathbf{s}$ and layer width $\mathbf{k}$. How to achieve such a disentanglement will be detailed in this section.

## 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:

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

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

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

Gradients with respect to $k$ through Eq. 3 are not defined. Except Eq. 3, we can use an alternative surrogate to represent $k$:

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

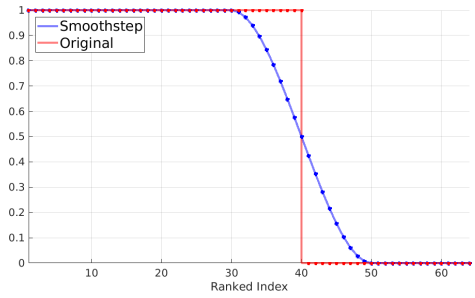



**Fig. 2.** Soft relaxation vs. naive binary mask. In this figure, we choose $C_l = 64$, $\gamma = 20$, $C_l k = 40$. Solid line represent the continuous function. Square dots represent the actual values taken by the vector $\tilde{\mathbf{a}}$.

where $s_0$ is a value between $k$th and $k+1$th value, and the indicator function $\mathbb{1}_{s_i > s_0}(\cdot)$ 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. 4, it still entangles the learning of importance and width. We then replace it with a regularization term:

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

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

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

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 $\mathcal{F}_l$.

### 3.3 Soft Top-$k$ Operation with Smoothstep Function

When performing discrete top-$k$ operation, we place 1 to the first $k$ elements of $\tilde{\mathbf{a}}$. Similarly, we use a smoothstep function [13] to generate values for soft relaxed $\tilde{\mathbf{a}}$:

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

In smoothstep function, $\gamma$ 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. 2. 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. 3, 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 choose the smoothstep function since it provides a 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 any more on $k$. We adjust $\gamma$ to round$(C_l - C_l k)$ to ensure $C_l k \approx \sum a_i$.

Binary values are often used to control open or close 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 for 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 $\tilde{\mathbf{a}} = P\mathbf{a}$.

### 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:

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

where $\mathcal{S} = (\mathbf{s}_1, \cdots, \mathbf{s}_L)$ is the collection of all scores across different layers, $\Theta_s$ 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 [4] and fully connected layers. Basically, we use GRU to capture inter-layer relationships, and fully connected layers are for inter-channel relationships. The additional computational costs introduced by $g_s$ is trivial, and it has little impact to the training time. Since $\mathcal{S}$ is not directly involved in the forward computation, we use straight-through gradient estimator [1] 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 [46] or other functions incurs much larger errors when estimating the gradients. This is probably because simgoid$(\cdot)$ better approximates binary values.

**Algorithm 1:** Disentangled Differentiable Network Pruning

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

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

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

where $x_k$ is the input to $g_k$, $\Theta_k$ 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 with fully connected layers. In addition, like $x_s$, $x_k$ is also generated randomly, and it is kept fix when training $g_k$.

### 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:

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

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, $T_{\text{total}}$ is the total prunable FLOPs, $\mathcal{J}$ is the overall objective function, $\mathcal{C}(\mathbf{k}, \mathcal{S})$ is defined in Eq. 5, 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. 10 can be optimized efficiently by using any stochastic gradient optimizer. Using learnable importance score

produces quite strong empirical performance. If better learning mechanism for importance score is designed, it can also be merged into our algorithm.

The overall algorithm is given in Algorithm 1. The input of Algorithm 1 are $D$: a dataset for pruning, $p$: the pruning rate defined in Eq. 10, $\lambda$ and $\rho$: 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 1, the gradients of $\Theta_k$ and $\Theta_s$ 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 [17], 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. 2 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. 1.

## 4 Connections to Previous Works

In this section, we will discuss the difference 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. 10:

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

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. 11 is closely related to Eq. 1. If we insert $\frac{1}{C_l} \sum_{i=1}^{C_l} \mathbb{1}_{s_i^l > 0.5}(s_i^l)$ to every $k_l$ in $T(\mathbf{k})$, we almost recover Eq. 1. Compared to Eq. 10, Eq. 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. 11 corresponds to set $\rho$ to $\infty$ in Eq. 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. 10 actually interpolates between previous differentiable pruning works and the complete disentangled formulation.

| Method | Architecture | Baseline Acc | Pruned Acc | $\Delta$-Acc | $\downarrow$ FLOPs |
|---|---|---|---|---|---|
| AMC [17] | ResNet-56 | 92.80% | 91.90% | −0.90% | 50.0% |
| DCP [59] | | 93.80% | 93.81% | +0.01% | 47.0% |
| CCP [41] | | 93.50% | 93.42% | −0.08% | 52.6% |
| HRank [32] | | 93.26% | 93.17% | −0.09% | 50.0% |
| LeGR [3] | | 93.90% | 93.70% | −0.20% | **53.0%** |
| DDNP (ours) | | 93.62% | 93.83% | **+0.21%** | 51.0% |
| Uniform [59] | MobileNetV2 | 94.47% | 94.17% | −0.30% | 26.0% |
| DCP [59] | | 94.47% | 94.69% | +0.22% | 26.0% |
| MDP [10] | | 95.02% | 95.14% | +0.12% | 28.7% |
| SCOP [49] | | 94.48% | 94.24% | −0.24% | 40.3% |
| DDNP (ours) | | 94.58% | 94.81% | **+0.23%** | **43.0%** |

**Table 1.** Comparison results on CIFAR-10 dataset with ResNet-56 and MobileNetV2. $\Delta$-Acc represents the performance changes before and after model pruning. +/- indicates increase or decrease compared to baseline results.

## 5 Experiments

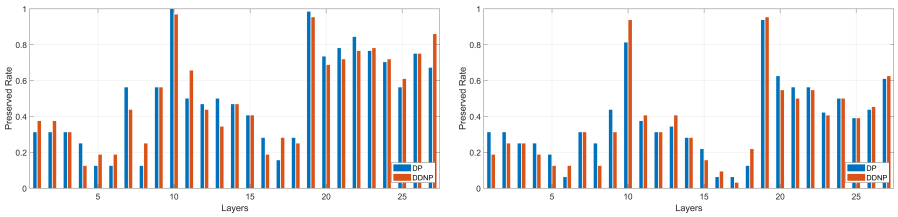
### 5.1 Settings

Similar to many model compression works, CIFAR-10 [27] and ImageNet [6] 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 also provided in supplementary materials. $\gamma$ in Eq. 7 is chosen as round$(0.1C_l)$. $\gamma$ then depends on layer width $C_l$, and it empirically works well.

Within the experiment section, our method is called as 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 [14], MobileNetV2 [45] and MobileNetV3 [19] are much harder to prune than earlier models like AlexNet [28] and VGG [48].

$\lambda$ 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 [40]. 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 [45]. 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 [19].

For training $g_k$ and $g_s$, we use ADAM [26] optimizer with a constant learning rate of 0.001 and train them for 200 epochs. We start pruning from the whole




(a) Layer-wise width ($p$=0.5) (b) Layer-wise width ($p$=0.35)

**Fig. 3.** (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.

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 1. 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 [40].

### 5.2 CIFAR-10

We present comparison results on CIFAR-10 in Tab. 1. On ResNet-56, our method achieves the largest performance gain (+0.21% $\Delta$-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 $\Delta$-Acc. Although DCP has the second best $\Delta$-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 $\Delta$-Acc. LeGR prunes more FLOPs than our method, but it has a much lower $\Delta$-Acc ($-0.20\%$ vs. $+0.21\%$).

For MobileNetV2, our method achieves the best $\Delta$-Acc and prunes most FLOPs (+0.23% $\Delta$-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 $\Delta$-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.

### 5.3 ImageNet Results

The ImageNet results are given in Tab. 2. We present both base and pruned Top-1/Top-5 accuracy in the table.

**ResNet-34.** Our method achieves the best $\Delta$ Top-1 and $\Delta$ Top-5 accuracy with ResNet-34. IE performs the second best regarding $\Delta$ Top-1/$\Delta$ 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

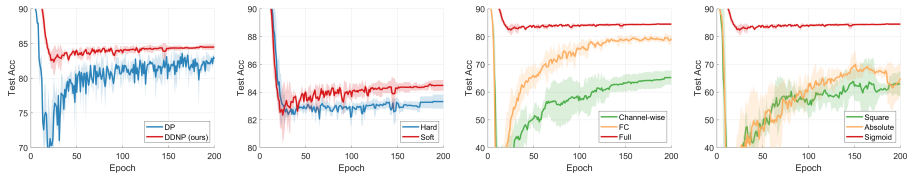
| Method | Architecture | Base/Pruned Top-1 | Base/Pruned Top-5 | $\Delta$ Top-1 | $\Delta$ Top-5 | $\downarrow$ FLOPs |
|---|---|---|---|---|---|---|
| SFP [15] | ResNet-34 | 73.93%/71.84% | 91.62%/89.70% | −2.09% | −1.92% | 41.1% |
| IE [38] | | 73.31%/72.83% | -/- | −0.48% | - | 24.2% |
| FPGM [16] | | 73.92%/72.63% | 91.62%/91.08% | −1.29% | −0.54% | 41.1% |
| SCOP [49] | | 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 [59] | ResNet-50 | 76.01%/74.95% | 92.93%/92.32% | −1.06% | −0.61% | 55.6% |
| CCP [41] | | 76.15%/75.21% | 92.87%/92.42% | −0.94% | −0.45% | 54.1% |
| MetaPruning [33] | | 76.60%/75.40% | -/- | −1.20% | - | 51.2% |
| GBN [54] | | 75.85%/75.18% | 92.67%/92.41% | −0.67% | −0.26% | 55.1% |
| HRank [32] | | 76.15%/74.98% | 92.87%/92.33% | −1.17% | −0.54% | 43.8% |
| LeGR [3] | | 76.10%/75.30% | -/- | −0.80% | - | 54.0% |
| SCOP [49] | | 76.15%/75.26% | 92.87%/92.53% | −0.89% | −0.34% | 54.6% |
| GReg [50] | | 76.13%/75.36% | -/- | −0.77% | - | **56.7%** |
| SRR [51] | | 76.13%/75.11% | 92.86%/92.35% | −1.02% | −0.51% | 55.1% |
| CC [31] | | 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 [45] | MobileNetV2 | 71.80%/69.80% | 91.00%/89.60% | −2.00% | −1.40% | 30.0% |
| AMC [17] | | 71.80%/70.80% | -/- | −1.00% | - | 30.0% |
| AGMC [55] | | 71.80%/70.87% | -/- | −0.93% | - | 30.0% |
| MetaPruning [33] | | 72.00%/71.20% | -/- | −0.80% | - | **30.7%** |
| GFS [53] | | 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 [19] | MobileNetV3 small | 67.50%/65.40% | -/- | −2.10% | - | **26.6%** |
| GFS [53] | | 67.50%/65.80% | -/- | −1.70% | - | 23.5% |
| DDNP (ours) | | 67.67%/**67.03%** | 87.40%/**86.94%** | **−0.64%** | **−0.46%** | 24.5% |

**Table 2.** Comparison results on ImageNet dataset with ResNet-34, ResNet-50, ResNet-101 and MobileNetV2. $\Delta$-Acc represents the performance changes before and after model pruning. +/- indicates increase or decrease compared to baseline results.

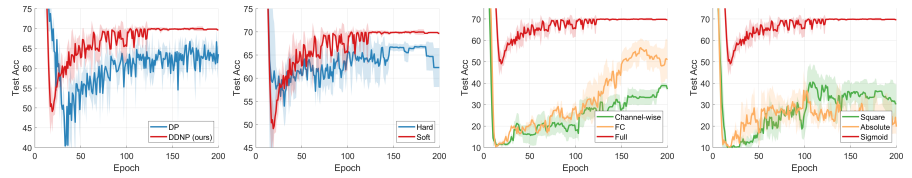
rate, but the FLOPs gap between our method and SCOP is quite marginal (only 0.6%). Given similar pruning rates, our method outperforms other baselines by at least 0.41% in terms of $\Delta$ Top-1 accuracy.

**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 $\Delta$ 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 $\Delta$ 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 [53], the Top-1/$\Delta$ 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.




(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 ($p = 0.35$)

**Fig. 4.** (a,e): Comparisons of our method and the diferentiable pruning (DP) baseline. (b,f): Comparisons of soft and hard setting 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.

**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/$\Delta$ 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.

### 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. 4 (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. 3. An interesting observation

is that, with different $p$, the relative order of width changes (like the first and second block) with our method.

In Fig. 4 (b,f), we verify the effectiveness of soft top-$k$ defined in section 3.3. The hard setting refers to set $\gamma = 0$ in Eq. 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. 4 (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. 4. 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, importance score with sigmoid function, absolute function and square function are defined as: $s = \text{sigmoid}(\bar{s})$, $s^{AF} = \mid \bar{s} \mid$ and $s^{SF} = \bar{s}^2$. From the Fig. 4, 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.

## 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.

## Acknowledgement

This work was partially supported by NSF IIS 1845666, 1852606, 1838627, 1837956, 1956002, 2217003.

## References


1. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
2. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
3. Chin, T.W., Ding, R., Zhang, C., Marculescu, D.: Towards efficient model compression via learned global ranking. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1518–1528 (2020)
4. Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. In: Conference on Empirical Methods in Natural Language Processing (EMNLP 2014) (2014)
5. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: Advances in neural information processing systems. pp. 3123–3131 (2015)
6. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. pp. 248–255. Ieee (2009)
7. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=rJl-b3RcF7
8. Ganjdanesh, A., Gao, S., Huang, H.: Interpretations steered network pruning via amortized inferred saliency maps. In: Proceedings of the European Conference on Computer Vision (ECCV) (2022)
9. Gao, S., Huang, F., Pei, J., Huang, H.: Discrete model compression with resource constraint for deep neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1899–1908 (2020)
10. Guo, J., Ouyang, W., Xu, D.: Multi-dimensional pruning: A unified framework for model compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1508–1517 (2020)
11. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
12. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems. pp. 1135–1143 (2015)
13. Hazimeh, H., Ponomareva, N., Mol, P., Tan, Z., Mazumder, R.: The tree ensemble layer: Differentiability meets conditional computation. In: International Conference on Machine Learning. pp. 4138–4148. PMLR (2020)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
15. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. In: International Joint Conference on Artificial Intelligence (IJCAI). pp. 2234–2240 (2018)
16. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4340–4349 (2019)

17. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: Amc: Automl for model compression and acceleration on mobile devices. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 784–800 (2018)
18. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
19. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al.: Searching for mobilenetv3. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 1314–1324 (2019)
20. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4700–4708 (2017)
21. Huang, Z., Wang, N.: Data-driven sparse structure selection for deep neural networks. In: Proceedings of the European conference on computer vision (ECCV). pp. 304–320 (2018)
22. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37. pp. 448–456. ICML, JMLR.org (2015), http://dl.acm.org/citation.cfm?id=3045118.3045167
23. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. arXiv preprint arXiv:1611.01144 (2016)
24. Kang, M., Han, B.: Operation-aware soft channel pruning using differentiable masks. In: International Conference on Machine Learning. pp. 5122–5131. PMLR (2020)
25. Kim, J., Park, C., Jung, H., Choe, Y.: Plug-in, trainable gate for streamlining arbitrary neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence (2020)
26. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
27. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)
28. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
29. Lee, N., Ajanthan, T., Torr, P.H.: Snip: Single-shot network pruning based on connection sensitivity. ICLR (2019)
30. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. ICLR (2017)
31. Li, Y., Lin, S., Liu, J., Ye, Q., Wang, M., Chao, F., Yang, F., Ma, J., Tian, Q., Ji, R.: Towards compact cnns via collaborative compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6438–6447 (2021)
32. Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., Shao, L.: Hrank: Filter pruning using high-rank feature map. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
33. Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.T., Sun, J.: Metapruning: Meta learning for automatic neural network channel pruning. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 3296–3305 (2019)
34. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: ICCV (2017)

35. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=rJlnB3C5Ym
36. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through $l_0$ regularization. In: International Conference on Learning Representations (2018), https://openreview.net/forum?id=H1Y8hhg0b
37. Molchanov, D., Ashukha, A., Vetrov, D.: Variational dropout sparsifies deep neural networks. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 2498–2507. JMLR. org (2017)
38. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 11264–11272 (2019)
39. Neklyudov, K., Molchanov, D., Ashukha, A., Vetrov, D.P.: Structured bayesian pruning via log-normal multiplicative noise. In: Advances in Neural Information Processing Systems. pp. 6775–6784 (2017)
40. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems. pp. 8024–8035 (2019)
41. Peng, H., Wu, J., Chen, S., Huang, J.: Collaborative channel pruning for deep networks. In: International Conference on Machine Learning. pp. 5113–5122 (2019)
42. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision. pp. 525–542. Springer (2016)
43. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 779–788 (2016)
44. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems. pp. 91–99 (2015)
45. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4510–4520 (2018)
46. Sehwag, V., Wang, S., Mittal, P., Jana, S.: Hydra: Pruning adversarially robust neural networks. In: NeurIPS (2020), https://proceedings.neurips.cc/paper/2020/hash/e3a72c791a69f87b05ea7742e04430ed-Abstract.html
47. Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. In: Advances in neural information processing systems. pp. 568–576 (2014)
48. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
49. Tang, Y., Wang, Y., Xu, Y., Tao, D., Xu, C., Xu, C., Xu, C.: Scop: Scientific control for reliable neural network pruning. Advances in Neural Information Processing Systems **33** (2020)
50. Wang, H., Qin, C., Zhang, Y., Fu, Y.: Neural pruning via growing regularization. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=o966_Is_nPA
51. Wang, Z., Li, C., Wang, X.: Convolutional neural network pruning with structural redundancy reduction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14913–14922 (2021)

52. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in neural information processing systems. pp. 2074–2082 (2016)
53. Ye, M., Gong, C., Nie, L., Zhou, D., Klivans, A., Liu, Q.: Good subnetworks provably exist: Pruning via greedy forward selection. In: International Conference on Machine Learning. pp. 10820–10830. PMLR (2020)
54. You, Z., Yan, K., Ye, J., Ma, M., Wang, P.: Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 2130–2141 (2019)
55. Yu, S., Mazaheri, A., Jannesari, A.: Auto graph encoder-decoder for neural network pruning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 6362–6372 (2021)
56. Zhang, D., Wang, H., Figueiredo, M., Balzano, L.: Learning to share: Simultaneous parameter tying and sparsification in deep learning (2018)
57. Zhang, Y., Gao, S., Huang, H.: Exploration and estimation for model compression. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 487–496 (2021)
58. Zhang, Y., Gao, S., Huang, H.: Recover fair deep classification models via altering pre-trained structure. In: Proceedings of the European Conference on Computer Vision (ECCV) (2022)
59. Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., Zhu, J.: Discrimination-aware channel pruning for deep neural networks. In: Advances in Neural Information Processing Systems. pp. 875–886 (2018)