MODEL COMPRESSION VIA HYPER-STRUCTURE NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we propose a novel channel pruning method to solve the problem of compression and acceleration of Convolutional Neural Networks (CNNs). Previous channel pruning methods usually ignore the relationships between channels and layers. Many of them parameterize each channel independently by using gates or similar concepts. To fill this gap, a hyper-structure network is proposed to generate the architecture of the main network. Like the existing hypernet, our hyper-structure network can be optimized by regular backpropagation. Moreover, we use a regularization term to specify the computational resource of the compact network. Usually, FLOPs is used as the criterion of computational resource. However, if FLOPs is used in the regularization, it may over penalize early layers. To address this issue, we further introduce learnable layer-wise scaling factors to balance the gradients from different terms, and they can be optimized by hyper-gradient descent. Extensive experimental results on CIFAR-10 and ImageNet show that our method is competitive with state-of-the-art methods.

1 INTRODUCTION

Convolutional Neural Networks (CNNs) have accomplished great success in many machine learning and computer vision tasks (Krizhevsky et al., 2012; Redmon et al., 2016; Ren et al., 2015; Simonyan & Zisserman, 2014a; Bojarski et al., 2016). To deal with real world applications, recently, the design of CNNs becomes more and more complicated in terms of width, depth, etc. (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014b; He et al., 2016; Huang et al., 2017). Although these complex CNNs can attain better performance on benchmark tasks, their computational and storage costs increase dramatically. As a result, a typical application based on CNNs can easily exhaust an embedded or mobile device due to its enormous costs. Given such costs, the application can hardly be deployed on resource-limited platforms. To tackle these problems, many methods (Han et al., 2015b;a) have been devoted to compressing the original large CNNs into compact models. Among these methods, weight pruning and structural pruning are two popular directions.

Unlike weight pruning or sparsification, structural pruning, especially channel pruning, is an effective way to truncate the computational cost of a model because it does not require any post-processing steps to achieve actual acceleration and compression. Many existing works (Liu et al., 2017; Ye et al., 2018; Huang & Wang, 2018; Kim et al., 2020; You et al., 2019) try to solve the problem of structure pruning by applying gates or similar concepts on channels of a layer. Although these ideas have achieved many successes in channel pruning, there are some potential problems. Usually, each gate has its own parameter, but parameters from different gates do not have dependence. As a result, they can hardly learn inter-channel or inter-layer relationships. Due to the same reason, the slimmed models from these methods could overlook the information between different channels and layers, potentially bringing sub-optimal model compression results.

To address these challenges, we propose a novel channel pruning method inspired by hypernet (Ha et al., 2016). In hypernet, they propose to use a hyper network to generate the weights for another network, while the hypernet can be optimized through backpropagation. We extend a hypernet to a hyper-structure network to generate an architecture vector for a CNN instead of weights. Each architecture vector corresponds to a sub-network from the main (original) network. By doing so, the inter-channel and inter-layer relationships can be captured by our hyper-structure network.

Besides the hyper-structure network, we also introduce a regularization term to control the computational budget of a sub-network. Recent model compression methods focus on pruning computational FLOPs instead of parameters. The problem of applying FLOPs regularization is that the gradients of the regularization will heavily penalize early layers which can be regarded as a bias towards latter layers. Such a bias will restrict the potential search space of sub-networks. To make our hyper-structure network explore more possible structures, we further introduce layer-wise scaling factors to balance the gradients from different losses for each layer. These factors can be optimized by hyper-gradient descent.

Our contributions are summarized as follows:

1) Inspired by hypernet, we propose to use a hyper-structure network for model compression to capture inter-channel and inter-layer relationships. Similar to hypernet, the proposed hyper-structure network can be optimized by regular backpropagation.
2) Gradients from FLOPs regularization are biased toward latter layers, which truncate the potential search space of a sub-network. To balance the gradients from different terms, layer-wise scaling factors are introduced for each layer. These scaling factors can be optimized through hyper-gradient descent with trivial additional costs.
3) Extensive experiments on CIFAR-10 and ImageNet show that our method can outperform both conventional channel pruning methods and AutoML based pruning methods on ResNet and MobileNetV2.

2 Related Works

2.1 Model Compression

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

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, Han et al. (2015b) proposes to use L_1 or L_2 magnitude as the criterion to prune weights and connections. SNIP (Lee et al., 2019) updates the importance of each weight by using gradients from loss function. Weights with lower importance will be pruned. Lottery ticket hypothesis (Frankle & Carbin, 2019) assumes there exist high-performance sub-networks within the large network at initialization time. They then retrain the sub-network with the same initialization. In rethinking network pruning (Liu et al., 2019b), they challenging 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.

One of the previous works (Li et al., 2017) 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 (Wen et al., 2016) 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 (Zhang et al., 2018). One of the problems when using Group Lasso is that weights with small values could still be important, and it's difficult for structures under Group Lasso regularization to achieve exact zero values. As a result, Louizos et al. (2018) propose to use explicit L_0 regularization to make weights within structures have exact zero values. Besides using the magnitude of structure weights as a criterion, other methods utilize the scaling factor of batchnorm to achieve structure pruning, since batchnorm (Ioffe & Szegedy, 2015) is widely used in recent neural network designs (He et al., 2016; Huang et al., 2017). A straight forward way to achieve channel pruning is to make the scaling factor of batchnorm to be sparse (Liu et al., 2017). If the scaling factor of a channel fell below a certain threshold, then the channel will be removed. The scaling factor can also be regarded as the gate parameter of a channel. Methods related to this concept include (Ye et al., 2018; Huang & Wang, 2018; Kim et al., 2020; You et al., 2019). Though it has achieved many successes in channel pruning, using gates can not capture the relationships between channels and across layers. Besides using gates, Collaborative channel pruning (Peng et al., 2019) try to prune channels by using Taylor expansion. Our method is also related to Automatic Model Compression(AMC) (He et al., 2018b). In AMC, they use policy gradient to update the policy

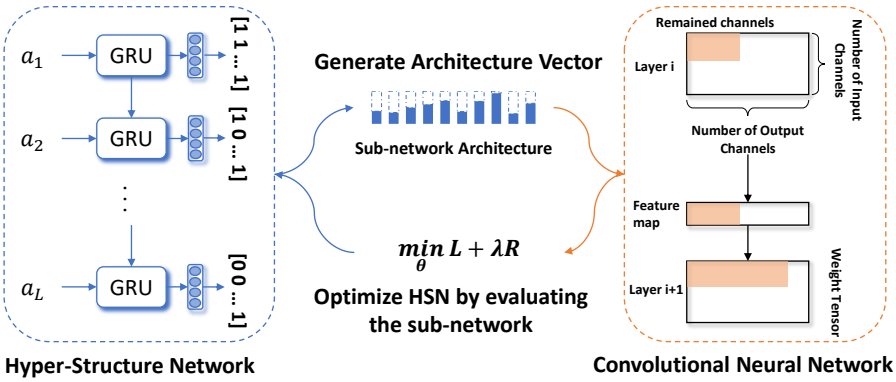


Figure 1: Overview of our proposed method. The width and height dimension of weight tensors are omitted. The architecture vector $\mathbf{v}$ is firstly generated from fixed input a_i, $i = 1, \ldots, L$. Then, a sub-network is sampled according to the architecture vector $\mathbf{v}$. The parameters of HSN are updated by using gradients from the loss function when evaluating the sub-network.

network, which potentially provides both inter-channel and inter-layer information. However, the high variance of policy gradient makes it less efficient and effective compared to our method. In this paper, we focus on channel pruning, since it provides a natural way to reduce computation and memory costs.

Besides weight and channel pruning methods, there are works from other perspectives, including bayesian pruning (Molchanov et al., 2017; Neklyudov et al., 2017), weight quantization (Courbariaux et al., 2015; Rastegari et al., 2016), and knowledge distillation (Hinton et al., 2015).

2.2 Hypernet

Hypernet (Ha et al., 2016) was introduced to generate weights for a network by using a hyper network. Hyper networks have been applied to many machine learning tasks. von Oswald et al. (2020) uses a hyper network to generate weights based on task identity to combat catastrophic forgetting in continual learning. MetaPruning (Liu et al., 2019a) utilizes a hyper network to generate weights when performing evolutionary algorithm. SMASH (Brock et al., 2018) is a neural architecture search method that can predict the weights of a network given its architecture. GPN (Zhang et al., 2019) extends the idea of SMASH and can be used on any directed acyclic graph. Other applications include Bayesian neural networks (Krueger et al., 2017), multi-task learning (Pan et al., 2018), generative models (Suarez, 2017) and so on. Different from original hyper network, the proposed hyper-structure network aims to generate the architecture of a sub-network.

3 Proposed Method

3.1 Notations

To better describe our proposed approach, necessary notations are introduced first. In a CNN, the feature map of ith layer can be represented by $\mathcal{F}_i \in \Re^{C_i \times W_i \times H_i}$, $i = 1, \ldots, L$, where C_i is the number of channels, W_i and H_i are height and width of the current feature map, L is the number of layers. The mini-batch dimension of feature maps is ignored to simplify notations. sigmoid$(\cdot)$ is the sigmoid function. round$(\cdot)$ rounds inputs to nearest integers.

3.2 Hyper-Structure Network

In the context of channel pruning, we need to decide whether a channel should be pruned or not. We can use 0 or 1 to depict the removal or keep of a channel. Consequently, the architecture of a sub-network can be represented as a concatenated vector (containing 0 or 1) from all layers. Our goal is then to use a neural network to generate this vector to represent the corresponding sub-network.

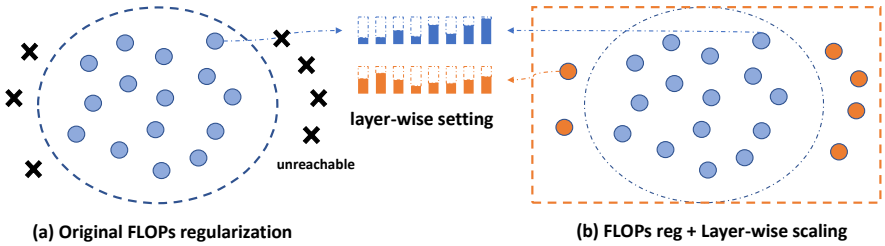


Figure 2: **(a)** For the original FLOPs regularization, some architectures may become unreachable. **(b)** After layer-wise scaling, the potential search space of architectures for a sub-network is increased.

For ith layer, the following output vector is generated:

$$o_i = \text{HSN}(a_i; \Theta), \tag{1}$$

where HSN is our proposed hyper-structure network composed of gated recurrent unit (GRU) (Cho et al., 2014) and dense layers, a_i is a fixed random vector generated from a uniform distribution $\mathcal{U}(0, 1)$, and Θ is the parameter of HSN. The detailed setup of HSN can be found in Appendix C. In short, GRU is used to capture sequential relationships between layers, and dense layers are capable of capturing inter-channel relationships. Note that a_i is a constant vector during training, if a_i is randomly sampled, it will make learning more difficult and result in sub-optimal performance.

Now we have the output o_i, we need to convert it to a 0-1 vector to evaluate the sub-network. The binarization process can be demonstrated by the following equations:

$$\begin{aligned} z_i &= \text{sigmoid}((o_i + g)/\tau), \\ v_i &= \text{round}(z_i), \text{ and } v_i \in \{0, 1\}^{C_i}, \end{aligned} \tag{2}$$

where g follows Gumbel distribution: $g \sim \text{Gumbel}(0, 1)$, v_i is the architecture vector of ith layer, and τ is the temperature hyper-parameter. This process can be summarized as using Gumbel-Softmax distribution (Jang et al., 2016) to approximate Bernoulli distribution. Since the round operation is not differentiable, we use straight through estimator (STE) (Bengio et al., 2013) to enable gradient calculation: $\frac{\partial \mathcal{J}}{\partial z_i} = \frac{\partial \mathcal{J}}{\partial v_i}$. This process can also be viewed as mapping from constant vectors $\{a_i\}_{i=1}^{L}$ to the architecture of a sub-network.

When we evaluate a sub-network, the feature map of ith layer is modified as follows:

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

where $\odot$ is element-wise multiplication, $\hat{v}_i$ is the expanded version of v_i, and $\hat{v}_i$ has the same size of $\mathcal{F}_i$. The feature map $\mathcal{F}_i$ is from the output of Conv-Bn-Relu block. The overall loss function is:

$$\min_{\Theta} \mathcal{J}(\Theta) := \mathcal{L}\big(f(x; \mathcal{W}, \mathbf{v}), y\big) + \lambda \mathcal{R}(T(\mathbf{v}), pT_{\text{total}}) \tag{4}$$

where $\mathbf{v} = (v_1, \ldots, v_L)$, $T(\mathbf{v})$ is the current FLOPs decided by the architecture vector $\mathbf{v}$, T_{total} is the total FLOPs of the original model, $p \in (0, 1]$ is a predefined parameter deciding the remaining fraction of FLOPs, λ is the hyper-parameter controlling the strength of FLOPs regularization, $f(x; \mathcal{W}, \mathbf{v})$ is the CNN parameterized by $\mathcal{W}$ and the sub-network structure is determined by architecture vector $\mathbf{v}$, $\mathcal{L}$ is the cross entropy loss function and $\mathcal{R}$ is the regularization term for FLOPs, and Θ again is the parameters of HSN. The regularization term $\mathcal{R}$ used in this paper is $\mathcal{R}(T(\mathbf{v}), pT_{\text{total}}) = \log(|T(\mathbf{v}) - pT_{\text{total}}| + 1)$.

3.3 Layer-Wise Scaling

The FLOPs regularization considered in Eq. 4 will heavily penalize layers with a larger amount of FLOPs (early layers for most architectures). Consequently, the resulting architecture from the original FLOPs regularization will have a larger pruning rate at early layers. The alternative architectures with similar FLOPs could be omitted. This phenomenon is also demonstrated in Fig. 2. Further analysis is provided in Appendix B.

Algorithm 1: Model Compression via Hyper-Structure Network

Input: dataset for training HSN: D_{HSN}; perversed rate of FLOPs: p; hyper-parameter: λ; training epochs: n_E; pre-trained CNN: f; learning rate β when updating $\{\alpha_i\}_{i=1}^{L}$.
Initialization: initialize Θ randomly; initialize $\alpha_i = 1$, $i = 1, \ldots, L$; freeze $\mathcal{W}$ in f.
for $e := 1$ *to* n_E **do**
  shuffle(D_{HSN})
  for *a batch* (x, y) *in* D_{HSN} **do**
    1. produce architecture vector $\mathbf{v}$ from HSN (Eq. 1 and 2)
    2. calculate gradients w.r.t Θ (Eq. 5).
    3. calculate hyper-gradient for α_i (Eq. 6).
    4. update layer-wise scaling factor $\alpha_i^t = \alpha_i^{t-1} - \beta \frac{\partial J(u(\theta_i^{t-2}, \alpha_i^{t-1}))}{\partial \alpha_i}$, $i = 1, \ldots, L$.
    5. update Θ by ADAM optimizer.
  end
end
return HSN with the final Θ.

To alleviate the problem caused by original FLOPs regularization, we introduce layer-wise scaling factors to dynamically balance gradients from the regularization term $\mathcal{R}$ and the loss term $\mathcal{L}$. Only gradients in dense layers are balanced since GRU is shared by all layers. The gradients w.r.t the parameters of ith dense layer can be written in the following equation:

$$\frac{\partial \mathcal{J}}{\partial \theta_i} = \alpha_i \frac{\partial \mathcal{L}}{\partial \theta_i} + \lambda \frac{\partial \mathcal{R}}{\partial \theta_i}, \tag{5}$$

where θ_i is the parameter of ith dense layer, α_i is the layer-wise scaling factor for ith layer. If no layer-wise scaling is applied, $\alpha_i = 1$. α_i can be regarded as a balancing factor between $\frac{\partial \mathcal{R}}{\partial \theta_i}$ and $\frac{\partial \mathcal{L}}{\partial \theta_i}$. α_i only appears in gradient calculation, as a result, it can not be directly optimized. To optimize α_i, we follow similar deriving process from (Baydin et al., 2018). We first define the update rule $\theta_i^t = u(\theta_i^{t-1}, \alpha_i^t)$ and it can be applied to any optimization algorithms. For example, under stochastic gradient descent, $u(\theta_i^{t-1}, \alpha_i^t) = \theta_i^{t-1} - \eta(\alpha_i^t \frac{\partial \mathcal{L}}{\partial \theta_i^{t-1}} + \lambda \frac{\partial \mathcal{R}}{\partial \theta_i^{t-1}})$. Ideally, our goal is to update α_i so that the corresponding architecture can obtain lower loss value with loss function $\mathcal{J}$. To do so, we want to $\min_{\alpha_i} J(u(\theta_i^{t-1}, \alpha_i^t))$ before update θ_i. For simplicity, the expectation is omitted. The hyper-gradient with respect to α_i can be calculated by:

$$\frac{\partial J(u(\theta_i^{t-1}, \alpha_i^t))}{\partial \alpha_i} = (\frac{\partial \mathcal{J}}{\partial u})^T \frac{\partial u(\theta_i^{t-1}, \alpha_i^t)}{\partial \alpha_i} = (\frac{\partial \mathcal{J}}{\partial \theta_i^t})^T \frac{\partial u(\theta_i^{t-1}, \alpha_i^t)}{\partial \alpha_i}. \tag{6}$$

Given the hyper-gradient of α_i, it can be updated by regular gradient descent method. In experiments, the update rule is ADAM optimizer (Kingma & Ba, 2014), and the detail derivation of $\frac{\partial J(u(\theta_i^{t-1}, \alpha_i^t))}{\partial \alpha_i}$ for ADAM optimizer is described in Appendix E.

3.4 Model Compression via Hyper-Structure Network

The overall algorithm of model compression via hyper-structure network is shown in Alg. 1. As shown in Alg. 1, our method can prune any pre-trained CNNs without modifications. It should be emphasized again that the gradient of GRU is not affected by α_i, which is simply $\frac{\partial \mathcal{L}}{\partial \theta_{\text{GRU}}} + \lambda \frac{\partial \mathcal{R}}{\partial \theta_{\text{GRU}}}$, and θ_{GRU} is the parameter for GRU. Moreover, HSN does not need a whole dataset for training, and a small fraction of the dataset is enough, which makes the training of HSN quite efficient. After the training of HSN, we then use HSN to generate an architecture vector $\mathbf{v}$, and prune the model according to this vector. Also, note that there is certain randomness (Eq. 2 approximates Bernoulli distribution) when generating $\mathbf{v}$, but we find that there is no need to generate the vector multiple times, and average them or conduct majority vote. When generating the vector multiple times, most parts of vectors are the same, the different parts are trivial and do not have impacts on the final performance.

Method	Architecture	Baseline Acc	Pruned Acc	Δ-Acc	$\downarrow$ FLOPs
Channel Pruning (He et al., 2017)	ResNet-56	92.80%	91.80%	-1.00%	50.0%
AMC (He et al., 2018b)		92.80%	91.90%	-0.90%	50.0%
Pruning Filters (Li et al., 2017)		93.04%	93.06%	+0.02%	27.6%
Soft Prunings (He et al., 2018a)		93.59%	93.35%	-0.24%	52.6%
DCP (Zhuang et al., 2018)		93.80%	93.59%	-0.31%	50.0%
DCP-Adapt (Zhuang et al., 2018)		93.80%	93.81%	+0.01%	47.0%
CCP (Peng et al., 2019)		93.50%	93.42%	-0.08%	**52.6%**
MCH(ours)		92.99%	93.23%	**+0.24%**	50.0%
WM (Zhuang et al., 2018)	MobileNetV2	94.47%	94.17%	-0.30%	26.0%
DCP (Zhuang et al., 2018)		94.47%	94.69%	+0.22%	26.0%
MCH(ours)		94.23%	94.68%	**+0.38%**	**40.0%**

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

4 Experimental Results

4.1 Implementation Details

Similar to many model compression works, CIFAR-10 (Krizhevsky & Hinton, 2009) and ImageNet (Deng et al., 2009) are used to evaluate the performance of our method. Our method requires one hyper-parameter p to control the FLOPs budget. The detailed choices of p are listed in Appendix F.

For CIFAR-10, we compare with other methods on ResNet-56 and MobileNetV2. For ImageNet, we select ResNet-34, ResNet-50, ResNet-101 and MobileNetV2 as our target models. The reason we choose these models is because that ResNet (He et al., 2016) and MobileNetV2 (Sandler et al., 2018) are much harder to prune than earlier models like AlexNet (Krizhevsky et al., 2012) and VGG (Simonyan & Zisserman, 2014b). λ decides the regularization strength in our method. We choose $\lambda = 4$ in all CIFAR-10 experiments and $\lambda = 8$ for all ImageNet experiments.

For CIFAR-10 models, we train ResNet-56 from scratch following the pytorch examples. After pruning, we finetune the model for 160 epochs using SGD with a start learning rate 0.1, weight decay 0.0001 and momentum 0.8, the learning rate is multiplied by 0.1 at epoch 80 and 120. For ImageNet models, we directly use the pre-trained models released from pytorch (Paszke et al., 2017; 2019). After pruning, we finetune the model for 100 epochs using SGD with a start learning rate 0.01, weight decay 0.0001 and momentum 0.9, and the learning rate is scaled by 0.1 at epoch 30, 60 and 90. For MobileNetV2 on ImageNet, we choose weight decay as 0.00004 which is the same with the original paper (Sandler et al., 2018).

For the training process of HSN, we use ADAM (Kingma & Ba, 2014) optimizer with a constant learning rate 0.001 and train HSN for 200 epochs. τ in Eq. 2 is set as 0.4. The β for α_i is chosen as 0.01, and α_i is updated as shown in Alg. 1. To build dataset D_{HSN}, we random sample $2,500$ and $10,000$ samples for for CIFAR-10 and ImageNet separately. In the experiments, we found that a stand-alone validation set is not necessary, all samples in D_{HSN} come from the original training set. All codes in this paper are implemented with pytorch (Paszke et al., 2017; 2019). The experiments are conducted on a machine with 4 Nvidia Tesla P40 GPUs.

4.2 CIFAR-10 Results

In Tab. 1, we present the comparison results on CIFAR-10 dataset. Our method is abbreviated as MCH (**M**odel **C**ompression via **H**yper Structure Network) in the experiment section. For ResNet-56, our method can prune 50% of FLOPs while obtain 0.24% performance gain in accuracy. On MobileNetV2, our method can obtain 0.38% gain in accuracy. Compared to all other methods, our method can achieve the best results. Our method can outperform the second best method (DCP-Adapt) by 0.23% on ResNet-56. On MobileNetV2, our method can outperform the the second best method by 0.16% while pruning 14% more FLOPs. For both models, our method performs much better than early methods (He et al., 2017; 2018b; Li et al., 2017; He et al., 2018a). Our method can outperform CCP by 0.32% in terms of Δ-Acc, which demonstrate that learning both inter-layer and inter-channel relationships are better than only considering inter-channel relationships.

Method	Architecture	Pruned Top-1	Pruned Top-5	Δ Top-1	Δ Top-5	$\downarrow$ FLOPs
Pruning Filters (Li et al., 2017)	ResNet-34	72.17%	-	-1.06%	-	24.8%
Soft Prunings (He et al., 2018a)		71.84%	89.70%	-2.09%	-1.92%	41.1%
IE (Molchanov et al., 2019)		72.83%	-	-0.48%	-	24.2%
FPGM (He et al., 2019)		72.63%	91.08%	-1.29%	-0.54%	41.1%
MCH(ours)		**72.85%**	**91.15%**	**-0.45%**	**-0.27%**	**44.0%**
IE (Molchanov et al., 2019)	ResNet-50	74.50%	-	-1.68%	-	45.0%
FPGM (He et al., 2019)		74.83%	92.32%	-1.32%	-0.55%	53.5%
GAL (Lin et al., 2019)		71.80%	90.82%	-4.35%	-2.05%	55.0%
DCP (Zhuang et al., 2018)		74.95%	92.32%	-1.06%	-0.61%	55.6%
CCP (Peng et al., 2019)		75.21%	92.42%	-0.94%	-0.45%	54.1%
MetaPruning (Liu et al., 2019a)		75.40%	-	-1.20%	-	51.2%
GBN (You et al., 2019)		75.18%	92.41%	-0.67%	-0.26%	55.1%
HRank (Lin et al., 2020)		74.98%	92.33%	-1.17%	-0.54%	43.8%
Hinge (Li et al., 2020)		74.70%	-	-1.40%	-	54.4%
MCH(ours)		**75.60%**	**92.67%**	**-0.55%**	**-0.20%**	**56.0%**
Rethinking (Ye et al., 2018)	ResNet-101	77.37%	-	-2.10%	-	47.0%
IE (Molchanov et al., 2019)		77.35%	-	-0.02%	-	39.8%
FPGM (He et al., 2019)		77.32%	93.56%	-0.05%	0.00%	41.1%
MCH(ours)		**77.58%**	**93.81%**	**+0.21%**	**+0.25%**	**56.0%**
MobileNetV2 0.75 (Sandler et al., 2018)	MobileNetV2	69.80%	89.60%	-2.00%	-1.40%	30.0%
AMC (He et al., 2018b)		70.80%	-	-1.00%	-	30.0%
MetaPruning (Liu et al., 2019a)		71.20%	-	-0.80%	-	**30.7%**
MCH(ours)		**71.54%**	**90.08%**	**-0.58%**	**-0.33%**	30.1%

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

4.3 IMAGENET RESULTS

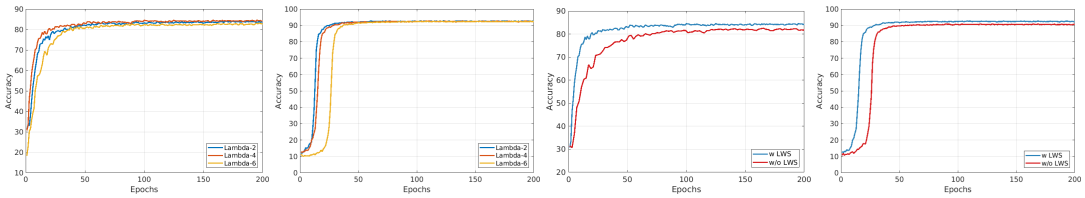
In Tab. 2, the results on ImageNet are presented, Top-1/Top-5 accuracy after pruning are presented. Most of our comparison methods comes from recently published papers including IE (Molchanov et al., 2019), FPGM (He et al., 2019), GAL (Lin et al., 2019), CCP (Peng et al., 2019), MetaPruning (Liu et al., 2019a), GBN (You et al., 2019), Hinge (Li et al., 2020), abd HRank (Lin et al., 2020).

ResNet-34. Our proposed MCH can prune 44.0% of FLOPs with 0.45% and 0.27% performance loss on Top-1 and Top-5 accuracy. Such a result is better than any other method. Proposed MCH performs similarly compared to IE (Molchanov et al., 2019) in Top-1 and Δ Top-1 accuracy ($72.85\%/-0.45\%$ vs. $72.83\%/-0.43\%$), while our method can prune almost 20% more FLOPs. Given similar FLOPs pruning rate, our method achieves better results compared to FPGM (He et al., 2019) (-0.45%/-0.27% vs. -1.29%/-0.54% for Δ Top-1/Δ Top-5). Besides IE and FPGM, The margin between our method and rest methods are even larger.

ResNet-50. ResNet-50 is a very popular model for evaluating model compression methods. With such intense competition, our method can still achieve the best Top-1/Top-5 and Δ Top-1/Δ Top-5 results. The second best method in terms of Top-1 accuracy is MetaPruning (Liu et al., 2019a), which can achieve 75.40% Top-1 result after pruning. Our method outperforms MetaPruning by 0.20% in Top-1 accuracy while our method can prune 5% more FLOPs. MetaPruning utilizes hypernet to generate weights when evaluating sub-networks, however, such a design paradigm prohibits MetaPruning to be directly used on pre-trained models. The weights inherited from the pre-trained model might be one of the reasons why our method can outperform MetaPruning. GBN (You et al., 2019) obtains the second-best Δ Top-1 accuracy, however, the accuracy after pruning is quite low compared to other methods. Our method can outperform GBN by 0.42% in Top-1 accuracy. Besides GBN and MetaPruning, our method can outperform two recent methods HRank (Lin et al., 2020) and Hinge (Li et al., 2020) by 0.62% to 0.90% on Top-1 accuracy.

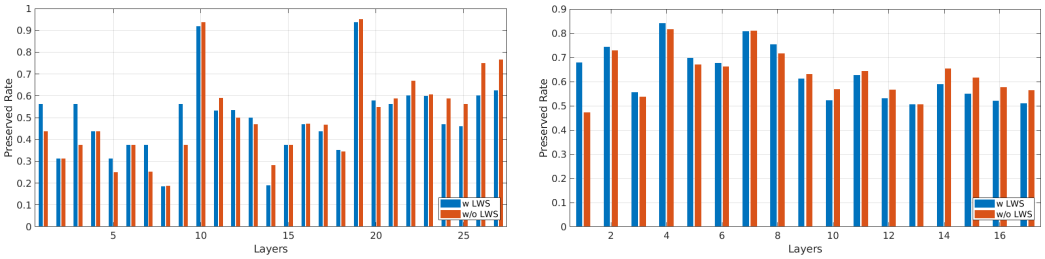
ResNet-101. For ResNet-101, our method can increase the performance of the baseline model by 0.21% and 0.25% on Top-1 and Top-5 accuracy, while removes 56% of FLOPs. The second best method FPGM (He et al., 2019) can maintain the performance and reducing 41% of FLOPs. In short, compared to FPGM, our method can obtain performance gain while pruning 15% more FLOPs.

MobileNetV2. On MobileNetV2, we mainly compare with AMC (He et al., 2018b) and MetaPruning (Liu et al., 2019a). Both of them can be regarded as representative works for AutoML related model compression methods (AMC uses reinforcement learning; MetaPruning uses evolutionary



(a) λ, ResNet (b) λ, MobileNetV2 (c) LWS, ResNet (d) LWS, MobileNetV2

Figure 3: (a,b): Effect of λ on the performance of sub-networks. (c,d): Effect of layer-wise scaling on the performance of sub-network. All experiments are done on CIFAR-10.



(a) ResNet-56 (b) MobileNetV2

Figure 4: Layer-wise preserved rate with or without layer-wise scaling (LWS) for ResNet-56 and MobileNetV2 on CIFAR-10.

algorithm and hypernet). Our method can achieve 71.54% Top-1 accuracy while pruning around 30% of FLOPs, which is 0.34% and 0.74% higher than MetaPruning and AMC. These results show that our method can outperform AutoML based methods.

In summary, our method can outperform these comparison methods and achieve the state-of-the-art performance. These experimental results also indicate that inter-channel and inter-layer relationships should be considered when designing model compression methods.

4.4 Effects of Layer-wise Scaling

We further study the impact of λ and layer-wise scaling (LWS) when training HSN on CIFAR-10. In Fig. 3 (a,b), we can see that changing λ does not have a large impact on the final performance of a sub-network, and our method is not sensitive to it. In Fig. 4 (c,d), it shows that using LWS can improve the final performance of a sub-network and obtain lower loss. Moreover, early layers usually have a larger preserved rate with LWS as shown in Fig 4, indicating that alternative sub-network architectures can be discovered from LWS. Without LWS, the final performance of ResNet-56 will decrease 0.19%, achieves 93.04% final accuracy on CIFAR-10. Similar observations hold for MobileNetV2 (94.45% final accuracy and the relative gap is 0.16%). These observations show that LWS indeed helps the training of the HSN.

5 Conclusion

In this paper, we proposed a hyper-structure network for model compression to capture inter-channel and inter-layer relationships. An architecture vector can be generated from HSN to select a sub-network from the original model. At the same time, we evaluated this sub-network by using classification and resource losses. The HSN can be updated by the gradients from them. Moreover, we also identified the problem of FLOPs constraint (bias towards latter layers), which limits the final search space of HSN. To solve it, we further proposed layer-wise scaling to balance the gradients. With the aforementioned novel techniques, our method can achieve state-of-the-arts performance on ImageNet with four different architectures.

REFERENCES

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. URL https://openreview.net/forum?id=BkrsAzWAb.

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

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.

Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rydeCEhs-.

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

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

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*, pp. 248–255. Ieee, 2009.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

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

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*, pp. 1135–1143, 2015b.

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*, pp. 770–778, 2016.

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)*, pp. 2234–2240, 2018a.

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*, pp. 4340–4349, 2019.

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*, pp. 1389–1397, 2017.

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)*, pp. 784–800, 2018b.

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

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

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

Sergey Ioffe and Christian Szegedy. 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*, ICML, pp. 448–456. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045167.

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

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

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

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

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

David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

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

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

Yawei Li, Shuhang Gu, Christoph Mayer, Luc Van Gool, and Radu Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8018–8027, 2020.

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.

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

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*, pp. 3296–3305, 2019a.

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

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=rJlnB3C5Ym.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1Y8hhg0b.

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

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*, pp. 11264–11272, 2019.

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

Zheyi Pan, Yuxuan Liang, Junbo Zhang, Xiuwen Yi, Yong Yu, and Yu Zheng. Hyperst-net: Hypernetworks for spatio-temporal forecasting. *arXiv preprint arXiv:1809.10889*, 2018.

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

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*, pp. 8024–8035, 2019.

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

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

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*, pp. 779–788, 2016.

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*, pp. 91–99, 2015.

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*, pp. 4510–4520, 2018.

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

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

Joseph Suarez. Language modeling with recurrent highway hypernetworks. In *Advances in neural information processing systems*, pp. 3267–3276, 2017.

Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJgwNerKvB`.

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

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

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

Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rkgW0oA9FX`.

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

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*, pp. 875–886, 2018.