 Dependency Aware Filter Pruning

Kai Zhao, Xin-Yu Zhang, Qi Han, and Ming-Ming Cheng

Abstract—Convolutional neural networks (CNNs) are typically over-parameterized, bringing considerable computational overhead and memory footprint in inference. Pruning a proportion of unimportant filters is an efficient way to mitigate the inference cost. For this purpose, identifying unimportant convolutional filters is the key to effective filter pruning. Previous work prunes filters according to either their weight norms or the corresponding batch-norm scaling factors, while neglecting the sequential dependency between adjacent layers. In this paper, we further develop the norm-based importance estimation by taking the dependency between the adjacent layers into consideration. Besides, we propose a novel mechanism to dynamically control the sparsity-inducing regularization so as to achieve the desired sparsity. In this way, we can identify unimportant filters and search for the optimal network architecture within certain resource budgets in a more principled manner. Comprehensive experimental results demonstrate the proposed method performs favorably against the existing strong baseline on the CIFAR, SVHN, and ImageNet datasets. The training sources will be publicly available after the review process.

Index Terms—Deep Learning, Network Compression, Filter Pruning.

I. INTRODUCTION

Convolutional neural networks (CNNs) have achieved remarkable performance on a wide range of vision and learning tasks [1]–[11]. Despite the impressive performance, CNNs are notably over-parameterized and thus lead to high computational overhead and memory footprint in inference. Therefore, network compression techniques are developed to assist the deployment of CNNs in real-world applications.

Filter pruning is an efficient way to reduce the computational cost of CNNs with negligible performance degradation. As shown in Fig. 1, a typical pipeline of filter pruning [12] works as follows: 1) train an over-parameterized model with the sparsity-inducing regularization; 2) estimate the importance of each filter and prune the unimportant filters; 3) finetune the compressed model to recover the accuracy. Among these, identifying unimportant filters is the key to efficient filter pruning. Prior work [12]–[15] prunes filters according to the magnitude of the corresponding model parameters. For example, Li et al. [14] prune convolutional filters of smaller $L_1$ norms as they are considered to have less impact on the functionality of the network. Network Slimming [12] then proposes to prune channels (i.e., filters) based on the corresponding scaling factors. To be specific, the scaling factors of the batch normalization (BN) [16] layer serve as an indicator of the channel importance, on which an $L_1$ regularization is imposed to promote sparsity. As a result, Liu et al. [12] derive an automatically searched network architecture of the compressed model.

However, existing methods select unimportant filters based only on the parameter magnitude of a single layer [12]–[14], [17]–[19], while neglecting the dependency between consecutive layers. For example, a specific channel with a small BN scaling factor may be followed by a convolution with a large weight magnitude at that channel, making the channel still important to the output. Besides, in the “smaller BN factor, less importance” strategy, BN factors from different layers are gathered together to rank and determine the filters to be pruned. We argue and empirically verify that this strategy is sub-optimal and may lead to unstable network architectures as it neglects the intrinsic statistical variation among the BN factors of different layers. Empirically, we observe that the pruned architectures of Network Slimming [12] are sometimes unbalanced and lead to severely degraded performance, especially when the pruning ratio is relatively high.

In this paper, we propose a dependency-aware filter pruning strategy, which takes the relationship between adjacent layers into consideration. Hence, we measure the filter importance in a more principled manner. Along this line, we introduce a novel criteria to determine the filters to be pruned by the local importance of the consecutive two layers. That is, if one layer is sparse, then more filters will be pruned and vice versa, regardless of the statistics of other layers. Finally, we propose an automatic-regularization-control mechanism in which the coefficient of the sparsity-inducing regularization is dynamically adjusted to meet the desired sparsity. Our contributions are summarized below:

- We propose a principled criteria of measuring the filter importance by taking the dependency between adjacent layers into consideration.
- Given the dependency-aware filter importance, we prune filters based on the local statistics of each layer, instead of ranking the filter importance across the entire network.
- We propose to dynamically control the coefficient of

Fig. 1. A typical pipeline of filter pruning: (a) train a over-parameterized model with sparsity-inducing regularization; (b) prune unimportant filters based on certain criteria; (c) finetune the compressed model till convergence.
the sparsity-inducing regularization to achieve the desired model sparsity. Comprehensive experimental results demonstrate that the improved filter pruning strategy performs favorably against the existing strong baseline [12] on the CIFAR, SVHN, and ImageNet datasets. We also validate our design choices with several ablation studies and verify that the proposed algorithm reaches more stable and well-performing architectures.

II. RELATED WORK

A. Network pruning

Network pruning is a prevalent technique to reduce redundancy in deep neural networks by removing unimportant neurons. Specifically, weight pruning approaches [20]–[26] remove network parameters without structural constraints, thus leading to unstructured architectures that are not well supported by the BLAS libraries. On the other hand, filter pruning methods [12], [14], [27]–[30] remove the entire filters (i.e., channels) from each layer, thus resulting in compact networks that can be conveniently incorporated into modern BLAS libraries. According to how to identify the unimportant filters, existing filter pruning methods can be further divided into two categories: data-dependent filter pruning and data-independent filter pruning.

Data-dependent filter pruning utilizes the training data to determine the filters to be pruned. Polyak et al. [31] remove filters that produce activations of smaller norms. He et al. [28] perform a channel selection by minimizing the reconstruction error. Zheng et al. [32] and Anwar et al. [33] both evaluate the filter importance via the loss of the validation accuracy without each filter. Molchanov et al. [29] approximate the exact contribution of each filter with the Taylor expansion. A recent work [34] proposes a layer-wise recursive Bayesian pruning method with a dropout-based metric of redundancy.

Data-independent filter pruning identifies less important filters based merely on the model itself (i.e., model structure and model parameters). Li et al. [14] discard filters according to the $L_1$ norm of the corresponding parameters as filters with smaller weights are considered to contribute less to the output. Network Slimming [12] imposes a sparsity-inducing regularization on the scaling factors of the BN layer and then prunes filters with smaller scaling factors. Zhou [35] using the evolutionary algorithm to search redundant filters during training. He et al. [18] propose to dynamically prune filters during training. In another work He et al. [27] propose to prune filters that are close to the geometric median. They argue that filters near the geometric median are more likely to be represented by others [36], thus leading to redundancy.

Our method belongs to the data-independent filter pruning, which is generally more efficient as involving the training data brings extra computation. For example, Zheng et al. [32] and Anwar et al. [33] measure the importance of each filter by removing the filter and re-evaluating the compressed model on the validation set. This procedure is extremely time-consuming. Essentially, we take the dependency between the consecutive layers into consideration, while previous data-independent methods [12], [14], [27] merely focus on the parameters (either the convolutional weights [14], [27] or the BN scaling factors [12]) of a single layer. Besides, we propose a novel mechanism to dynamically control the coefficient of the sparsity-inducing regularization, instead of pre-defining it based on human heuristics [12]. Incorporating these components, our principled approaches and better estimate the filter importance (Sec. V-A) and achieve more balanced pruned architectures (Sec. V-D).

B. Neural Architecture Search

While most state-of-the-art CNNs [37]–[39] manually designed by human experts, there is also a line of research that explores automatic network architecture learning [40]–[46], called neural architecture search (NAS). Specifically, automatically tuning channel width is also studied in NAS. For example, ChamNet [41] builds an accuracy predictor on the Gaussian Process with the Bayesian optimization to predict the network accuracy with various channel widths in each layer. FBNet [45] adopts a gradient-based method to optimize the CNN architecture and search for the optimal channel width. The proposed pruning method can be regarded as a particular case of channel width selection as well, except that we impose the resource constraints on the selected architecture. However, our method learns the architecture through a single training process, while typical NAS methods may train hundreds of models with different architectures to determine the best-performing one [41], [46]. We highlight that our efficiency is in line with the goal of neural architecture search.

C. Other Alternatives for Network Compression

a) Low-Rank Decomposition: There is a line of research [47]–[50] that aims to approximate the weight matrices of the neural networks with several low-rank matrices using techniques like the Single Value Decomposition (SVD) [47]. However, these methods cannot be applied to the convolutional weights, and thus the acceleration in inference is limited.

b) Weight Quantization: Weight quantization [51]–[55] reduces the model size by using a low bit-width number of the weights and hidden activations. For example, Courbariaux et al. [52] and Rastegari et al. [53] quantize the real-valued weights into binary or ternary ones, i.e., the weight values are restricted to $\{-1, 1\}$ or $\{-1, 0, 1\}$. Cheng et al. [55] quantize CNNs with a predefined codebook. Despite the significant model-size reduction and inference acceleration, these methods often come with a mild accuracy drop due to the low precision.

III. DEPENDENCY-AWARE FILTER PRUNING

A. Dependency Analysis

Generally, we assume a typical CNN involves multiple convolution operators (Conv layers), batch normalizations (BN layers) [16], and non-linearities, which are applied to the input signals sequentially as in Fig. 2. Practically, each channel is transformed independently in the BN layers and non-linearities, while inter-channel information is fused in the Conv layers. To prune filters (i.e., channels) with minimal impact on
the network output, we analyze the role each channel plays in the Conv layers as follows.

Let \( \mathbf{X}^l \in \mathbb{R}^{C_l \times H^l \times W^l} \) be the hidden activations after normalization before scaling in the \( l^{th} \) BN layer. The scaled activations \( \mathbf{Y}^l \) can be formulated as:

\[
\mathbf{Y}^l = \gamma^l_{c} \mathbf{X}^l_c,
\]

where \( \gamma^l_{c} \in \mathbb{R}^{c} \) denotes the scaling factor of the \( l^{th} \) BN layer and \( \mathbf{X}^l_c \) (resp. \( \mathbf{Y}^l_c \)) is the \( c^{th} \) channel of \( \mathbf{X}^l \) (resp. \( \mathbf{Y}^l \)). Then, a Lipschitz-continuous non-linearity \( \sigma \) is applied to \( \mathbf{Y}^l \), namely,

\[
\mathbf{Z}^l = \sigma(\mathbf{Y}^l).
\]

Afterward, all channels of \( \mathbf{Z}^l \) are fused into \( \mathbf{F}^{l+1} \in \mathbb{R}^{C_{l+1} \times H^{l+1} \times W^{l+1}} \) via a convolution operation, and different channels contribute to the fused activation \( \mathbf{F}^{l+1} \) differently. Formally, let \( \mathbf{W}^{l+1} \in \mathbb{R}^{C_{l+1} \times C_{l} \times k \times k} \) be the \( (l+1)^{th} \) convolution filter, where \( k \) denotes the kernel size. We have

\[
\mathbf{F}^{l+1} = \mathbf{W}^{l+1} \odot \mathbf{Z}^l,
\]

where \( \odot \) denotes the convolution operator. As convolution is an affine transformation, we re-formulate the linearity of Eq. (3) explicitly:

\[
\mathbf{F}^{l+1} = \mathbf{W}^{l+1} \mathbf{Z}^l,
\]

where \( \mathbf{F}^{l+1} \in \mathbb{R}^{C_{l+1} \times H^{l+1} \times W^{l+1}} \), \( \mathbf{W}^{l+1} \in \mathbb{R}^{C_{l+1} \times C_{l} \times k \times k} \), and \( \mathbf{Z}^l \in \mathbb{R}^{C_{l} \times H^l \times W^l} \) are the unfolded versions of \( \mathbf{F}^{l+1}, \mathbf{W}^{l+1}, \) and \( \mathbf{Z}^l \), respectively. Figure 2 shows a sequential illustration of the “batch normalization, convolution” operation. The input feature \( \mathbf{X}^l \) (after whitening) is stretched by the scaling factors \( \gamma^l_{c} \) and convolved with three convolutional kernels, resulting in a threec-channel output feature. For the dependency-aware criterion, the importance of the \( c^{th} \) feature map is estimated by the product of the absolute value of the scaling factor \( |\gamma^l_{c}| \) and the magnitude of corresponding convolutional kernel \( ||\mathbf{W}^{l+1}_{c}|| \). (See Eq. (7).)

\[
||\mathbf{F}^{l+1}|| \leq \sum_{c=1}^{C_l} ||\mathbf{W}^{l+1}_{c}|| \mathbf{Z}^l_c \leq \sum_{c=1}^{C_l} ||\mathbf{W}^{l+1}_{c}|| \cdot ||\mathbf{Z}^l_c||
\]

\[
\leq \sum_{c=1}^{C_l} ||\mathbf{W}^{l+1}_{c}|| \cdot L \cdot ||\mathbf{Z}^l_c||
\]

\[
= L \sum_{c=1}^{C_l} |\gamma^l_{c}| \cdot ||\mathbf{W}^{l+1}_{c}|| \cdot ||\mathbf{Z}^l_c||,
\]

where \( L \) denotes the Lipschitz constant of function \( \sigma \), and \( \mathbf{X}^l \) and \( \mathbf{Y}^l \) are the unfolded versions of \( \mathbf{X}^l \) and \( \mathbf{Y}^l \), respectively. Since the normalization operation in BN layer uniformizes the activations \( \mathbf{X}^l \) (i.e., \( \mathbf{X}^l \)) across channels, we quantify the contribution of the \( c^{th} \) channel by

\[
S^l_c = |\gamma^l_{c}| \cdot ||\mathbf{W}^{l+1}_{c}||,
\]

which serves as our metric for network pruning.

### B. Filter Selection

Let \( r \in (0, 1) \) be the pruning ratio, and \( C^l \) \( (l \in \{1, 2, \cdots, L\}) \) be the number of filters in the \( l^{th} \) convolutional layer. Generally, previous works can be divided into two groups according to the target network.

- **a) Pruning with Pre-defined Target Network:** Many previous work [17], [18], [27] prune a fixed ratio of filters in each layer. In other words, there will be \( r \cdot C^l \) filters pruned from the \( l^{th} \) layer. The architecture of the target network is known even without pruning. However, recent work [13], [60] reveals that this strategy cannot find the optimal distribution of the neuron numbers of each convolutional layer across the network, as some layers will be over-parameterized while some under-parameterized.

- **b) Pruning as Architecture Search:** Network Slimming [12] treats pruning as a special form of architecture search, i.e., search for the optimal channel width of each layer. It compares the importance of each convolutional filter across the entire network and prunes filters of less importance. This approach provides more flexibility of the compressed architecture as a higher pruning ratio can be achieved if a specific layer is sparse and vice versa.
However, according to our practice, we find that sometimes too many filters of a layer (or occasionally all filters of a layer) are pruned in this strategy, leading to severely degraded performance. This is because it does not take the intrinsic statistical variation among different layers into consideration. Suppose there are two layers and the corresponding scaling factors are \{0.10, 0.01, 0.03, 0.15\} and \{1, 100, 2, 200\}, respectively. Our target is to prune half of the filters, i.e., \( r = 0.5 \). Apparently, the second and third channels should be pruned from the first layer, and the first and third channels should be pruned from the second layer. However, if we rank the scaling factors globally, all filters of the first layer will be pruned, which is obviously unreasonable.

To alleviate this issue, we instead select the unimportant filters based on the intra-layer statistics. Let \( S'_c \) be the importance of the \( c \)-th channel in the \( l \)-th layer. Then, filters with importance factor \( S'_c \leq \max(S') \cdot p \) will be pruned, where the threshold \( p \in (0, 1) \) is a hyper-parameter. Formally, the set of filters to be pruned in the \( l \)-th layer is:

\[
\mathcal{F}_l^{\text{pruned}} = \{ c : S'_c \leq \max(S') \cdot p \}. \tag{8}
\]

In our solution, the choice of the filters to be pruned in one layer is made independent of the statistics of other layers, so that the intrinsic statistical differences among layers will not result in dramatically unbalanced neural architecture.

C. Automatic Control of Sparsity Regularization

Network Slimming [12] imposes an \( L_1 \) regularization on the model parameters to promote model sparsity. However, choosing a proper regularization coefficient \( \lambda \) is non-trivial and mostly requires manual tuning based on human heuristics. For example, Network Slimming performs a grid search in a set of candidate coefficients for each dataset and network architecture. However, different pruning ratios require different levels of model sparsity, and thus different coefficients \( \lambda \). It is extremely inefficient to tune \( \lambda \) for each experimental setting.

To escape from manually choosing \( \lambda \) and meet the required model sparsity at the same time, we propose to automatically control the regularization coefficient \( \lambda \). Following the practice in [12], an \( L_1 \) regularization is imposed on the scaling factors of the batch normalization layers. As shown in Alg. 1, at the end of the \( t \)-th epoch, we calculate the overall sparsity of the model:

\[
P = \frac{\sum_l |\mathcal{F}_l^{\text{pruned}}|}{\sum_l |\mathcal{C}_l|}. \tag{9}
\]

Given the total number of epochs \( N \), we compute the expected sparsity gain, and if the sparsity gain within an epoch does not meet the requirement, i.e., \( P_t - P_{t-1} < (r - P_{t-1})/(N - t + 1) \), the regularization coefficient \( \lambda \) is increased by \( \Delta \lambda \). If the model is over-sparse, i.e., \( P_t > r \), the coefficient \( \lambda \) is decreased by \( \Delta \lambda \). This strategy guarantees that the model meets the desired model sparsity, and that the pruned filters contribute negligibly to the outputs.

IV. EXPERIMENTAL RESULTS

In this section, we first describe the details of our implementation in Sec. IV-A, and report the experimental results on the CIFAR [61] datasets in Sec. IV-B and the ImageNet [62] dataset in Sec. IV-D.

Algorithm 1: Automatic Regularization Control

| Initialization | \( \lambda_1 = 0 \), \( P_1 = 0 \), \( N = \#\text{epochs} \) |
|---------------|--------------------------------------------------|
| for \( t := 1 \) to \( N \) do | \( P_t = \frac{\sum_l |\mathcal{F}_l^{\text{pruned}}|}{\sum_l |\mathcal{C}_l|} \) |
| if \( P_t - P_{t-1} < \frac{r - P_{t-1}}{N - t + 1} \) then | \( \lambda_{t+1} = \lambda_t + \Delta \lambda \) |
| else if \( P_t > r \) then | \( \lambda_{t+1} = \lambda_t - \Delta \lambda \) |
| end |

A. Implementation Details

Our implementation is based on the official training sources of Network Slimming in the PyTorch [63] library.\(^4\) We follow the “train, prune, and finetune” pipeline as depicted in Fig. 1.

a) Datasets and Data Augmentation: We conduct image classification experiments on the CIFAR [61], SVHN [64], and ImageNet [62] datasets. For the CIFAR and SVHN datasets, we follow the common practice of data augmentation: zero-padding of 4 pixels on each side of the image and random crop of a \( 32 \times 32 \) patch. On the ImageNet dataset, we adopt the standard data augmentation strategy: resize images to have the shortest edge of 256 pixels and then randomly crop a \( 224 \times 224 \) patch. Besides, we adopt random horizontal flip on the cropped image for the CIFAR and ImageNet datasets. The input data is normalized by subtracting the channel-wise means and dividing the channel-wise standard deviations before being fed to the network.

b) Backbone Architectures: We evaluate the proposed method on two representative architectures: VGGNet [39] and ResNet [37]. Following the practice of Network Slimming [12], we use the Pre-Act-ResNet architecture [65] in which the BN layers and non-linearities are placed before the convolutional layers. (See Fig. 3.)

c) Hyper-parameters: The threshold in Eq. (8) is set to 0.01 unless otherwise specified, and \( \Delta \lambda = 10^{-5} \) in all experiments. We use the SGD optimizer with a momentum of 0.9 and a weight decay of \( 10^{-4} \). The initial learning rate is 0.1 and divided by a factor of 10 at the specified epochs. We train for 160 epochs on the CIFAR datasets and 40 epochs on the SVHN dataset. The learning rate decays at 50% and 75% of the total training epochs. On the ImageNet dataset, we train for 100 epochs and decay the learning rate every 30 epochs.

d) Half-precision Training on ImageNet: We train models on the ImageNet dataset with half-precision (FP16), using the Apex library,\(^5\) where parameters of batch normalization are represented in FP32 while others in FP16. This allows us to

\( ^4 \) https://github.com/Eric-mingjie/rethinking-network-pruning
\( ^5 \) https://github.com/NVIDIA/apex
train the ResNet-50 model within 40 hours on 4 RTX 2080Ti GPUs. Despite training with FP16, we do not observe obvious performance degradation in our experiments. For example, as shown in Tab. IV, we achieve a top-1 accuracy of 76.27% with the Pre-ResNet-50 architecture on the ImageNet dataset, which is very close to that in the original paper [65] or reported in [29].

e) **Train, Prune, and Finetune:** We adopt the three-stage pipeline, i.e., train, prune, and finetune, as in many previous pruning methods [12], [17], [18], [27], [30], [66]. (See Fig. 1.) In the experiments, we found that in the first stage, the model sparsity $P$ grows rapidly when the learning rate is large. After the learning rate decays, the model sparsity hardly increases unless an extremely large $\lambda$ is reached. Therefore, to effectively promote model sparsity, we keep the learning rate fixed in the first stage, and decays the learning rate normally when in the third stage. On CIFAR datasets, we train for 160 epochs for the first stage, and on the ImageNet dataset, we train only 40 epochs for the first stage. On both CIFAR and ImageNet datasets, we finetune for a full episode.

f) **Prune with Short Connections:** In the Pre-Act-ResNet architecture, operators are arranged in the “BN, ReLU, and Conv” order. As depicted in Fig. 3, given the input feature maps, we perform a “feature selection” right after the first batch-norm layer, and prune only the input dimension of the last convolutional layer. Consequently, the number of channels is unchanged in the residual path.

**B. Results on CIFAR**

We first evaluate our method on the CIFAR10 and CIFAR100 datasets. Experiments on the CIFAR datasets are conducted using the VGGNets and ResNets with various depths. On the CIFAR datasets, we record the mean and
### Table II

Experimental results on the CIFAR100 dataset. Here, “N/A” indicates the compressed model collapses in all runs. Still, our approach consistently outperforms the Network Slimming (SLM) [12] baseline. Notably, our approach outperforms Network Slimming by up to 2% on the ResNet-164 backbone.

| Model     | Methods            | ratio r | Baseline accuracy | Finetune accuracy |
|-----------|--------------------|---------|-------------------|-------------------|
| VGG11     | SLM [12]           | 0.3     | 69.33 (±0.26)     | 66.54 (±0.14)     |
|           | Ours               |         | 68.24 (±0.11)     | 67.84 (±0.11)     |
| VGG16     | SLM                | 0.3     | 73.50 (±0.18)     | 73.36 (±0.28)     |
|           | Ours               |         | 72.16 (±0.23)     | 73.59 (±0.37)     |
| VGG19     | SLM (from [67])    | 0.5     | 72.63 (±0.21)     | 72.32 (±0.28)     |
|           | Ours               |         | 71.19 (±0.54)     | 72.48 (±0.28)     |
| Res164    | SLM                | 0.4     | 76.80 (±0.19)     | 76.22 (±0.20)     |
|           | Ours               |         | 76.43 (±0.26)     | 77.74 (±0.17)     |
|           | SLM                | 0.6     | 76.80 (±0.19)     | 74.17 (±0.33)     |
|           | Ours               |         | 76.43 (±0.26)     | 76.28 (±0.27)     |

Fig. 4. Performance (mean and standard deviation over a 10-fold validation) of pruning the ResNet-56 network on the CIFAR10 dataset under various pruning ratios r.

standard deviation over a 10-fold validation. It is worthy of noting that, as described in Sec. III-B, Network Slimming [12] often results in unstable architectures, whose performance is greatly degraded. (See Sec. V-D for details.) Therefore, for Network Slimming, we skip the outliers and restart the pipeline if the accuracy is 10% lower than the mean accuracy. Quantitative results on CIFAR10 and CIFAR100 datasets are summarized in Tab. I and Tab. II, respectively. Additionally, a curve of the classification accuracy v.s. the pruning ratio r is shown in Fig. 4.

a) VGGNet: We start with the simpler architecture, VGGNet, which is a sequential architecture without skip connections. We find that pruning a large number of filters brings a puny performance drop. Take the VGGNet-19 as an example. On the CIFAR10 dataset, with 70% of the filters pruned, both Network Slimming and our method even bring a little performance gain. And interestingly, increasing model depth does not always enhance performance. On both CIFAR10 and CIFAR100 datasets, VGGNet-16 achieves better (or comparable) performance than VGGNet-19. These observations demonstrate the VGGNet is heavily over-parameterized for the CIFAR datasets, and that pruning a proportion of filters brings negligible influence to the performance.

b) ResNet: Pruning the ResNet architectures is more complicated because of the residual paths. As described in Sec. IV-A and Fig. 3, we preserve the number of channels in the residual path and only prune filters inside the bottleneck architecture. By pruning the same proportion of filters, our method consistently achieves better results compared with the Network Slimming [12] baseline.

### Table III

Experimental results on the SVHN dataset. Similarly, “N/A” indicates the compressed model collapses in all runs. It can be seen that our approach is tolerant of high pruning ratios and outperforms the Network Slimming (SLM) [12] baseline under various experimental settings.

| Model     | Methods | ratio r | Baseline accuracy | Finetune accuracy |
|-----------|---------|---------|-------------------|-------------------|
| VGG11     | SLM     | 0.2     | 95.85 (±0.07)     | 95.82 (±0.18)     |
|           | Ours    |         | 95.85 (±0.07)     | 96.18 (±0.09)     |
| VGG16     | SLM     | 0.4     | 95.85 (±0.07)     | 95.77 (±0.10)     |
|           | Ours    |         | 95.85 (±0.07)     | 96.20 (±0.11)     |
| VGG19     | SLM     | 0.6     | 95.85 (±0.07)     | 95.66 (±0.07)     |
|           | Ours    |         | 95.85 (±0.07)     | 96.15 (±0.05)     |
| Res164    | SLM     | 0.8     | 95.85 (±0.07)     | N/A               |
|           | Ours    |         | 95.85 (±0.07)     | 95.49 (±0.13)     |

C. Results on SVHN

We then apply the proposed pruning algorithm to the ResNet family on the SVHN dataset, following the same evaluation protocol as in Sec. IV-B. It can be seen from Tab. III that our approach outperforms the state-of-the-art baseline method [12] under various model depths and pruning ratios. Also, Network Slimming [12] often collapses when the pruning ratio is high, e.g., 80%, while our approach is more tolerant of high pruning ratios and still maintains a competitive accuracy. For example, only an accuracy of 0.10% is sacrificed for 80% of filters being pruned from the ResNet-56 backbone. Furthermore, similar to the circumstances on the CIFAR datasets, pruning a proportion of filters may even bring a performance gain (e.g., when 20% or 40% of filters are pruned), indicating a moderate pruning ratio can alleviate the over-fitting problem on the relatively small datasets, such as CIFAR and SVHN.
D. Results on ImageNet

Here, we evaluate the proposed method on the large-scale and challenging ImageNet [62] benchmark. The results of Network Slimming [12] and our method are obtained from our implementation, while other results come from the original papers. We compare against several recently-proposed pruning methods with various criteria, including the weight norm [14], norm of batch-norm factors [12], [19], and a data-dependent pruning method [29]. As summarized in Tab. IV, under the same pruning ratios, our method consistently outperforms the Network Slimming baseline, and retains a comparable number of parameters and complexity (FLOPs). Even compared with the data-dependent pruning method [29], our method still achieves competitive performance.

| Model | Methods | ratio | Acc. (%) | #Params | FLOPs |
|-------|---------|-------|----------|---------|-------|
| VGG11 | Baseline | - | 70.84 | 3.18 | 7.61 |
|       | SLM [12] | 0.50 | 68.62 | 1.18 | 6.93 |
|       | Ours | 0.50 | 69.12 | 1.18 | 6.97 |
|       | Baseline | - | 76.27 | 2.56 | 4.13 |
| Res50 | ThiNet [66] | 0.50 | 71.01 | 1.24 | 3.48 |
|       | Thinet | 0.70 | 68.42 | 0.87 | 2.20 |
|       | Li et al. [14] | N/A | 72.04 | 1.93 | 2.76 |
|       | SSR-L2.1 [68] | N/A | 72.13 | 1.59 | 1.9 |
|       | SSR-L2.0 [68] | N/A | 72.29 | 1.55 | 1.9 |
|       | SLM | 0.50 | 71.99 | 1.11 | 1.87 |
|       | Ours | 0.50 | 72.41 | 1.07 | 1.86 |
|       | Taylor [29] | 0.19 | 73.48 | 1.79 | 2.66 |
|       | SLM | 0.20 | 75.12 | 1.78 | 2.81 |
|       | Ours | 0.20 | 75.37 | 1.76 | 2.82 |
|       | Ye et al. [19]-v1 | N/A | 74.56 | 1.73 | 3.69 |
|       | Ye et al. [19]-v2 | N/A | 75.27 | 2.36 | 4.47 |
|       | Taylor [29] | 0.45 | 75.95 | 2.07 | 2.85 |
|       | SLM | 0.50 | 75.97 | 2.09 | 3.16 |
|       | Ours | 0.50 | 76.54 | 2.17 | 3.23 |
|       | Taylor [29] | 0.25 | 77.35 | 3.12 | 4.70 |
| Res101 | Ours | 0.20 | 77.36 | 3.18 | 4.81 |

With the same pruning ratio, e.g., $r = 0.5$, we assume that the importance estimation is more accurate if the pruned model (without finetuning) achieves higher performance on the validation set. Thus, the accuracy of importance estimation can be measured by the performance of pruned networks under the same pruning ratio. In this experiment, we compare the following three strategies: (a) Network Slimming [12] which measures filter importance by the batch-norm scaling factors only; (b) the dependency-aware importance estimation in Eq. (7); and (c) the dependency-aware importance estimation + automatic regularization control.

Firstly, we conduct an illustrative experiment on the VGGNet-16 backbone with a pruning ratio of 0.3. As shown in Fig. 5, the strategy (c) obtains a compressed model with the desired sparsity and achieves the best accuracy after finetuning. Then, we quantitatively compare these three strategies on the VGGNet-16 and ResNet-56 backbones. The statistics over a 10-fold validation are reported in Tab. V.

The results in Tab. V reveal that 1) the dependency-aware importance estimation is able to measure the filter importance more accurately as it achieves a much higher performance before finetuning compared with the Network Slimming, and 2) the automatic regularization control assists to derive a model with desired sparsity and search for a better architecture, evidenced by the favorable performance after finetuning.

B. Fixed v.s. Adjustable Regularization Coefficient

There are two alternative approaches that can help achieve the desired mode sparsity: (a) fix the threshold $p$ and adjust the regularization coefficient $\lambda$ during training; and (b) fix $\lambda$ and search for a suitable $p$ after training.

We compare these two alternatives on the ResNet-56 backbone with a pruning ratio of 0.5, which means 50% of the filters will be pruned. For strategy (a), the regularization coefficient $\lambda$ is fixed to $10^{-5}$, as suggested by [12].

As shown in Tab. VI, under the same pruning ratio, strategy (a) performs favorably against strategy (b) in terms of the

| Method | Before Pruning | threshold $p$ | Before Finetune | After Finetune |
|--------|----------------|---------------|----------------|---------------|
| (a)    | 60.86 | 0.01 | 60.86 | 75.24 |
| (b)    | 73.59 | 0.41 | 1.53 | 74.36 |
sparse penalty (train) and achieves the best performance after finetuning. Compared with the Network Slimming baseline, the dependency-aware importance estimation assists to identify less important filters, leading to an architecture induced by Network Slimming and our method.

Fig. 5. Training dynamics of pruning the VGGNet-16 backbone (r = 0.3) on the CIFAR100 dataset with the three different strategies. The horizontal axis represents the training epochs in all three plots. Plot (a), (b), and (c) represent the regularization coefficient \( \lambda \), model sparsity \( P \), and the finetune accuracy, respectively. Compared with the Network Slimming baseline, the dependency-aware importance estimation assists to identify less important filters, leading to higher performance before/after finetuning. Then, equipped with the automatic regularization control, the model meets the desired sparsity at the end of the first stage, and achieves the best performance after finetuning.

C. Pruning as Architecture Search

As pointed out in Sec. III-B, Network Slimming [12] may lead to unreasonable compressed architectures as too many filters can be pruned in a single layer. In this experiment, we verify that our method can derive better compressed architectures. To test the difference of the pruned architectures, we re-initialize the parameters of pruned models, and then train the pruned models for a full episode as in the standard pipeline. Note that we are essentially training the compressed architecture from scratch under the “scratch-E” setting in [67]. The results in Tab. VII indicate that our method derives better compressed architectures, as evidenced by the superior performance when training from scratch.

D. Pruning Stability

As stated in Sec. III-B, Network Slimming [12] selects filters to be pruned by ranking channel importance of different layers across the entire network, leading to unstable architectures. We empirically verify the claim that with a large pruning ratio, our method can still achieve promising results, while Network Slimming leads to collapsed models with a high probability.

Here, we design two experiments. In the first experiment, we give an intuitionistic illustration of the compressed network architecture induced by Network Slimming and our method.

TABLE VII

| Model | Method | Baseline Accuracy (%) | Finetune Accuracy (%) | Scratch |
|-------|--------|-----------------------|-----------------------|---------|
| Res164 | SLM    | 76.80 (±0.19) 74.17 (±0.33) 75.05 (±0.08) | 76.43 (±0.26) 76.43 (±0.27) 76.41 (±0.32) |
|       | Ours   |                       |                       |         |

We use the VGGNet-16 backbone with a pruning ratio of 0.4. The filter distributions of compressed architectures are shown in Fig. 6.

In the second experiment, we conduct a 5-fold validation on the CIFAR10 and CIFAR100 datasets, again using the VGGNet-16 backbone. The results in Tab. VIII indicate that under a relatively high pruning ratio, our method can still achieve high performance while Network Slimming collapses in all runs.

VI. CONCLUSION

In this paper, we propose a principled criteria to identify the unimportant filters with consideration of the inter-layer dependency. Based on this, we prune filters based on...
the local channel importance, and introduce an automatic-regularization-control mechanism to dynamically adjust the coefficient of sparsity regularization. In the end, our method is able to compress the state-of-the-art neural networks with a minimal accuracy drop. Comprehensive experimental results on CIFAR, SVHN, and ImageNet datasets demonstrate that our approach performs favorably against the Network Slimming [12] baseline and achieve competitive performance among the concurrent data-dependent and data-independent pruning approaches, indicating the essential role of the layer dependency in principled filter pruning algorithms.

ACKNOWLEDGMENTS

This research was supported by Major Project for New Generation of AI under Grant No. 2018AAA0100400, NSFC (61922046), the national youth talent support program, and Tianjin Natural Science Foundation (18ZXZNX00110).

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Neural Information Processing Systems, 2012, pp. 1097–1105.
[2] G. Deshpande, P. Wang, D. Rangaprakash, and B. Wilamowski, “Fully connected cascade artificial neural network architecture for attention deficit hyperactivity disorder classification from functional magnetic resonance imaging data,” IEEE transactions on cybernetics, vol. 45, no. 12, pp. 2668–2679, 2015.
[3] R. Girshick, “Fast r-cnn,” in International Conference on Computer Vision and Pattern Recognition, 2015, pp. 1440–1448.
[4] Y. Sun, Y. Chen, X. Wang, and X. Tang, “Deep learning face representation by joint identification-verification,” in Advances in neural information processing systems, 2014, pp. 1988–1996.
[5] Y. Yang, Z. Ma, Y. Yang, F. Nie, and H. T. Shen, “Multitask spectral clustering by exploring intertask correlation,” IEEE transactions on cybernetics, vol. 45, no. 5, pp. 1083–1094, 2014.
[6] X. Chang, Z. Ma, Y. Yang, Z. Zeng, and A. G. Hauptmann, “Bi-level semantic representation analysis for multimedia event detection,” IEEE transactions on cybernetics, vol. 47, no. 5, pp. 1180–1197, 2016.
[7] M. Luo, X. Chang, L. Nie, Y. Yang, A. G. Hauptmann, and Q. Zheng, “An adaptive semisupervised feature analysis for video semantic recognition,” IEEE transactions on cybernetics, vol. 48, no. 2, pp. 648–660, 2017.
[8] B. Du, W. Xiong, J. Wu, L. Zhang, L. Zhang, and D. Tao, “Stacked convolutional denoising auto-encoders for feature representation,” IEEE transactions on cybernetics, vol. 47, no. 4, pp. 1017–1027, 2016.
[9] Y. Wei, Y. Zhao, C. Liu, S. Wei, L. Liu, Z. Zhu, and S. Yan, “Cross-modal retrieval with cnn visual features: A new baseline,” IEEE transactions on cybernetics, vol. 47, no. 2, pp. 449–460, 2016.
[10] L. Zhang and P. N. Suganthan, “Visual tracking with convolutional random vector functional link network,” IEEE transactions on cybernetics, vol. 47, no. 10, pp. 3243–3253, 2016.
[11] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” in International Conference on Computer Vision, 2017, pp. 2736–2744.
[12] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, “Morphnet: Fast & simple resource-constrained structure learning of deep networks,” in International Conference on Computer Vision, 2018, pp. 1586–1595.
[13] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” in International Conference on Learning Representations, 2017.
[14] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” in Neural Information Processing Systems, 2016, pp. 2074–2082.
[15] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in International Conference on Machine Learning, 2015.
[16] Y. He, K. Zhang, X. Dong, Y. Fu, and Y. Yang, “Soft filter pruning for accelerating deep convolutional neural networks,” in IJCAI. International Joint Conferences on Artificial Intelligence, 2018, pp. 2234–2240.
[17] Y. He, X. Dong, K. Zhang, Y. Fu, C. Yan, and Y. Yang, “Asymptotic soft filter pruning for deep convolutional neural networks,” IEEE transactions on cybernetics, 2019.
[18] Y. Ye, X. Su, Z. Lin, and J. Z. Wang, “Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers,” in International Conference on Learning Representations, 2018.
[19] M. A. Carreira-Perpinan and Y. Idelbayev, “Learning-compression algorithms for neural net pruning,” in International Conference on Computer Vision and Pattern Recognition, 2018, pp. 8532–8541.
[20] X. Dong, S. Chen, and S. Pan, “Learning to prune deep neural networks via layer-wise optimal brain surgeon,” in Neural Information Processing Systems, 2017, pp. 4857–4867.
[21] Y. Guo, A. Yao, and Y. Chen, “Dynamic network surgery for efficient dnns,” in Neural Information Processing Systems, 2016, pp. 1379–1387.
[22] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in Neural Information Processing Systems, 2015, pp. 1135–1143.
[23] B. Hassibi and D. G. Stork, “Second order derivatives for network pruning: Optimal brain surgeon,” in Neural Information Processing Systems, 1993, pp. 164–171.
[24] Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” in Neural Information Processing Systems, 1990, pp. 598–605.
[25] S. Srinivas, A. Subramanya, and R. Venkatesh Babu, “Training sparse neural networks,” in CVPRW, 2017, pp. 138–145.
[26] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, “Filter pruning via geometric median for deep convolutional neural networks acceleration,” in International Conference on Computer Vision and Pattern Recognition, 2019, pp. 4340–4349.
[27] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in International Conference on Computer Vision, 2017, pp. 1398–1406.
[28] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, “Importance estimation for neural network pruning,” in International Conference on Computer Vision and Pattern Recognition. IEEE, 2019, pp. 11 264–11 272.
[29] L. Zeng and X. Tian, “Accelerating convolutional neural networks by removing interepatial and interkernel redundancies,” IEEE transactions on cybernetics, vol. 50, no. 2, pp. 452–464, 2018.
[30] A. Polyak and L. Wolf, “Channel-level acceleration of deep face representations,” IEEE Access, vol. 3, pp. 2163–2175, 2015.
[31] Z. Zheng, Z. Li, A. Nagar, and K. Park, “Compact deep neural networks for device based image classification,” in 2015 IEEE International Conference on Multimedia & Expo Workshops, ICME Workshops 2015, Turin, Italy, June 29 - July 3, 2015, 2015, pp. 1–6.
[32] S. Anwar, K. Hwang, and W. Sung, “Structured pruning of deep convolutional neural networks,” ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 13, no. 3, p. 32, 2017.
[33] Y. Zhou, Y. Zhang, Y. Wang, and Q. Tian, “Accelerate cnn via recursive bayesian pruning,” in International Conference on Computer Vision, 2018.
[34] Y. Zhou, G. G. Yen, and Z. Yi, “A kneeguided evolutionary algorithm for compressing deep neural networks,” IEEE transactions on cybernetics, 2019.
[35] P. T. Fletcher, S. Venkatasubramanian, and S. Joshi, “Robust statistics on riemannian manifolds via the geometric median,” in International Conference on Computer Vision and Pattern Recognition. IEEE, 2008, pp. 1–8.
[36] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in International Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[37] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in International Conference on Computer Vision and Pattern Recognition, 2017, pp. 4700–4708.
[38] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in ICLR, 2015.
[39] H. Cai, L. Zhu, and S. Han, “ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware,” in International Conference on Learning Representations, 2019.
[40] X. Dai, P. Zhang, B. Zhou, X. Yin, F. Sun, Y. Yang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia et al., “Channet: Towards efficient network design through platform-aware model adaptation,” in International Conference on Computer Vision and Pattern Recognition, 2019, pp. 11 398–11 407.
