Pruning by Training: A Novel Deep Neural Network Compression Framework for Image Processing

Guanzhong Tian, Jun Chen, Xianfang Zeng, and Yong Liu, *Member, IEEE*

***Abstract*—Filter pruning for a pre-trained convolutional neural network is most normally performed through human-made constraints or criteria such as norms, ranks, etc. Typically, the pruning pipeline comprises two-stage: first learn a sparse structure from the original model, then optimize the weights in the new prune model. One disadvantage of using human-made criteria to prune filters is that the design and selection of threshold criteria depend on complicated prior knowledge. Besides, the pruning process is less robust due to the impact of directly regularizing on filters. To address the problems mentioned, we propose an effective one-stage pruning framework: introducing a trainable collaborative layer to jointly prune and learn neural networks in one go. In our framework, we first add a binary collaborative layer for each original filter. Then, a new type of gradient estimator - asymptotic gradient estimator is first introduced to pass the gradient in the binary collaborative layer. Finally, we simultaneously learn the sparse structure and optimize the weights from the original model in the training process. Our evaluation results on typical benchmarks, CIFAR and ImageNet, demonstrate very promising results against other state-of-the-art filter pruning methods.**

***Index Terms*—Convolutional neural networks, effective feature scheme, filter pruning, model compression.**

I. Introduction

DEEP neural networks have exhibited their incomparable success on diverse tasks such as image classification, semantic segmentation, object detection, etc. However, people still find it difficult to practically deploy existing deep models on real-world applications such as mobile phones due to the overwhelming memory consumption and computation cost. Therefore, how to obtain a compact and efficient network from existing models become a research hotspot.

Current methods for pruning CNN models mainly focus on weight pruning [1], [2], and [22] and filter pruning [4], [6], [10], [12], [15], and [16]. Most of these pruning methods design various threshold criteria or constraints based on prior knowledge and separate the pruning process from the training process. However, there is one weakness for these filter pruning methods [10], [15], and [16]: Once being pruned, the filters cannot be retrieved. Moreover, the generalization abilities and training stabilities are constrained due to directly regularizing on filters. To address these disadvantages, we abandon the traditional way of finding out a pruning criterion and attempt to obtain a sparse structure from the original model by training. Just like [19] integrate low-rank approximation and regularization into the training process, we integrate pruning into training.

In this letter, we present a novel one-stage pruning framework that adaptively combines filter pruning with parameter optimization in one go. After inserting the collaborative layer, we perform a dot-product operation between the convolutional layer and its corresponding collaborative layer, which is shown in Fig. 1. Then original filters that have 0-valued corresponding masks from collaborative layers are directly removed after training. Finally, all collaborative layers are removed as well. The output weights in the collaborative layer are binary and the gradients are zero everywhere. Therefore, the key challenge of our framework is how to pass the gradient through the collaborative layer. By inserting a collaborative layer, we can the aggregate gradient perturbations induced by noise or pruning filters into the collaborative layer, which will be removed in the end. Moreover, temporarily incorrect pruning filters can be recovered in our framework since the weights in collaborative layers are alterable in the training process. We cannot directly adopt a gradient descent algorithm to train the model since gradients will vanish in the collaborative layers and cannot pass through. To solve this problem, we introduce a novel asymptotic gradient estimator to estimate gradients in the collaborative layer. We prove the rationality and effectiveness of our asymptotic gradient estimator through both mathematical proof (Theorem 1) and experiment results (Fig. 2). To testify our proposed algorithm, we evaluate our method on VGGNet and ResNet with various widths.

II. Problem Formulation

Considering a training dataset consisting of N sample pairs $\{x_i, y_i\}_{i=1}^{N}$. Then a CNN model can be described as $f_(x_i, W)$ with collection of all parameters W. W can be obtained by solving the optimization problem:

$$\min_{w} \frac{1}{n}\left(\sum_{i=1}^{n} L(f(x_i, W), y_i)\right) + R(W), \tag{1}$$

where $L(f(x_i, W), y_i)$ is the loss and $R(W)$ is a regularization term denotes the amount of non-zero filters. We need to find a

Manuscript received December 7, 2020; revised January 19, 2021; accepted January 20, 2021. Date of publication January 25, 2021; date of current version February 12, 2021. This work is funded in part by theNational Key R&D Program of China under Grant 2018AAA0101503 and in part by the Science and Technology Project of SGCC (State Grid Corporation of China): Fundamental theory of human-in-the-loop hybrid-augmented intelligence for power grid dispatch and control. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Sumohana S. Channappayya. *(Guanzhong Tian and Jun Chen contributed equally to this work.) (Corresponding author: Yong Liu.)*

The authors are with the Institute of Cyber-systems and Control, Zhejiang University, Hangzhou 310027, China (e-mail: gztian@zju.edu.cn; junc@zju.edu.cn; zzlongjuanfeng@zju.edu.cn; yongliu@iipc.zju.edu.cn).

Digital Object Identifier 10.1109/LSP.2021.3054315

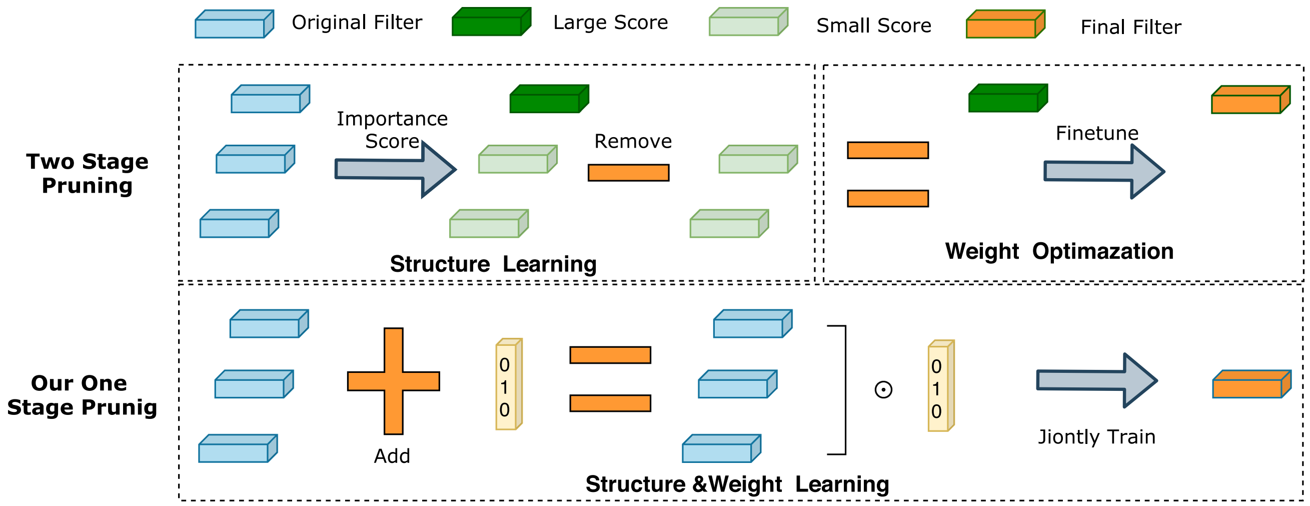

Fig. 1. Pruning procedures for two-stage based filter pruning method and our proposed one-stage method. Both methods prune some of the filters and keep the others. Blue boxes indicate the original filters, and green boxes denote filters with different important scores, where deeper color represents a larger score. Only filters with larger scores are preserved based on the assumption that smaller score filters are less critical. In contrast, our proposed framework prunes filter with the corresponding '0' values in the collaborative layer.

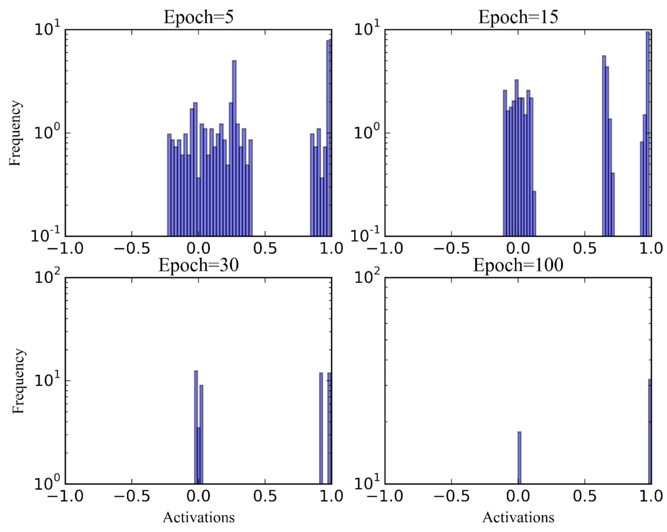

Fig. 2. Histogram of output values for collaborative layer in ResNet-20 at different epochs. We select the logarithmic scale for the y axis.

minimum subset $w \in W$ to formulate a sparse structure while preserving the original model's accuracy. Since the differential for the last term does not exist, we cannot solve the problem by gradient descent. Besides, if we directly regularize on w, the batch training process will tend to be more instable.

Like [20] which use a binary indicator function to indicate whether the weights have been quantized, we bring in a similar indicator function for collaborative layers to indicate which filter to be pruned. Output Weights in collaborative layers are first designed to be binary (0 or 1) by the indicator function:

$$\sigma(v_j^i) = \begin{cases} 0, & \text{if } \left|v_j^i\right| \le t, \\ 1, & \text{if } \left|v_j^i\right| > t, \end{cases} \tag{2}$$

where i represents the $i-th$ convolutional layer and j represents the $j-th$ filter in that layer. v_i^j denotes the original parameters in the collaborative layer. As shown in Eq. 2, neurons in the collaborative layer are designed to face binary decisions: when the absolute value exceeds a certain threshold t, the output is '1' and vice versa. The mask '1' means the corresponding filter should be preserved and '0' means that it should be removed. By fusing the threshold hyperparameter t into the indicator function, we are able to control the pruning rate of each conventional layer. Therefore, we do not need to configure the pruning rate offline as in other methods. For different convolutional layer, the value of t can be different. After inserting the collaborative layer, the optimization problem can be reformulated as

$$\min_{m,w} \frac{1}{n}\left(\sum_{i=1}^{n} L(f(x_i, W \odot V), y_i)\right), \tag{3}$$

where V is weight matrix of the collaborative layer. Obviously, the change of v_j^i will not have a direct impact on w_j^i, thus making the training process more robust.

III. Gradient Estimator

As described in Eq. 2, the indicator function in collaborative layers is a step function and output values are designed to be binary. In this case, the standard back-propagation algorithm cannot work for training these binary collaborative layers since the gradients of constant step functions are zero everywhere. To solve this problem, Hinton introduces Straight Through Estimator (STE) [7] to estimate gradient as if back-propagating by the identity function. The performance of Binarized neural networks (BNNs) [8] where binary weights are represented using sign function and trained with hard sigmoid STE prove the effectiveness and practicability of STE. However, the estimated gradient from STE is definitely not the original gradient from the chain rule, making the forward and backward propagation do not match for the training process with STE. Besides, we still can not prove the rationality of STE through mathematical derivations.

Based on the observation that constant indicator functions will generate zero gradients, we propose to add another non-constant term into the indicator function to avoid '0' gradients. Therefore, we propose to adopt a different indicator function to replace the original step function and bypass the gradient vanish problem. Without the loss of generality, the indicator function in our collaborative layer are defined as follows:

$$a_j^i = \lambda\sigma(v_j^i) + (1-\lambda)v_j^i, \tag{4}$$

The first term is the original indicator function $\sigma(v_j^i)$ defined in Eq. (2), which can be used to generate binary values. And the second term is an identity function to generate gradients. $\lambda \in (0, 1)$ is an adjustable hyperparameter to balance the two terms. The output of collaborative layer: a_j^i degenerates into binary values if $\lambda = 1$ and degenerates into full-precision values if $\lambda = 0$. Obviously, when $\lambda \neq 1$, Eq. (4) is a continuous function and the output values are not binary. But we can prove that when the training process is accomplished, our indicator function is able to approach the discrete function and generate binary results. The theorem and its proof is presented in the following:

Theorem 1: When the number of iterations is adequate, a_j^i will approach to binary values.

Proof: Since the training process is a iterative process, we can suppose $v_j^i(n)$ is the n-th iteration of v_j^i, then $a_j^i(n) = v_j^i(n+1)$ is the next iteration of $v_j^i(n)$. We can assume a_0 is the value chosen from $\sigma(v_j^i)$ (0,1). Therefore, using Eq. (4), we can obtain the general terms of series $a_j^i(n)$:

$$\begin{cases} a_j^i(2) - (1-\lambda)a_j^i(1) = \lambda a_0 & (1) \\ a_j^i(3) - (1-\lambda)a_j^i(2) = \lambda a_0 & (2) \\ \quad\vdots & \\ a_j^i(n) - (1-\lambda)a_j^i(n-1) = \lambda a_0 & (n-1) \\ a_j^i(n+1) - (1-\lambda)a_j^i(n) = \lambda a_0 & (n) \end{cases}, \tag{5}$$

Let $(1-\lambda)^{n-1} \times (1) + (1-\lambda)^{n-2} \times (2) + (1-\lambda) \times (n-1) + (1-\lambda)^0 \times (n)$, then we can get:

$$\begin{aligned} a_j^i(n+1) - (1-\lambda)^{n-1} \\ &= \lambda a_0[(1-\lambda) + (1-\lambda)^2 + \cdots + (1-\lambda)^{n-1} \\ &= \lambda\frac{1-(1-\lambda)^n}{1-(1-\lambda)}a_0 \\ &= a_0[1-(1-\lambda)^n], \end{aligned} \tag{6}$$

Since $0 < \lambda < 1$, $a_j^i(n+1)$ will asymptotically approach to a_0 when the number of iterations is big enough ($n \to \infty$). This means that the output of collaborative layers will approach binary values (0 or 1) when iterations increase.

IV. Propagation and Updates

In this section, we present the training algorithm for our method, which is shown in Algorithm 1. Details like Batch Normalization layers are ignored.

First, we feed the input images into the neural network and calculate the activation values layer by layer. This step is denoted as the forward pass. Second, we calculate the training gradients according to the targets and the activation values from each layer. This step is denoted as the backward pass. Third, we update the parameters in both collaborative layers and convolutional layers based on the previous values and gradients computed in the above steps. This step is the parameter update.

Algorithm 1: Training Algorithm For Our Method (BN and Pooling Layers Are Ignored). Activation Values in Collaborative Layers Are Calculated by Eq. (2) and Eq. (4). θ is the Indicator Function For Convolutional Layers and n Is the Amount of Layers. η_1 is the Learning Rate of the Original Layers While η_2 Is the Learning Rate of the Collaborative Layer.

```
Input: Training batches of outputs and targets$(a_n, y)$,
  previous Weights $W_l$ and $V_l$, learning rate $\eta_1$ for $W$ and
  learning rate $\eta_2$ for $V$.
Output: Updated Collaborative Layer Weight Matrix
  $V^{t+1}$ and Weight Matrix $W^{t+1}$
  1. Gradient Computation:
  1.1 Forward Pass:
  for $L = 1$ to $n$ do
    $\hat{V}_l = \lambda\sigma(V_l) + (1-\lambda)(V_l)$, $\hat{W}_l = W_l \odot \hat{V}_l$
    $z_l \leftarrow a_{l-1}\hat{W}_l$, $a_l \leftarrow \theta(\tilde{z})$
  end for
  1.2 Backward Pass:
  Computing $g_{a_n} = \frac{\partial\mathcal{L}}{\partial a_n}$ based on $a_n$ and $y$
  for $L = n$ to $1$ do
    $g_{a_{l-1}}^W \leftarrow g_{a_l}W_l$, $g_{W_l} \leftarrow (g_{a_l}^W)^T a_{l-1}$
    $g_{a_{l-1}}^V \leftarrow g_{a_l}\hat{V}_l$, $g_{V_l} \leftarrow (g_{a_l}^W)^T a_{l-1}$
  end for
  2. Parameter Update:
  for $L = 1$ to $n$ do
    $W_l^{t+1} \leftarrow Update(W_t, g_{W_l}, \eta_1)$
    $V_l^{t+1} \leftarrow Update(V_t, g_{V_l}, \eta_2)$
  end for
```

V. Experimental Results and Discussions

To evaluate our proposed framework, we conduct experiments on different state-of-the-art algorithms with the most prevailing CNN models, including single-branch network VGGNet [18] and the most prevailing multiple-branch network: ResNet [3].

We adopt widely-used protocols: the amount of parameters to measure the model size of certain neural networks and FLOPs (Floating Points Operations) to measure the computational consumption. To be fair, we adjust the hyperparameter t to compare the accuracy loss under similar FLOPs or Parameters drops between algorithms. Or we evaluate the FLOPs and parameter reductions when the pruned accuracy is similar.

Since different methods may have different baseline accuracy, we adopt $Acc.\downarrow$, $FLOPs\downarrow$ and $Params.\downarrow$, which represents the reduction of accuracy, computational consumption, and model size, to further assess the performance of different methods. We implement our framework on PyTorch [17]. Normally, we set $\eta_2 = \eta_1 * 0.06$ in the training process. λ is set to start at 0.5 and end with 1.0 gradually as epochs increase.

TABLE I
LAYER BY LAYER PRUNING STATISTICS (PBT-1, PBT-2 AND PBT-3) FOR VGGNET ON CIFAR-10 DATASET

		Baseline	PBT-1	PBT-2	PBT-3
Layer	CONV1_1	64	54	48	34
	CONV1_2	64	55	49	29
	CONV2_1	128	107	87	63
	CONV2_2	128	110	92	55
	CONV3_1	256	188	164	109
	CONV3_2	256	221	176	113
	CONV3_3	256	212	175	125
	CONV4_1	512	147	152	151
	CONV4_2	512	146	143	143
	CONV4_3	512	152	150	148
	CONV5_1	512	137	144	140
	CONV5_2	512	148	146	154
	CONV5_3	512	134	130	132
	FC6	512	512	512	512
	FC7	10	10	10	10
Total Parameters		15M	2.502M	2.047M	1.498M
Accuracy		93.96	93.71	93.33	92.62
FLOPs		314.95M	145.66M	107.23M	53.05M

TABLE II
PRUNING RESULTS ON CIFAR-100/ILSVRC-2012

	Model	Params	Latency[a]	FLOPs	Top1Acc.
VGG-19	Baseline	20.3M	139μs	399.3M	72.56
	Slimming [15]	4.8M	-[b]	246.7M	73.01
	SCP [9]	4.6M	-	235.9M	72.99
	PBT	5.7M	120μs	218.5M	72.95
	Slimming [15]	2.21M	-	161.0M	67.81
	SCP [9]	2.15M	-	152.0M	72.15
	PBT	1.84M	95μs	89.3M	69.6
ResNet-50	Baseline	25.56M	2.6ms	4.09G	76.15
	SFP [4]	15.34M	1.9ms	2.40G	74.61
	FPGM [5]	15.36M	2.0ms	2.37G	75.03
	HRank [13]	16.18M	1.8ms	2.30G	74.98
	PBT	13.04M	1.9ms	2.35G	74.80
ResNet-164	Baseline	1.71M	549μs	250.1M	77.24
	Slimming [15]	1.01M	-	94.31M	71.54
	SCP [9]	0.79M	-	87.71M	75.05
	PBT	0.84M	477μs	87.52M	75.20

[*]VGG-19 and ResNet-164 are evaluated on CIFAR-100, ResNet-50 is evaluated on ILSVRC-2012.
[a]The latency is measured under batch size 1 on a server with one NVIDIA RTX 2080 GPU.
[b]means that the compared method does not report the corresponding metric value.

And hyperparameter t is set to be the same for all convolutional layers for simplicity. When the training process is finished, filters with ‘0’ mask and all the collaborative layers are removed from the model to obtain the final compressed model. It is worth mentioning that fine-tuning is not necessary for our framework.

A. Pruning Results

VGG. Table I presents the layer-wise pruning statistics of VGG-16 on the CIFAR10 dataset. We use different sets of t to generate compact models with different size, denoted as PBT-1, PBT-2, and PBT-3. We can see that our PBT still gets a pretty good accuracy (0.2%–1.5% accuracy loss) when $(6\times - 10\times)$ of the parameters and $(2\times - 6\times)$ of the FLOPs are pruned.

Table II presents our pruning results of VGG-19 on a bigger dataset: CIFAR-100. We can see that our method could prune more FLOPs under similar accuracy and parameter drop. With high pruning ratios (about 90% of the parameters are pruned), we can still achieve much better FLOPs reduction.

TABLE III
PRUNING RESULTS OF RESNET ON CIFAR-10

	Method	Ref[a] (%)	Pruned[b] (%)	Acc.↓ (%)	Params.↓ (%)	FLOPs↓ (%)
ResNet-56	SFP [4]	93.59	93.78	-0.19[d]	-	41.10
	GAL [14]	93.26	92.98	0.28	11.80	37.60
	FPGM [5]	93.59	92.93	0.66	-	**52.60**
	NISP [21]	-	-	0.03	42.60	43.61
	FCF [11]	93.14	93.38	**-0.24**	43.09	42.78
	PBT	93.41	93.12	0.29	**47.19**	43.09
ResNet-110	SFP [4]	93.68	93.38	0.30	-	40.80
	Pruning [10]	93.53	93.30	0.20	32.40	38.60
	GAL [14]	93.50	92.55	0.95	44.80	48.50
	FPGM [5]	93.68	93.73	-0.05	-	**52.30**
	NISP [21]	-	-	0.18	43.25	43.78
	FCF [11]	93.58	93.67	-0.09	43.19	43.08
	PBT	93.63	93.84	**-0.21**	**49.41**	49.59

[a]Ref represents the accuracy of the baseline neural network.
[b]Pruned represents the accuracy of the pruned neural network.
[c]means that the method does not report the corresponding metric value.
[d]Negative value means the pruned model accuracy is higher than the Ref Acc.

ResNet. We adopt plenty of state-of-the-art filter pruning algorithms as comparisons: GAL [14], FPGM [5], NISP [21], SFP [4], Pruning [10], FCF [11], Slimming [15], HRank [13] and SCP [9].

For CIFAR-10, experimental results on ResNet with different depths (56, 110) are shown in Table III. Based on the performance, we can conclude that our PBT can obtain promising results with both shallow and deep ResNet. For ResNet-56, in comparison with FCF, we observe that we can prune 4.1% more of the total parameters under similar pruned accuracy and FLOPs drop. Specially, compare to all other methods on ResNet-110, Our method achieves the best accuracy raise (0.21%) after pruning with nearly half of the parameters and FLOPs are removed (49.41% for parameters and 49.59% for FLOPs).

For CIFAR-100, we compare our pruning results with Slimming [15] and SCP [9] on ResNet-164. Results in Table II reveal that we can achieve higher accuracy under similar parameters and FLOPs drop.

For ILSVRC-2012, pruning results compare with several SOTA methods shows that we can prune much more parameters under similar accuracy and FLOPs drop.

B. Behavior of Collaborative Layer

We have proved that the collaborative layer possesses an asymptotic behavior by Theorem 1 during training. This means that the output values will gradually approach to 0 or 1. Therefore, when the training process is over, the values in the collaborative layers will be binarized. We can extract output values from all collaborative layers of different epochs in the training process to testify Theorem 1. Fig. 2 depict the distribution of output values at epoch 5, 15, 30 and 100 for ResNet-20. The histograms in Fig. 2 demonstrate that the output values asymptotically approach to 0 and 1 as the epoch grows. Moreover, the ratio of binary values is increasing as well. These validate the correctness of Theorem 1 from the experiment perspective. Therefore, we can conclude that our asymptotic gradient estimator is reasonable and effective.

REFERENCES

[1] Y. Guo, A. Yao, and Y. Chen, “Dynamic network surgery for efficient DNNs,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1379–1387.

[2] S. Han, H. Mao, and W. J Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” in *Proc. Int. Conf. Learn. Representations*, 2016.

[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[4] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, “Soft filter pruning for accelerating deep convolutional neural networks,” in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 2234–2240.

[5] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, “Filter pruning via geometric median for deep convolutional neural networks acceleration,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4335–4344.

[6] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 1389–1397.

[7] G. Hinton, N. Srivastava, and K. Swersky, “Neural networks for machine learning,” in *Coursera [Video Lectures*], 2012.

[8] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.

[9] M. Kang and B. Han, “Operation-aware soft channel pruning using differentiable masks,” in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5122–5131.

[10] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Peter Graf, “Pruning filters for efficient convnets,” in *Proc. Int. Conf. Learn. Representations*, 2017.

[11] T. Li, B. Wu, Y. Yang, Y. Fan, Y. Zhang, and W. Liu, “Compressing convolutional neural networks via factorized convolutional filters,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 3972–3981.

[12] J. Lin, Y. Rao, J. Lu, and J. Zhou, “Runtime neural pruning,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2181–2191.

[13] M. Lin *et al.*, “HRank: Filter pruning using high-rank feature map,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1529–1538.

[14] S. Lin *et al.*, “Towards optimal structured CNN pruning via generative adversarial learning,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 2785–2794.

[15] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2736–2744.

[16] Jian-Hao Luo, J. Wu, and W. Lin, “THINet: A. filter level pruning method for deep neural network compression,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 5058–5066.

[17] A. Paszke *et al.*, “Automatic differentiation in pytorch,” in *Proc. Adv. Neural Inf. Process. Syst. Autodiff Workshop*, 2017.

[18] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proc. Int. Conf. Learn. Representations*, 2015.

[19] Y. Xu *et al.*, “Trained rank pruning for efficient deep neural networks,” in *Proc. Int. Joint Conf. Artif. Intell.*, 2020, pp. 977–983.

[20] Y. Xu, Y. Wang, A. Zhou, W. Lin, and H. Xiong, “Deep neural network compression with single and multiple level quantization,” in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 4335–4342.

[21] R. Yu *et al.*, “NISP: Pruning networks using neuron importance score propagation,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9194–9203.

[22] T. Zhang *et al.*, “A systematic DNN weight pruning framework using alternating direction method of multipliers,” in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 184–199.