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

Shangqian Gao[1], Yanfu Zhang[2], Feihu Huang[1], Heng Huang[3]*
[1] Electrical and Computer Engineering, University of Pittsburgh
[2] Computer Science, College of William and Mary
[3] Computer Science, University of Maryland College Park

Abstract

Most existing dynamic or runtime channel pruning methods have to store all weights to achieve efficient inference, which brings extra storage costs. Static pruning methods can reduce storage costs directly, but their performance is limited by using a fixed sub-network to approximate the original model. Most existing pruning works suffer from these drawbacks because they were designed to only conduct either static or dynamic pruning. In this paper, we propose a novel method to solve both efficiency and storage challenges via simultaneously conducting dynamic and static channel pruning for convolutional neural networks. We propose a new bi-level optimization based model to naturally integrate the static and dynamic channel pruning. By doing so, our method enjoys benefits from both sides, and the disadvantages of dynamic and static pruning are reduced. After pruning, we permanently remove redundant parameters and then finetune the model with dynamic flexibility. Experimental results on CIFAR-10 and ImageNet datasets suggest that our method can achieve state-of-the-art performance compared to existing dynamic and static channel pruning methods.

1. Introduction

Convolutional neural networks (CNNs) have recently achieved great successes in many machine learning and computer vision tasks [3, 37, 56, 57, 61]. Despite the remarkable performance, the computational and storage costs of most CNNs are quite expensive due to their complex architectures. Such costs have become the major bottleneck to deploying CNNs on portable devices with limited resources (*e.g.*, memory, CPU, energy). To solve this problem, many researchers focus on how to truncate the costs of deep models effectively. These researches can be summarized into several directions, such as weight pruning [24], weight quantization [6], structural pruning [39], matrix decomposition [10] and so on. Among these approaches, channel pruning, which belongs to structural pruning, is a promising way to effectively reduce computational and storage costs since other methods often require additional post-processing steps to acquire actual compression. Thus, this work focuses on investigating the channel pruning technique.

A series of channel pruning approaches [27, 50, 77] use different criteria to evaluate the importance of each channel, and the redundant (less important) channels are pruned. These approaches are also called static channel pruning. The benefit of static channel pruning is that unessential channels are permanently removed, which results in savings of both storage and computational costs. However, the model capacity of static pruning is restricted by using a fixed sub-network. Some more recent works [23, 45] try to select important channels based on inputs and intermediate feature maps at inference time, and they belong to dynamic channel pruning. Given different inputs, different sub-networks are dynamically selected, which largely improves the model capacity. Most existing dynamic pruning methods preserve all channels to ensure the model has the largest capacity. Compared to static pruning, dynamic pruning methods often achieve better performance but at the cost of requiring extra storage space.

As mentioned in recent storage efficient dynamic pruning work [5], the large storage costs of most dynamic pruning methods prohibit them from being deployed in resource-limited portable devices. To save storage costs for dynamic pruning, storage efficient pruning [5] heuristically combines static and dynamic channel pruning by using reinforcement learning. The final pruned model is obtained by combining the outputs of both static and dynamic pruning through a hand-designed function. Channels with low importance are permanently removed. Although this approach achieves good results, there are several drawbacks. First, the sub-networks from dynamic and static pruning in their method are treated separately. In their work, static sub-networks are not considered when conducting dynamic pruning and vice

*This work was partially supported by NSF IIS 2347592, 2347604, 2348159, 2348169, DBI 2405416, CCF 2348306, CNS 2347617.

versa, which generally hurts the performance. Moreover, the learning of position and importance of remaining channels are also separated. Second, they use a hand-designed function to fuse dynamic and static pruning results, leading to sub-optimal performance due to the lack of the learning process.

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

In our method, the selection of channels for both dynamic and static pruning is based on differentiable gates, and they can be optimized through backpropagation. Under this setting, we can apply parameter constraints on the static sub-network and FLOPs constraints on dynamic sub-networks. Previous dynamic pruning works [5, 45] often require hyper-parameters to implicitly specify the computational budget and/or the trade-off between dynamic and static pruning. But our method can set them directly, which is an additional benefit of our method.

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

- We propose a novel channel pruning method, which unifies both dynamic and static pruning. Dynamic and static sub-networks are connected by evaluating the static sub-network through dynamic sub-networks instead of training them in parallel.
- We integrate static and dynamic pruning by formulating them as a bi-level optimization problem. By doing so, our method enjoys benefits from both static and dynamic pruning. In addition, we present an efficient method for optimizing the matrix-vector product in bi-level optimization.
- The experimental results on CIFAR-10 and ImageNet datasets suggest that our method achieves state-of-the-art performance compared to existing dynamic and static pruning methods.

2. Related Works

2.1. Regular Pruning

Weight pruning. Weight pruning aims to eliminate redundant parameters. An early work [67] prunes model weights based on minimum description length. Optimal brain damage [38] and surgeon [25] utilize second-order information to remove connections. The drawback is that the computation of second-order derivatives is expensive. More recently, *Han et al.* [24] propose to prune weights based on their magnitude. Magnitude pruning is very efficient, and the cost of computing L_1 or L_2 magnitude is negligible. Regular network pruning approaches follow a three-stage pipeline: training, pruning, and fine-tuning. *Zhang et al.* [49] raise questions about such standard procedure and argue that the sub-network architecture obtained by pruning is more valuable than the remaining weights. They also show that retraining sub-networks from scratch is enough to recover the performance. On the other hand, the lottery ticket hypothesis (LTH) [15] shows that good sub-networks exist at the initialization stage. A series of works [52, 58] related to LTH extend this work to larger datasets and more complicated architectures. Another line of research [55, 76] shows that training masks on top of untrained models can also lead to ideal performance. The model after weight pruning has much fewer parameters but it requires sparse matrix libraries or specific hardware to achieve actual savings in storage and computational costs.

Structural Pruning. Structural pruning tries to remove certain structures in a deep model, such as kernels, channels, layers, and so on. In contrast to weight pruning, structural pruning can accelerate inference speed and save storage costs without additional effort. Filter pruning [39] tries to prune filters from CNNs that are having small effects on the outputs. Similar to magnitude pruning, the importance of each filter is measured by L_1 or L_2 norm of the filter, and L_1 norm performs better in their settings. Unlike filter pruning, soft filter pruning [28] does not remove filters during training, and they instead reset these filters and put them into training again. Network slimming [47] uses L_1 sparsity regularization on scaling factors of channels from batch normalization layers, and channels with small scaling factors are removed. Other related works [16–22, 33] also added learnable parameters for different structures. Discrimination-aware pruning [77] not only considers the norms of channels but also uses classification loss to identify unimportant channels. Automatic model compression [29] applies reinforcement learning (RL) for structural pruning. RL is used since it can better cooperate with the discrete nature of structural pruning. Greedy pruning [70] starts from an empty model and adds connections that reduce the loss value most. Static pruning methods directly reduce storage costs, but the pruned model is fixed leading to limited model capacity.

Alongside progress in vision tasks, Natural Language Processing (NLP) has significantly advanced, demonstrated by key studies [62, 68, 71–75]. Concurrently, structure pruning is enhancing large model efficiency [66].

2.2. Dynamic Pruning

Regular pruning methods are designed to find a fixed subnetwork for all inputs. On the other hand, dynamic pruning aims to provide different sub-networks for different inputs, which increases the model capacity given the same inference budget. Runtime neural pruning [42] treats dynamic pruning for different layers as a Markov decision process and uses reinforcement learning for training. SkipNet [65] uses a gating module to skip convolution blocks based on previous feature maps dynamically. The dynamic skipping problem is formulated as a sequential decision-making problem, which is jointly solved by reinforcement and supervised learning. Adaptive neural networks [4] adaptively select the components of a deep model based on the input examples. They also introduce an early exit mechanism to further reduce computational costs. In feature boosting and suppression [23], they propose to skip unimportant input and output channels dynamically. They use Lasso regularization to introduce sparsity on the runtime channel importance. Besides pruning, some works utilize the power of dynamic computation to improve the design of CNNs. CondConv [69] replace traditional convolutions with learned specialized convolutional kernels for each input. Dynamic convolution [7] applies input-dependent attention on multiple convolution kernels, which drastically improves the model capacity. Most aforementioned works need to keep the full model to achieve the best performance. To reduce storage costs, storage efficient dynamic pruning [5] introduces static pruning along with dynamic pruning to reduce storage costs.

3. Proposed Method

3.1. Notations

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

3.2. Static and Dynamic Settings

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

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

where $g_s \in R^{C_i}$ is the static pruning vector, $\mu \sim \text{Gumbel}(0, 1)$, τ is the temperature hyper-parameter, and $\theta_s \in R^{C_i}$ are learnable parameters for static pruning. v_s is a continuous vector, we further round it to its nearest neighbor g_s. The rounding function is not differentiable, we solve this problem by using the straight-through estimator [2] to calculate gradients. Now we have the binary vector g_s for static pruning. The generation of g_s can be seen as using the straight-through Gumbel-sigmoid [34] trick to approximate Bernoulli distribution.

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

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

where $g_d \in R^{B\times C_i}$ is the dynamic pruning vector, and $h(\cdot; \theta_d)$ is a routing function parameterized by θ_d to dynamically select channels, the rest settings are the same as static pruning. The routing function $h(\cdot; \theta_d)$ is composed with global average pooling followed by squeeze and excitation (SE) [31], which is suggested by FBS [23]. By using SE, we can save parameters when some layers in a model is too wide (like later layers of MobileNet-V2).

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

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

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

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

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

3.3. Unified Dynamic and Static Pruning

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

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

where Θ_s is the collection of all learnable parameters θ_s for static pruning, $\hat{T}_p$ is the number of all prunable parameters,

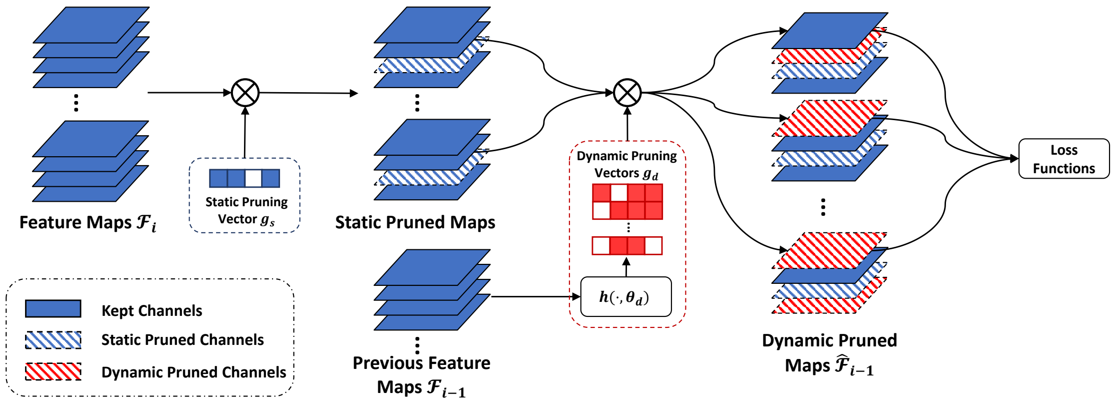

Feature Maps $\mathcal{F}_i$
Static Pruning Vector g_s
Static Pruned Maps
Previous Feature Maps $\mathcal{F}_{i-1}$
Dynamic Pruning Vectors g_d
$h(\cdot, \theta_d)$
Dynamic Pruned Maps $\hat{\mathcal{F}}_{i-1}$
Loss Functions
Kept Channels
Static Pruned Channels
Dynamic Pruned Channels

Figure 1. The flowchart of the proposed method. In the figure, we first conduct static pruning followed by dynamic pruning. Instead of naively combining static pruning and dynamic pruning, we formulate the pruning problem as a bi-level optimization problem to unify static and dynamic pruning. The whole process is differentiable, which allows efficient gradient based optimization.

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

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

where Θ_d is the collection of all θ_d, $\hat{T}_r$ is the total prunable FLOPs of the model, $T_r(\Theta_d)$ is the average FLOPs of B dynamic sub-networks, $\mathcal{R}_r$ is the regularization term to push average FLOPs of dynamic sub-networks to the corresponding threshold p_r, γ is the hyper-parameter for $\mathcal{R}_d$, and $\mathcal{R}_d$ is a regularization term to prevent dynamic sub-networks from collapsing to a single trivial solution. We define $R_d(v_d)$ as follows:

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

where v_d^i is the continuous dynamic vector of ith layer, and $\bar{v}_d^i = \frac{1}{B} \sum v_d^i$ is the average of dynamic vectors from the current layer. We use continuous v_d^i instead of discrete g_d^i. Because the variance of g_d^i could be very small for early layers (also pointed out in [69]), which results in instability and difficulty when optimizing Eq. 7.

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

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

To unify the training of dynamic and static sub-networks, we can consider the following bi-level optimization [8] problem:

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

where the outer problem is to find good dynamic sub-networks based on the optimal static sub-network, and the inner problem is getting the optimal static sub-network. The formulation of the problem in Eq. 8 also appeared in gradient-based hyperparameter optimization [1, 14] and differentiable neural architecture search [44].

The inner problem in Eq. 8 is not easy to solve since how to generate dynamic sub-networks without learning is unclear. Naive random sampling of dynamic sub-networks only produces trivial results. We then approximate Θ_s^* by training one step, and it has been proven effective in previous works [1, 44]. Multi-step approximation can also be used, but it will dramatically increase the computational costs since one has to perform backpropagation through multiple time steps. As a result, we chose to use the one-step approximation. To simplify notations, we use $\mathcal{L}(\Theta_s, \Theta_d) = \mathcal{L}(f(x; \Theta_s, \Theta_d), y)$ for the following derivations. Let us first define the update rule u for Θ_s: $\Theta_s^{'} = u(\Theta_s, \eta)$, and η is the learning rate. Take SGD as an example, the update rule of Θ_s is $\Theta_s^{'} = u(\Theta_s, \eta) = \Theta_s - \eta(\nabla_{\Theta_s} \mathcal{L}(\Theta_s, \Theta_d) + \lambda \nabla_{\Theta_s} \mathcal{R}_p)$.

We then approximate Θ_s^* with $\Theta_s^{'}$, and the gradient with respect to Θ_d is:

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

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

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

To simplify derivation, θ_d^i is flattened as a vector, and we omit all transpose notations. The result of Eq. 10 indicates that the second-order term is just the multiplication of two Jacobians matrices followed by $\nabla_{g^i}\mathcal{L}(\theta_s^i, \theta_d^i)$, which is more efficient than using finite difference approximation for the matrix-vector product [44, 60]. The calculation of the Jacobians matrix is also simple since the computation of g_s^i and g_d^i only includes simple operations like matrix multiplications and element-wise functions. With Eq. 9 and Eq, 10, we always update Θ_d by taking Θ_s into consideration, which is ignored in storage efficient pruning [5]. The calculation of the Jacobians matrix is listed in the supplementary materials. In practice, we use the Adam optimizer [35] instead of SGD. The derivation of gradients w.r.t Θ_d given the Adam optimizer is also provided in the supplementary materials.

3.4. The Overall Algorithm

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

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

where f' is the pruned model with around $p_p\hat{T}_p$ parameters. Here, we abuse notations Θ_d and $\mathcal{W}$ to represent weights after static pruning, and they are different from the original weights. We only modify the feature maps during fine-tuning, which takes advantage of mini-batch training. During the evaluation, we dynamically prune the channels. For both pruning and fine-tuning, we choose $R_r(x,y) = R_p(x,y) = \log(\max(x,y)/y)$. Typically, regular regression loss functions, like MAE and MSE, can be used for R_r and R_p, but they can hardly achieve target values for some architectures like MobileNet-V2. We insert g_s and g_d after the Conv-Bn-ReLU block and before the next convolution layer for pruning, which can accurately reflect the pruned model. The overall algorithm of our method is provided in Alg. 1. The whole process of our method is summarized in Fig. 1.

Algorithm 1: Unified Dynamic and Static Channel Pruning

```
Input: dataset for pruning: $D_{\text{prune}}$; remained rate of
  FLOPs and parameters: $p_r$ and $p_p$; hyper-parameter:
  $\lambda$ and $\gamma$; training epochs for pruning: $E_{\text{prune}}$;
  pre-trained CNN: $f$.
Initialization: initialize $\Theta_d$ randomly; initialize $\Theta_s$
  uniformly; freeze $\mathcal{W}$ in $f$.
for $e := 1$ to $E_{prune}$ do
    for a mini-batch $(x, y)$ in $D_{prune}$ do
        1. produce static and dynamic vectors: $g_s$ and
          $g_d$. (Eq. 1 and 2)
        2. calculate gradients w.r.t $\Theta_s$ from Eq. 5.
        3. update $\Theta_s$ by Adam optimizer.
        4. calculate gradients w.r.t $\Theta_d$ (Eq. 9 and
          Eq. 10).
        5. update $\Theta_d$ by Adam optimizer.
    end
end
Get $f'$ by pruning $f$ based on $g_s$.
Return $f'$ for fine-tuning.
```

4. Experiments

4.1. Settings

In the experiment section, we call our method UDSP (**U**nified **D**ynamic and **S**tatic channel **P**runing). We use CIFAR-10 [36] and ImageNet [9] to verify the performance of our method, as most previous pruning works use these datasets.

Method	Architectures	Dynamic	Base Acc	Acc	Δ-Acc	↓FLOPs	↓ #Params
FBS [23]	CifarNet	✓	91.37%	89.88%	-1.49%	74.6%	-11.0%
SEP-A [5]		✓	92.07%	91.23%	-0.84%	74.5%	**22.0%**
SEP-B [5]		✓	92.07%	91.42%	-0.65%	74.5%	-31.0%
UDSP (ours)		✓	92.36%	**91.89%**	**-0.47%**	**75.1%**	20.1%
AMC [29]	ResNet-56	✗	92.80%	91.90%	-0.90%	50.0%	-
FPGM [30]		✗	93.59%	92.93%	-0.66%	**52.6%**	-
HRank [43]		✗	93.26%	93.17%	-0.21%	50.6%	**42.4%**
DSA [53]		✗	93.13%	92.91%	-0.22%	52.2%	-
SEP [5]		✓	93.12%	93.44%	+0.32%	50.0%	19.8%
UDSP (ours)		✓	93.12%	**93.78%**	**+0.66%**	50.1%	20.0%

Table 1. Comparison of the accuracy changes (Δ-Acc), reduction in FLOPs, and the number of parameters of various channel pruning algorithms on CIFAR-10. '+/-' of Δ-Acc indicates increase/decrease compared to baselines. '-' in '↓ #Params' indicates increase of parameters.

On CIFAR-10, we use CifarNet following several dynamic pruning works [5, 23]. Besides CifarNet, we also test our method on ResNet-56. For ImageNet, we evaluate our method on ResNets [26] and MobileNet-V2 [59]. p_p and p_r are used to decide how much FLOPs and parameters to be pruned. Detailed settings of p_p and p_r are provided in the supplementary materials. λ and γ in Eq. 5 and Eq. 6 are set to 2.0 and 0.1 separately for all models and datasets. τ in Eq. 1 and Eq. 2 is set to 0.4. Other implementation details are given in the supplementary materials.

4.2. CIFAR-10 Results

We present CIFAR-10 results in Tab. 1. For CifarNet, all comparison methods are dynamic. From Tab. 1, we can see that our method can outperform other comparison methods with similar pruned FLOPs. Compared to FBS, our method saves 27.9% of parameters (79.9% vs. 111% #Params compared to the original model) while achieving 1.02% improvements with Δ-Acc. SEP-A has similar parameter savings as our method, but the Δ-Acc is 0.37% lower than our method. SEP-B keeps all channels, and our method still outperforms it by 0.18% with Δ-Acc. Moreover, our method only uses 60.9% parameters of SEP-B.

We compare our method with both static and dynamic pruning methods on ResNet-56. All comparison methods reduce around 50% FLOPs. Our approach has similar pruning rates of FLOPs and parameters as SEP, but our method performs better than SEP by 0.34%. HRank achieves the best performance among static pruning methods. Static pruning methods prune more parameters compared to dynamic pruning methods, but the performance of our method is 0.87% higher than HRank in terms of Δ-Acc. In summary, our method achieves a better trade-off between storage costs and performance than SEP [5].

4.3. ImageNet Results

On the ImageNet dataset, we use ResNet-18, ResNet-34, ResNet-50, and MobileNet-V2 to evaluate the performance of different methods. All results are shown in Tab. 2. The results of other comparison baselines are directly adapted from their original paper following the common practice.

ResNet-18. For static pruning methods, DSA [53] achieves the best performance. The Δ Top-1 accuracy of our method is 0.83% higher than DSA, and our method prunes 10.2% more FLOPs. This result suggests that dynamic pruning still has advantages when the model capacity is reduced to some extent. FBS and CGNN use additional parameters for dynamic pruning. Our method outperforms FBS and CGNN by 2.26% and 0.79% in terms of Δ Top-1 accuracy separately. In addition, our method prunes 11.5% more FLOPs than CGNN and saves 20% of parameters. Finally, our method is better than SEP by 0.75% in terms of Δ Top-1 accuracy, while both methods prune similar FLOPs and parameters.

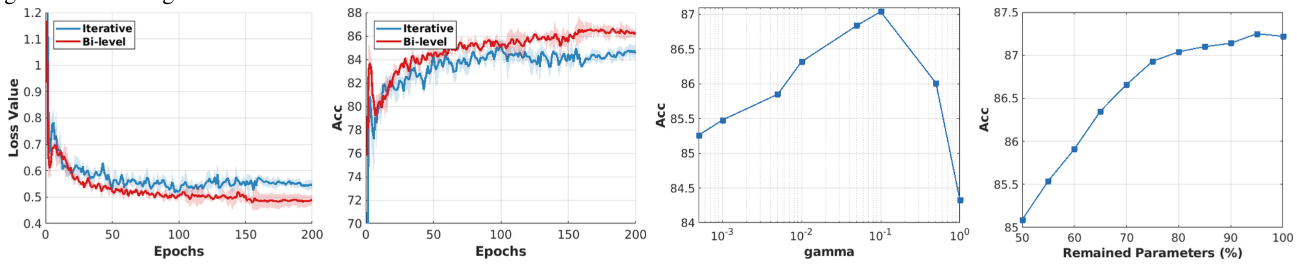
ResNet-34. IE [51] performs better than other static pruning methods, but it prunes less FLOPs and parameters. Our method has similar parameters and performance as IE, but we can prune 27.7% more FLOPs than IE. Our method saves 20% parameters and performs better than CGNN by 0.71% in terms of Δ Top-1 accuracy, and both methods prune similar FLOPs.

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

MobileNet-V2. AMC, MetaPruning and MobileNet-V2 0.75 all remove around 30% FLOPs. MetaPruning achieves the lowest accuracy lost. Our method prunes around 6% more FLOPs than MetaPruning, and performs better (0.52% and 0.44% higher with Top-1 and Δ Top-1 accuracy). Our

Method	Architectures	Dynamic	Pruned Top-1	Δ Top-1	↓ FLOPs	↓ #Params
AMC [29]	ResNet-18	✗	66.63%	-3.13%	50.0%	24.0%
FPGM [30]		✗	68.41%	-1.87%	41.5%	**28.0%**
DSA [53]		✗	68.61%	-1.11%	40.0%	-
FBS [23]		✓	68.17%	-2.54%	49.5%	-12.0%
CGNN [32]		✓	67.95%	-1.07%	38.7%	-
SEP [5]		✓	68.73%	-1.03%	48.5%	19.0%
FTWT [12]		✓	67.49%	-2.27%	**51.6%**	-
UDSP (ours)		✓	**69.48%**	**-0.28%**	50.2%	20.0%
SFP [28]	ResNet-34	✗	71.84%	-2.09%	41.1%	-
FPGM [30]		✗	72.63%	-1.29%	41.5%	**28.9%**
IE [51]		✗	72.83%	-0.48%	22.3%	21.1%
CGNN [32]		✓	72.40%	-1.10%	**50.4%**	-
FTWT [12]		✓	72.17%	-1.13%	47.4%	-
UDSP (ours)		✓	**72.91%**	**-0.39%**	50.0%	20.0%
SCOP [63]	ResNet-50	✗	75.26%	-0.89%	54.6%	51.8%
GFP [46]		✗	76.42%	-0.37%	51.0%	55.8%
3DP [64]		✗	75.90%	-0.25%	53.0%	50.0%
ResRe [11]		✗	75.97%	-0.12%	56.1%	-
DepGraph [13]		✗	75.97%	-0.12%	51.18%	-
DTP [41]		✗	75.55%	-0.58%	56.7%	-
UDSP (ours)		✓	**76.51%**	**+0.38%**	**58.4%**	25.0%
MobileNet-V2 0.75 [59]	MobileNet-V2	✗	69.80%	-2.00%	30.0%	**24.8%**
AMC [29]		✗	70.80%	-1.10%	30.0%	17.2%
MetaPruning [48]		✗	71.20%	-0.60%	30.9%	-
GSS [70]		✗	71.20%	-0.80%	36.0%	22.9%
UDSP (ours)		✓	**71.72%**	**-0.16%**	**36.6%**	15.3%

Table 2. Comparison of the accuracy changes (Δ Top-1), reduction in FLOPs, and the number of parameters of various channel pruning algorithms on ImageNet.



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

Figure 2. (a,b): Comparison of loss and accuracy given different training settings. Mean and variance are provided by running the experiment 3 times. (c) Impact of γ during the pruning process. (d) Impact of p_s during the pruning process. All results are obtained with ResNet-56 on CIFAR-10.

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

Table 3. Comparisons between different pruning settings of our algorithm on ResNet-56 for the CIFAR-10 dataset.

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

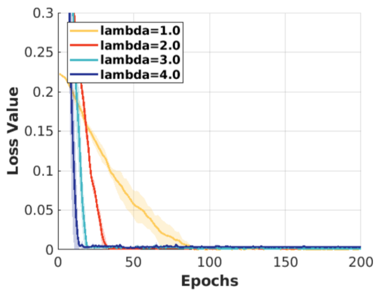
Table 4. Comparisons given different pruning rates for #Params with ResNet-56 on CIFAR-10.

method and GSS prune a similar amount of FLOPs, and the Δ Top-1 accuracy of our method is higher than GSS by 0.64%. In addition to FLOPs reduction, our method can also remove around 15.3% of parameters.

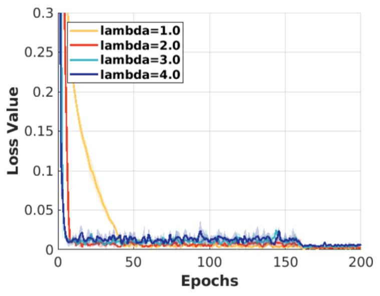
In summary, our method provides a larger model capacity compared to static pruning methods, and the storage costs are reduced compared to dynamic pruning methods. Moreover, our method achieves a better trade-off between storage costs and performance than SEP, indicating that integrating dynamic and static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

4.4. Analysis of Different Settings

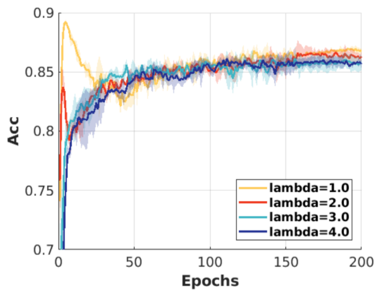
To understand different design choices and hyper-parameter settings, we provide additional analysis in this section. In Fig. 2(a,b), we plot the loss value and model accuracy given



(a) $\mathcal{R}_p$ Loss

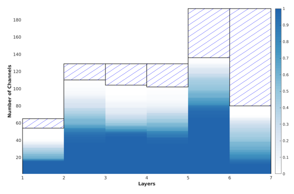


(b) $\mathcal{R}_r$ Loss

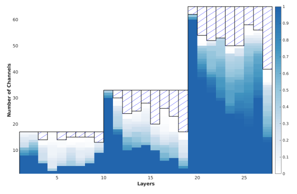


(c) Accuracy

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



(a) CifarNet



(b) ResNet-56

Figure 4. The resulting architectures of ResNet-56 and CifarNet on CIFAR-10 with our method. We plot the probability of using each channel, and the probability is calculated on the whole test dataset. Channels with dashed lines are permanently removed.

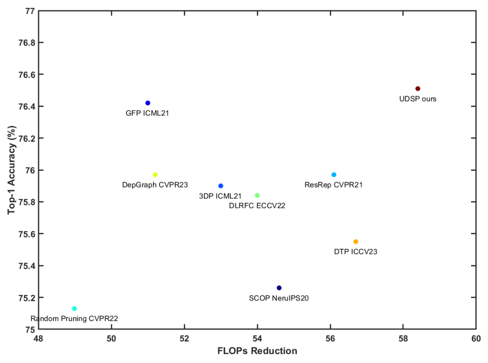


Figure 5. ResNet-50 on ImageNet dataset.

different pruning settings. We can see that bi-level optimization outperforms iterative training with both accuracy and loss values, which suggests that integrating dynamic and static pruning is beneficial. We also show the difference between the finetuned model in Tab. 3, and we can draw similar conclusions.

In Fig. 2(c), we provide the accuracy after pruning (before fine-tuning) given different γ. A too-large γ usually hurts the performance, and γ around 0.1 provides relatively good results.

In Fig. 2(d), we fix $p_r = 0.5$ and plot the accuracy after pruning given different percentages of remaining parameters (p_p). We can see that the performance does not decrease a lot when we keep more than 75% of parameters. We further present results when pruning more parameters after finetuning in Tab 4. When pruning 30% of parameters (UDSP^2), the performance of our method does not decrease too much. However, there is a large performance drop when pruning 40% of parameters. Under this setting (UDSP^1), the parameter reduction of our method is similar to the static pruned model from HRank, and the dynamic flexibility is largely restricted. These observations suggest that, under the same FLOPs pruning rate, our method can maintain a good trade-off between dynamic flexibility and storage costs until the pruning rate for parameters is similar to static pruning methods.

In Fig. 3, we plot the value of regularization losses and model accuracy given different choices of λ. From the figure, it can be seen that our method is robust to different choices of λ. A lower λ can lead to a little better final performance, but the difference is small.

In Fig. 4, we plot the final architectures of ResNet-56 and CifarNet for our method. Our method tends to preserve more channels when the width of the original model changes. Later layers often have more dynamic flexibility, probably because they are less penalized by the FLOPs constraint $\mathcal{R}_r$. This figure also suggests that our method does not collapse into a single static solution.

We plot the Top-1 accuracy vs. FLOPs in Fig. 5. Besides baselines introduced in Tab. 2, we also include Random Pruning [40] in the figure. In the figure, it is clear that our method has the best FLOPs vs. Accuracy trade-off.

5. Conclusion

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

References

[1] Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*, 2018. 4

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

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. 1

[4] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*, pages 527–536. PMLR, 2017. 3

[5] Jianda Chen, Shangyu Chen, and Sinno Jialin Pan. Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 14747–14758. Curran Associates, Inc., 2020. 1, 2, 3, 5, 6, 7

[6] 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

[7] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11030–11039, 2020. 3

[8] Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007. 4

[9] 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

[10] Misha Denil, Babak Shakibi, Laurent Dinh, Marc' Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, 2013. 1

[11] Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4510–4520, 2021. 7

[12] Sara Elkerdawy, Mostafa Elhoushi, Hong Zhang, and Nilanjan Ray. Fire together wire together: A dynamic pruning approach with self-supervised mask prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 7

[13] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16091–16101, 2023. 7

[14] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR, 2018. 4

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

[16] Alireza Ganjdanesh*, Shangqian Gao*, and Heng Huang. Interpretations steered network pruning via amortized inferred saliency maps. In *European Conference on Computer Vision*, pages 278–296. Springer, 2022. 2

[17] Alireza Ganjdanesh, Shangqian Gao, and Heng Huang. Effconv: efficient learning of kernel sizes for convolution layers of cnns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7604–7612, 2023.

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

[19] 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.

[20] Shangqian Gao, Feihu Huang, Yanfu Zhang, and Heng Huang. Disentangled differentiable network pruning. In *European Conference on Computer Vision*, pages 328–345. Springer, 2022.

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

[22] Shangqian Gao, Zeyu Zhang, Yanfu Zhang, Feihu Huang, and Heng Huang. Structural alignment for network pruning through partial regularization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17402–17412, 2023. 2

[23] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng zhong Xu. Dynamic channel pruning: Feature boosting and suppression. In *International Conference on Learning Representations*, 2019. 1, 3, 6, 7

[24] 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

[25] Babak Hassibi and David G Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993. 2

[26] 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. 6

[27] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings*

of the IEEE International Conference on Computer Vision, pages 1389–1397, 2017. 1

[28] 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, 7

[29] 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. 2, 6, 7

[30] 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. 6, 7

[31] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 3

[32] Weizhe Hua, Yuan Zhou, Christopher M De Sa, Zhiru Zhang, and G. Edward Suh. Channel gating neural networks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. 7

[33] 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

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

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5, 1

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

[37] 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

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

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

[40] Yawei Li, Kamil Adamczewski, Wen Li, Shuhang Gu, Radu Timofte, and Luc Van Gool. Revisiting random channel pruning for neural network compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 191–201, 2022. 8

[41] Yunqiang Li, Jan C van Gemert, Torsten Hoefler, Bert Moons, Evangelos Eleftheriou, and Bram-Ernst Verhoef. Differentiable transportation pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16957–16967, 2023. 7

[42] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017. 3

[43] 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

[44] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 4, 5

[45] Liu Liu, Lei Deng, Xing Hu, Maohua Zhu, Guoqi Li, Yufei Ding, and Yuan Xie. Dynamic sparse graph for efficient deep learning. In *International Conference on Learning Representations*, 2019. 1, 2

[46] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021. 7

[47] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017. 2

[48] 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. 7

[49] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019. 2

[50] Jian-Hao Luo, Hao Zhang, Hong-Yu Zhou, Chen-Wei Xie, Jianxin Wu, and Weiyao Lin. Thinet: pruning cnn filters for a thinner net. *IEEE transactions on pattern analysis and machine intelligence*, 2018. 1

[51] 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. 6, 7

[52] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems 32*, pages 4932–4942. Curran Associates, Inc., 2019. 2

[53] Xuefei Ning, Tianchen Zhao, Wenshuo Li, Peng Lei, Yu Wang, and Huazhong Yang. Dsa: More efficient budgeted pruning via differentiable sparsity allocation. *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 6, 7

[54] 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. 1

[55] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11893–11902, 2020. 2

[56] 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

[57] 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

[58] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2020. 2

[59] 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. 6, 7, 1

[60] Zebang Shen, Alejandro Ribeiro, Hamed Hassani, Hui Qian, and Chao Mi. Hessian aided policy gradient. In *International Conference on Machine Learning*, pages 5729–5738. PMLR, 2019. 5

[61] 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

[62] Hannah Smith, Zeyu Zhang, John Culnan, and Peter Jansen. ScienceExamCER: A high-density fine-grained science-domain corpus for common entity recognition. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4529–4546, Marseille, France, 2020. European Language Resources Association. 3

[63] Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chunjing Xu, Chao Xu, and Chang Xu. Scop: Scientific control for reliable neural network pruning. *Advances in Neural Information Processing Systems*, 33, 2020. 7

[64] Wenxiao Wang, Minghao Chen, Shuai Zhao, Long Chen, Jinming Hu, Haifeng Liu, Deng Cai, Xiaofei He, and Wei Liu. Accelerate cnns from three dimensions: A comprehensive pruning framework. In *International Conference on Machine Learning*, pages 10717–10726. PMLR, 2021. 7

[65] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018. 3

[66] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162, Online, 2020. Association for Computational Linguistics. 3

[67] 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

[68] Dongfang Xu, Zeyu Zhang, and Steven Bethard. A generate-and-rank framework with semantic type regularization for biomedical concept normalization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8452–8464, Online, 2020. Association for Computational Linguistics. 3

[69] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. *NeurIPS*, 2019. 3, 4

[70] 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. 2, 7

[71] Zeyu Zhang and Steven Bethard. Improving toponym resolution with better candidate generation, transformer-based reranking, and two-stage resolution. In *Proceedings of the 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023)*, pages 48–60, Toronto, Canada, 2023. Association for Computational Linguistics. 3

[72] Zeyu Zhang, Thuy Vu, and Alessandro Moschitti. Joint models for answer verification in question answering systems. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3252–3262, Online, 2021. Association for Computational Linguistics.

[73] Zeyu Zhang, Thuy Vu, Sunil Gandhi, Ankit Chadha, and Alessandro Moschitti. Wdrass: A web-scale dataset for document retrieval and answer sentence selection. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, page 4707–4711, New York, NY, USA, 2022. Association for Computing Machinery.

[74] Zeyu Zhang, Thuy Vu, and Alessandro Moschitti. In situ answer sentence selection at web-scale. *arXiv preprint arXiv:2201.05984*, 2022.

[75] Zeyu Zhang, Thuy Vu, and Alessandro Moschitti. Double retrieval and ranking for accurate question answering. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1751–1762, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. 3

[76] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, 2019. 2

[77] 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

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

Supplementary Material

A. Implementation Details

We train ResNet-56 and CifarNet on CIFAR-10 from scratch following pytorch [54] examples. For ImageNet models, we can directly use pre-trained models from Pytorch, since our method is able to prune any pre-trained models.

To improve efficiency, we only use part of the dataset for pruning. We randomly sample 5,000 and 50,000 images from CIFAR-10 and ImageNet as D_{prune} in Alg. 1. Adam [35] optimizer is used to train both Θ_s and Θ_d, and the training lasts for 200 epochs with mini-batchsize 256. The start learning rate and weight decay are set to 10^{-3} and 10^{-4}, and the learning rate is decayed to 10^{-4} at epoch 160. We initialize all Θ_s to 3.0 to ensure most initial g_s are 1.0, which means that static pruning starts from the whole network.

On CIFAR-10, we finetune the model for 160 epochs by using the Adam optimizer with a start learning rate 10^{-3}. The learning rate is changed to 10^{-4} at epoch 80, and it is further reduced to 10^{-5} at epoch 120. Following storage efficient pruning [5], we continue to use the Adam optimizer on ImageNet models. After pruning, we finetune ResNet models for 100 epochs with a start learning rate 10^{-3}. The learning rate is then decayed to 10^{-4} at epoch 30, and it is further decayed to 10^{-5} and 10^{-6} at epoch 60 and epoch 90. For MobileNet-V2, we also use the Adam optimizer and finetune it for 100 epochs. We use the cos-annealing learning rate scheduler following their original setting [59]. The mini-batch size and weight decay are 256 and 10^{-4} for both CIFAR-10 and ImageNet models. All codes are implemented with pytorch [54]. The experiments are conducted on a machine with 4 Nvidia Tesla P40 GPUs.

B. Visualization of the Resulting Architectures

In Fig. 6, we further plot the final architectures of ResNet-56 and CifarNet for our method and SEP. For both CifarNet and ResNet-56, SEP does not fully utilize the capacity of early layers, especially on ResNet-56. These results justify why our method can outperform SEP. This also suggests that a more sophisticated interaction (bi-level optimization) between static and dynamic sub-networks is crucial to achieving good results.

C. Negative Impacts of Joint Training

In section 3.3 of the main text, we argue that joint training of dynamic and static pruning will interfere with each other. In Fig. 7, we present the comparison results between joint training and iterative training. We can see that joint training lacks exploration during learning and its performance is lower than the iterative baseline.

D. Derivation of Gradients w.r.t Θ_d from Adam

Algorithm 2: Update Θ_s with Adam

```
Input: $\eta, \beta_1, \beta_2 \in (0, 1]$, $\epsilon \geq 0$: learning rate and
  decay rate for ADAM.
Initialize $m_0, n_0, t = 0$
Update rule at step $t$:
$m_t =$
  $\beta_1 m_{t-1} + (1-\beta_1)(\nabla_{\Theta_s}\mathcal{L}(\Theta_s, \Theta_d) + \lambda\nabla_{\Theta_s}\mathcal{R}_p)$
$n_t =$
  $\beta_2 n_{t-1} + (1-\beta_2)(\nabla_{\Theta_s}\mathcal{L}(\Theta_s, \Theta_d) + \lambda\nabla_{\Theta_s}\mathcal{R}_p)^2$
$\hat{m}_t = m_t/(1-\beta_1^t)$
$\hat{n}_t = n_t/(1-\beta_2^t)$
$\Theta'_s = u(\Theta_s, \eta) = \Theta_s - \eta\hat{m}_t/(\sqrt{\hat{n}_t} + \epsilon)$
```

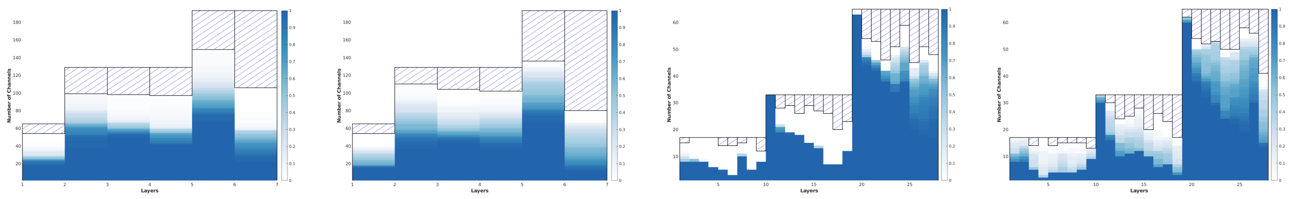
In Eq. 9 of our paper, we provide the gradients w.r.t Θ_d when updating Θ_s with SGD, since it's simple and easy to follow. In practice, SGD can hardly achieve satisfactory performance when dealing with discrete values. As a result, we use Adam to update Θ_s. As a result, we will show how to calculate the gradients of Θ_d under the Adam optimizer. We show the update rule of Θ_s in Al. 2, and we omit timestep t of Θ_s to simplify notations. We focus on the first term in Eq. 9, and the gradient w.r.t Θ_d is:

$$\begin{aligned}
&\nabla_{\Theta_d}\mathcal{L}(\Theta_s^*, \Theta_d) \\
\approx&\nabla_{\Theta_d}\mathcal{L}(\Theta_s - \eta\hat{m}_t/(\sqrt{\hat{n}_t} + \epsilon), \Theta_d) \\
=&\nabla_{\Theta_d}\mathcal{L}(\Theta'_s, \Theta_d) - \eta\nabla^2_{\Theta_d,\Theta_s}\mathcal{L}(\Theta_s, \Theta_d)((\frac{\hat{\beta}_1}{2(\sqrt{\hat{n}_t} + \epsilon)} \\
&- \frac{\hat{\beta}_2\hat{m}_t(\nabla_{\Theta_s}\mathcal{L}(\Theta_s, \Theta_d) + \lambda\nabla_{\Theta_s}\mathcal{R}_p)}{2(\hat{n}_t + \epsilon\sqrt{\hat{n}_t})^2})\nabla_{\Theta'_s}\mathcal{L}(\Theta'_s, \Theta_d)),
\end{aligned} \tag{12}$$

where $\hat{\beta}_1 = \frac{1-\beta_1}{1-\beta_1^t}$, $\hat{\beta}_2 = \frac{1-\beta_2}{1-\beta_2^t}$. The derivation in Eq. 12 is a little bit complicated compared to the SGD update, but it is still the result of the chain rule.

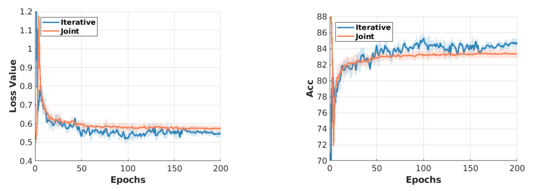
E. Calculation of Jacobian Matrix

Recall that we need to calculate $\nabla_{\theta_s^i} g_s^i \cdot \nabla_{\theta_d^i} g_d^i$ in Eq. 10. Let us first focus on the element-wise function $g_s^i = \lfloor v_s^i \rceil$



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

Figure 6. The resulting architectures of ResNet-56 and CifarNet on CIFAR-10 with our method and SEP. We plot the probability of using each channel, and the probability is calculated on the whole test dataset. Channels with dashed lines are permanently removed.



(a) Loss (b) Accuracy

Figure 7. (a,b): Comparison of loss and accuracy given joint and iterative training. Mean and variance are provided by running the experiment 3 times.

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

Table 5. Choice of p_r and p_p.

and $v_s^i = \sigma((\theta_s^i + \mu)/\tau)$, and we have:

$$\nabla_{\theta_s^i} g_s^i = \mathbf{I}\nabla_{\theta_s^i} v_s^i = \nabla_{\theta_s^i} v_s^i, \tag{13}$$

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

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

$$\nabla_{\theta_d^i} g_d^i = \nabla_{z_i} v_s^i \nabla_{\theta_d^i} z_i, \tag{14}$$

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

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

$$(\nabla_{\theta_s^i} g_s^i \cdot \nabla_{\theta_d^i} g_d^i)_{[p,:,:]} = ((\nabla_{\theta_s^i} g_s^i)^\intercal_{[p,:]} (\nabla_{\theta_d^i} g_d^i)_{[p,:]})^\intercal. \tag{15}$$

F. Selections of p_r and p_p

We present the choices of p_r and p_p in Tab. 5. The choices of p_r and p_p are not hard to calculate. Let T_p^{all} be the number of all parameters given a CNN, and $\hat{T}_p$ is the total number of prunable parameters. If we want to remove 20% of parameters, then $p_p\hat{T}_p = 0.8T_p^{\text{all}}$. Finally, $p_p = 0.8T_p^{\text{all}}/\hat{T}_p$. Similar calculations can be applied on p_r.