Training Compact CNNs for Image Classification Using Dynamic-Coded Filter Fusion

Mingbao Lin, Bohong Chen, Fei Chao, Member, IEEE, and Rongrong Ji, Senior Member, IEEE

Abstract—The mainstream approach for filter pruning is usually either to force a hard-coded importance estimation upon a computation-heavy pretrained model to select “important” filters, or to impose a hyperparameter-sensitive sparse constraint on the loss objective to regularize the network training. In this paper, we present a novel filter pruning method, dubbed dynamic-coded filter fusion (DCFF), to derive compact CNNs in a computation-economical and regularization-free manner for efficient image classification. Each filter in our DCFF is first given an inter-similarity distribution with a temperature parameter as a filter proxy, on top of which, a fresh Kullback-Leibler divergence based dynamically coded criterion is proposed to evaluate the filter importance. In contrast to simply keeping high-score filters in other methods, we propose the concept of filter fusion, i.e., the weighted averages using the assigned proxies, as our preserved filters. We obtain a one-hot inter-similarity distribution as the temperature parameter approaches infinity. Thus, the relative importance of each filter can vary along with the training of the compact CNN, leading to dynamically changeable fused filters without both the dependency on the pretrained model and the introduction of sparse constraints. Extensive experiments on classification benchmarks demonstrate the superiority of our DCFF over the compared counterparts. For example, our DCFF derives a compact VGGNet-16 with only 72.77M FLOPs and 1.06M parameters while reaching top-1 accuracy of 93.47% on CIFAR-10. A compact ResNet-50 is obtained with 63.8% FLOPs and 58.6% parameter reductions, retaining 75.60% top-1 accuracy on ILSVRC-2012. Our code, narrower models and training logs are available at https://github.com/lmbxmu/DCFF.

Index Terms—Image classification, filter pruning, filter fusion, compact CNNs.

I. INTRODUCTION

CONVOLUTIONAL neural networks (CNNs) have revolutionized many visual tasks by enabling unprecedented performance, ranging from image classification [1], [2], object detection [3], [4], visual tracking [5], [6] and many others. However, such a performance boost is often built on the basis of huge computation cost and increasing parameter amount. While it is possible to run a large-scale CNN in an environment with powerful GPUs, it is still very challenging to deploy a large CNN model on resource-constrained mobile devices and embedded systems that demand a real-time response. Thus, finding out parameter and computation redundancy in CNNs has become an active research area in computer vision.

To this end, a large collection of research work has been spurred to derive compact CNNs, so as to improve the inference efficiency without the compromise on accuracy performance. Prevailing methods include, but are not limited to, weight sharing [9], [10], [11], low-precision quantization [12], [13], [14], tensor decomposition [15], [16], [17], knowledge distillation [18], [19], [20] and network pruning [21], [22], [23]. Among these methods, pruning convolutional filters, a.k.a. filter pruning, has attracted increasing attention since it removes entire filters without changing the original convolution structures and thus without extra requirements for inference engines. According to its procedures of learning compact CNNs, we generally categorize existing methods into pretraining-dependency filter pruning and regularized-retraining filter pruning.

Pretraining-Dependency. A bunch of existing methods build filter pruning on top of a pretrained CNN model [7], [8], [24], [25], [26], [27], [28], [29], [30]. To that effect, many studies aim to preserve “important” filters measured by an intrinsic criterion based on either pretrained filter weights such as \( \ell_1 \)-norm [7] and coreset [29], or data-driven activations such as output sparsity [24], rank of feature map [8] and influence to the accuracy or loss [25], [30]. Another group formulates filter pruning as an iterative optimization problem to minimize reconstruction errors [26], [27], [28]. However, for all these methods, the capacity of pruned CNNs seriously relies on a computation-heavy pretrained model. Besides, the filter selection is hard-coded where the “important” filters are fixed, incurring a bottleneck of performance improvement [31]. In particular, fine-tuning is required to boost the accuracy. However, such fine-tuning is even more expensive than pretraining a large-scale CNN when implemented in layer-wise fashion [8], [26], [27]. As illustrated in Fig. 1, the “important” filters using \( \ell_1 \)-norm [7] or rank of feature map [8] no longer maintain high scores after fine-tuning.

Manuscript received 28 July 2021; revised 18 December 2022; accepted 13 March 2023. Date of publication 22 March 2023; date of current version 30 June 2023. This work was supported in part by the National Key R&D Program of China under Grant 20222ZD0118202, in part by the National Science Fund for Distinguished Young Scholars under Grant 62025603, in part by the National Natural Science Foundation of China under Grants U21B2037, U22B2051, 62176222, 62176223, 62176226, 62072386, 62072389, 62002305, and 62272401, and in part by the Natural Science Foundation of Fujian Province of China under Grants 2021J01002 and 2022J06001. Recommended for acceptance by V. Morariu. (Corresponding author: Rongrong Ji.)

Mingbao Lin is with the Media Analytics and Computing Laboratory, Department of Artificial Intelligence, School of Informatics, Xiamen University, Xiamen 361005, China, and also with Tencent YouTu Lab, Shanghai 200233, China (e-mail: linmb001@163.com).

Bohong Chen and Fei Chao are with the Media Analytics and Computing Laboratory, Department of Artificial Intelligence, School of Informatics, Xiamen University, Xiamen 361005, China (e-mail: bchenc@163.com; fchao@xmu.edu.cn).

Rongrong Ji is with the Media Analytics and Computing Laboratory, Department of Artificial Intelligence, School of Informatics, Institute of Artificial Intelligence, Xiamen University, Xiamen 361005, China (e-mail: rjj@xmu.edu.cn).

Digital Object Identifier 10.1109/TPAMI.2023.3259402
formulated the pruning as a convex optimization, 

By utilizing the inter-similarity distribution, a novel con-

This phenomenon contradicts the motivation that high-score 

Regularized-Retraining. This direction embeds hand-crafted 

Regularized-Retraining. This direction embeds hand-crafted 

This strategy removes the dependency on a pretrained model, it 

Overall, training compact CNNs through filter pruning re-

II. RELATED WORK

We discuss the major topics that are the most related to this 

Weight Pruning. Weight pruning removes individual neurons in 

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
with the initial weight values of the original dense model. Lin et al. [53] proposed a dynamic allocation of sparsity pattern and incorporated feedback signal to reactivate prematurely pruned weights. However, weight pruning results in an irregular sparsity which hardly supports practical speedup without delicate hardware/software [54].

Filter Pruning. In contrast, filter pruning can be well supported by general-purpose hardware and basic linear algebra subprograms (BLAS) libraries, since it removes entire filters without changing the original convolution structures. To this end, Li et al. [7] measured filter importance using the weight magnitude. Hu et al. [24] believed that channels with more sparse outputs are redundant and thus removed the corresponding filters. Lin et al. [8] observed the invariance of feature map rank and removed filters with low-rank feature maps. Molchanov et al. [25] adopted Taylor expansion to approximate the influence to the loss function induced by removing each filter. Similarly, [28] optimizes the reconstruction error of the final output response and propagates an “importance score” for each channel. [26] prunes channels using LASSO regression-based selection and the least square reconstruction. Luo et al. [27] established filter pruning as an optimization problem, and removed less important filters based on the statistics of the next layer. In [32], the scaling factor in the batch normalization (BN) layer is considered as a filter selection indicator to decide whether a filter is important. However, the influence of shifting parameters in the BN layer is totally ignored [34]. Inspired by this, [36] considers both the channel scaling and shifting parameters for pruning.

Discussion. To the best of our knowledge, only He et al.’s work [31] implements filter pruning without pretrained models or sparse constraints. However, the main differences between our DCFF and this approach are as below: (1) [31] picks up “important” filters by off-the-shelf $\ell_p$-norm, whereas we propose a fresh Kullback-Leibler divergence-based criterion by exploring the inter-similarity among different filters. (2) [31] achieves filter pruning in a soft-coded manner where “unimportant” filters are zeroized in each forward step; in contrast, our dynamic-coded scheme, as shown in Fig. 2, does not zeroize any filter, but fuses all filters into a compact set. Also, our DCFF essentially differs from the recent ResRep [39] that considers convolutional reparameterization for CNN pruning: (1) As stressed across the paper, our DCFF does not introduce any regularization. Differently, ResRep inserts a compactor consisting of $1 \times 1$ convolution regularized by penalty gradients to indicate filter removal. (2) Our DCFF is to merge all filters into a number of desired ones in the forward propagation while ResRep picks up filters corresponding to non-zero masks. (3) Our filter fusion results in a compact model after training without any post-processing; instead, ResRep requires to reparameterize the introduced compactor into the proceeding conv-BN layers after training.

III. METHODOLOGY

As discussed in Section I, existing filter pruning methods have to pretrain a computation-heavy model, or introduce a hyperparameter-sensitive regularization. In this section, we introduce our DCFF implemented in a computation-economical and regularization-free manner, by detailing its two essential components: dynamic-coded importance and filter fusion, followed by necessary analyses.

A. Preliminary

Let $M(L^{(1)}, L^{(2)}, \ldots, L^{(N)})$ be an $N$-layer CNN, where $L^{(i)}$ denotes the $i$th convolutional layer with a total of $c_{\text{out}}^{(i)}$ convolutional filters, which in this paper are represented in a matrix form $W^{(i)} = [w_1^{(i)}, w_2^{(i)}, \ldots, w_{c_{\text{out}}^{(i)}}^{(i)}] \in \mathbb{R}^{d^{(i)} \times c_{\text{in}}^{(i)}}$ with $d^{(i)} = c_{\text{in}}^{(i)} \cdot w^{(i)} \cdot h^{(i)}$, where $c_{\text{in}}^{(i)}$ is the number of input channels, and $w^{(i)}$ and $h^{(i)}$ are the width and height of the filters, respectively. Then, we append the biases of the filters to $W^{(i)}$, to form a matrix of dimensions $(d^{(i)} + 1) \times c_{\text{out}}^{(i)}$. Given its input $O^{(i-1)}$, i.e., output from the last layer, the output $O^{(i)}$ of $L^{(i)}$ is computed by

$$o_k^{(i)} = w_k^{(i)} \odot O^{(i-1)}, \quad k = 1, 2, \ldots, c_{\text{out}}^{(i)}$$

Fig. 2. Training and inference flows of our dynamic-coded filter fusion. Training: In the forward step, the original filters in the $i$th layer are fused into a smaller group of filters that form the $i$th fusion layer. The fusion layers make up the network backbone to process the input images. Notice the fused filters are intermediate results. They are unlearnable and provide gradients to update the original filters according to the chain rule. Inference: Since the inference only involves forward propagation, the fused filters are preserved and serve as our compact model for an efficient inference.
where \( O_k^{(i)} \) is the \( k \)th channel of \( O^{(i)} \) and \( \oplus \) denotes the standard convolution operation.

The goal of filter pruning is to derive an \( N \)-layer compact CNN \( M(\tilde{L}^{(1)}, \tilde{L}^{(2)}, \ldots, \tilde{L}^{(N)}) \) with a total of \( \hat{c}_c^{(i)} \) filters \( \tilde{W}^{(i)} = [\tilde{w}_1^{(i)}, \tilde{w}_2^{(i)}, \ldots, \tilde{w}_{c^{(i)}}^{(i)}] \in \mathbb{R}^{h^{(i)} \times c^{(i)}} \) in \( \tilde{L}^{(i)} \) and ideally it should be satisfied that \( \hat{c}_c^{(i)} \leq c^{(i)} \). For ease of the representation, the superscript "(i)" may be dropped from time to time in the following sections.

B. Dynamic-Coded Importance

Conventional hard-coded methods resort to selecting fixed “important” filters upon a pretrained model. We argue that these designs are paradoxical since filters that lead to high performance after fine-tuning no longer follow the high-score standards, as illustrated in Fig. 1. The main reasons include two aspects: 1) Although these criteria are indeed the intrinsic property of each filter, the inter-similarity among different filters cannot be well reflected. 2) These criteria are proposed on the basis of a pretrained model. However, as observed in Fig. 3, at different training stages, the relative importance of many filters significantly changes a lot. Besides, after training, the scores among different filters are almost the same (Layer 10). Thus, it is inappropriate to evaluate filter importance based on a pretrained model.

A suitable scenario for measuring filter importance should be constructed on the premise that it can reflect inter-similarity among filters. Also, this scenario should be conducted in a dynamic-coded manner to track real-time importance of each filter during the training of the CNN. Thus, we propose to maintain a distribution \( p_k = (p_{k1}, p_{k2}, \ldots, p_{kc_{out}}) \) as a proxy of \( w_k \). With all probabilities summed up to 1, i.e., \( \sum_{j=1}^{c_{out}} p_{kj} = 1 \), we define \( p_{kj} \) as follows:

\[
p_{kj} = \frac{\exp(-D(w_k, w_j) \cdot t)}{\sum_{g=1}^{c_{out}} \exp(-D(w_k, w_g) \cdot t)}, \quad k, j = 1, 2, \ldots, c_{out},
\]

where \( D(\cdot, \cdot) \) denotes a distance measurement, and \( t \) is a temperature parameter that controls the smoothness of the distribution proxy. While others can be adopted, we find euclidean distance performs the best as verified in Section IV-F.

In particular, the proxy, \( p_k \), standardizes all the distances by transforming each into a probability depending on all the filters, which thus models the inter-similarity between a filter \( w_k \) and other filters in a distribution space. Then, we build the importance of filter \( w_k \) on top of the proxy \( p_k \), instead of the intrinsic property of \( w_k \) such as \( \ell_1 \)-norm [7] or the rank of feature map [8]. Thus, a natural measurement for \( w_k \) can be defined through the distribution difference between \( w_k \) and others using the Kullback-Leibler (KL) divergence, as defined in the following:

\[
I_k = \frac{1}{c_{out}} \sum_{g=1}^{c_{out}} \log \frac{p_{kj}}{p_{gj}}, \quad k = 1, 2, \ldots, c_{out}.
\]

According to the definition of KL-divergence, it is conventional to derive whether the distribution \( p_k \) is different from others. If so, (3) returns a high importance score \( I_k \), denoting that \( w_k \) is more important. The rationale lies in that if one filter differentiates a lot from others, it should be representative; otherwise, \( w_k \) can be replaced with its similar counterparts and thus it is less representative. So far, we have derived our inter-similarity standard for selecting \( c_{out} \) filters in \( W \) with the highest importance scores.

Then, to realize dynamic-coded importance evaluation along with network training, one naive solution is to re-compute the filter importance before each training epoch, so as to update \( W \). However, this strategy damages the performance as experimentally verified in Section IV-E. Specifically, in the early training stage, all filters are initialized randomly and thus they should be authorized equally to compete for important filters. In this case, \( W \) is allowed to be updated drastically. However, the over-frequent updating of the important set \( W \) in the late training stages could unstabilize the network training. Therefore, the relative importance of all filters should be gradually stable as the training continues. To this end, we must adjust the temperature parameter \( t \) by formulating it in a training-adaptive manner. Thus, we derive the following:

\[
t = (T_e - T_s) \frac{1 + \exp(-E)}{1 - \exp(-E)} \frac{1 - \exp(-e)}{1 + \exp(-e)} + T_s, \quad (4)
\]

where \( T_s = 1, \quad T_e = +\infty,^{1} \quad E \) is the total number of training epochs and \( e \in [0, E) \) is the current training epoch.

Eq. (4) indicates that, starting with a small value of temperature parameter \( t = T_s \) at the beginning of training, the proxy of the distribution \( p_k \) defined in (2) becomes a soft vector and thus the important score for each filter using (3) can be easily changed, leading to a frequent updating of \( W \). While with an infinite temperature parameter \( t = T_e \), \( p_k \) is close to a one-hot distribution vector, where the relative importance score would be gradually stabilized, result of which freezes the updating of \( W \) and stabilizes the training of the network in the end.

C. Filter Fusion

By using our dynamic-coded importance described in Section III-B, we train the compact CNN from scratch to remove the dependency on pretraining a computation-heavy model. In the literature [7, 8, 24, 25, 29, 30], a compact filter set \( W = [\tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_{c_{out}}] \) is obtained by selecting \( c_{out} \) filters with the

\[^{1}T_e = 10^{4} \text{ in our practical implementation.}\]
Algorithm 1: Dynamic-Coded Filter Fusion.

Input: An N-layer CNN \( M(\{L^{(1)}, L^{(2)}, \ldots, L^{(N)}\} ) \) with filter sets \( \{ W^{(i)} \}_{i=1}^{N} \), the number of training epochs \( E \), and the number of preserved filter in each layer \( c^{(i)}_{\text{out}} \).

Output: A compact CNN \( \tilde{M}(\{\tilde{L}^{(1)}, \tilde{L}^{(2)}, \ldots, \tilde{L}^{(N)}\} ) \) with filter sets \( \{ \tilde{W}^{(i)} \}_{i=1}^{N} \) and \( \tilde{W}^{(i)} \in \mathbb{R}^{d^{(i)} \times c^{(i)}_{\text{out}}} \).

1. for \( e = 0 \rightarrow E \) do
2.  Compute the temperature \( t \) via Eq. (4);
3.  for \( i = 1 \rightarrow N \) do
4.    for \( k = 1 \rightarrow c^{(i)}_{\text{out}} \) do
5.      Compute the distribution proxy \( p_{k} \) for filter \( w^{(i)}_{k} \) via Eq. (2);
6.    end
7.    for \( k = 1 \rightarrow c^{(i)}_{\text{out}} \) do
8.      Get the importance score for filter \( w^{(i)}_{k} \) via Eq. (5);
9.    end
10.   Compute the fused filter set \( \tilde{W}^{(i)} \) via Eq. (6);
11.  Forward the input image batch using the fused filter set \( \{ \tilde{W}^{(i)} \}_{i=1}^{N} \) via Eq. (7);
12.  Update the original filter set \( \{ W^{(i)} \}_{i=1}^{N} \);
13. end

highest importance scores in \( \mathbf{W} \) as discussed in Section III-B, which can be formulated as

\[
\tilde{w}_{k} = w^{(i)}_{f(k)}, \quad k = 1, 2, \ldots, c^{(i)}_{\text{out}}, \quad (5)
\]

where \( f(k) \) returns the index \( i \in \{1, 2, \ldots, c^{(i)}_{\text{out}}\} \) of the \( k \)th filter whose importance score ranks in the \( k \)th position.

However, existing methods simply discard low-score filters to obtain the compact filter set, \( \mathbf{W} \), and ask for a fine-tuning process in order to pull back the performance. Such a way is even more time-consuming than the cost on the pretrained model when conducted in a layer-wise manner [8], [26], [27]. We believe that despite their low scores, the information of these filters is also crucial to the network performance, since the removal of them leads to significant performance degradation. The fact that filters with large importance values may have small values after fine-tuning (Fig. 1) also supports our claim. Thus, a reasonable manner should be that \( \tilde{w}_{k} \) fuses all information from the original filter set, \( \mathbf{W} \), but considers more information from the important filter \( w^{(i)}_{f(k)} \) and less from others rather than directly discarding them. This inspires us to turn back to explore the distribution proxy \( p_{f(k)} \) since it is centered on \( w^{(i)}_{f(k)} \). Under this framework, we can refine the compact filters in (5) as

\[
\tilde{w}_{k} = \mathbf{W} p_{f(k)}, \quad k = 1, 2, \ldots, c^{(i)}_{\text{out}}. \quad (6)
\]

Therefore, each fused filter, \( \tilde{w}_{k} \), is a linear combination of all filters in \( \mathbf{W} \), i.e., the weighted average regarding the distribution \( p_{f(k)} \). The innovation of our filter fusion can be explained via the training-adaptive temperature parameter. Specifically, a small temperature smooths the proxy \( p_{f(k)} \), which thus integrates more information from all filters in \( \mathbf{W} \). As the training proceeds, \( p_{f(k)} \) gradually approximates to a one-hot vector centered on \( w^{(i)}_{f(k)} \), and then our fusion formulation in (6) becomes (5). It can be seen that our filter fusion is a generalization of (5).

In the forward step, we first update the temperature parameter so as to re-compute the compact filter set \( \mathbf{W} \). Then, the convolution in the \( i \)th layer (1) under our compact training framework can be reformulated as

\[
\tilde{o}_{k}^{(i)} = \tilde{w}_{k}^{(i)} \odot \tilde{o}^{(i-1)}, \quad k = 1, 2, \ldots, c^{(i)}_{\text{out}}. \quad (7)
\]

As shown in Fig. 2, for the backpropagation, we update the original filters \( \mathbf{W}^{(i)} \) via the chain rule. After a standard network training without any sparse constraint, the compact filter sets for all layers \( \{ \mathbf{W}^{(i)} \}_{i=1}^{N} \) are then preserved for inference, which greatly facilitates the practical deployment of filter pruning and differentiates our DCFF from existing regularized-retraining studies.

We summarize the main steps of our dynamic-coded filter fusion for training compact CNNs in Algorithm 1.

IV. EXPERIMENTS

To demonstrate the ability of the proposed DCFF, in this section, we conduct model pruning for representative networks, including VGGNet-16 [42], GoogLeNet [1] and ResNet-56/110 [2] on CIFAR-10 [41]. In addition, we train compact versions of ResNet-50 [2] and MobileNet-V2 [44] on the large-scale ILSVRC-2012 [43]. We manually determine the pruned filter number \( c^{(i)}_{\text{out}} \) in this paper, and to ensure the reproducibility, we have provided all per-layer pruning ratios in our project link at https://github.com/lmbxmu/DCFF. Note that our method is complementary to the recent ABCPruner [40] and EagleEye [58] that adopt structure searching or global ranking to find a per-layer pruning ratio. Therefore, these methods can be considered as a supplementary means for determining per-layer pruning ratios without the necessity of human involvement. Usually, better performance can be observed as demonstrated in Section IV-F.

A. Training Settings

We train our compact CNN models from scratch using the SGD optimizer with a momentum of 0.9 and the batch size is set to 256. On CIFAR-10, we train the compact CNNs for a total of 300 epochs and the weight decay is set to \( 5 \times 10^{-4} \). The learning rate is initially set to 0.1, and then divided by 10 at the training points of 150 and 225 epochs. On ILSVRC-2012, a total of 90 epochs are given to train compact ResNet-50 with the weight decay set to \( 1 \times 10^{-4} \), and the initial learning rate is set to 0.1, which is then multiplied by 0.1 at the points of 30 and 60 training epochs. Besides, following [30], [37], [39], we also consider the cosine scheduler [59] to adjust the learning rate for ResNet-50/MobileNet-V2 [44] with the weight decay set to \( 1 \times 10^{-4}/4 \times 10^{-5} \). The initial learning rate is set to \( 1 \times 10^{-2}/1 \times 10^{-1} \) for ResNet-50/MobileNet-V2 [44]. Also, the training epochs for MobileNet-V2 are 180.
TABLE I

| Method       | Top-1 acc | FLOPs   | Pruning Rate | Parameters | Pruning Rate |
|--------------|-----------|---------|--------------|------------|--------------|
| VGGNet-16 [42] | 93.02%    | 314.39M | 0.0%         | 14.73M     | 0.0%         |
| SSS [33]     | 93.02%    | 183.13M | 41.6%        | 3.93M      | 73.8%        |
| Zhao et al. [34] | 93.18%    | 190.00M | 39.1%        | 3.92M      | 73.3%        |
| IIRank [8]   | 92.34%    | 108.61M | 65.3%        | 2.64M      | 82.1%        |
| Hinge [35]   | 92.91%    | 191.68M | 39.1%        | 2.94M      | 80.1%        |
| SWP [55]     | 92.85%    | 90.60M  | 71.2%        | 1.08M      | 92.7%        |
| DCFF (Ours)  | 93.47%    | 72.77M  | 76.8%        | 1.06M      | 92.8%        |
| GoogLeNet [1]  | 93.03%    | 1.53B   | 0.0%         | 0.21B      | 64.7%        |
| IIRank [8]   | 94.53%    | 0.49B   | 67.9%        | 2.18M      | 64.7%        |
| DCFF (Ours)  | 94.92%    | 0.46B   | 70.1%        | 2.08B      | 66.3%        |
| ResNet-56 [2] | 93.26%    | 127.62M | 0.0%         | 0.85M      | 0.0%         |
| L1 [7]       | 93.08%    | 90.80M  | 27.6%        | 0.73M      | 14.1%        |
| NISP [28]    | 93.01%    | 81.00M  | 35.5%        | 0.49M      | 42.4%        |
| FPGM [56]    | 93.26%    | 59.40M  | 52.6%        | -          | -            |
| LFPC [57]    | 93.24%    | 59.10M  | 52.9%        | -          | -            |
| IIRank [8]   | 93.17%    | 62.72M  | 50.0%        | 0.49M      | 42.4%        |
| SCF [36]     | 93.23%    | 61.89M  | 51.5%        | 0.44M      | 48.4%        |
| DCFF (Ours)  | 93.26%    | 55.84M  | 55.9%        | 0.38M      | 55.0%        |
| ResNet-110 [2] | 93.50%    | 297.09M | 0.0%         | 1.73M      | 0.0%         |
| L1 [7]       | 93.30%    | 155.00M | 38.7%        | 1.16M      | 32.6%        |
| HRank [8]    | 93.36%    | 105.70M | 58.2%        | 0.70M      | 59.2%        |
| LFPC [57]    | 93.07%    | 101.00M | 60.3%        | -          | -            |
| DCFF (Ours)  | 93.80%    | 85.30M  | 66.6%        | 0.56M      | 67.9%        |

For quantitative comparison, we report four widely-used metrics including accuracy, FLOPs, the amount of parameters, and the pruning rate of the compact models.

B. Performance Metrics

For quantitative comparison, we report four widely-used metrics including accuracy, FLOPs, parameters, and pruning rate. Following most off-the-shelf pruning methods, for CIFAR-10, we report the top-1 accuracy of the pruned models. As for ILSVRC-2012, we report both top-1 and top-5 classification accuracies.

C. Results on CIFAR-10

VGGNet [42]. We start by applying our DCFF to train a compact VGGNet-16. As displayed in Table I, our DCFF achieves 93.47% top-1 accuracy meanwhile removing 76.8% FLOPs and 92.8% parameters. DCFF significantly outperforms its existing competitors and leads to a large reduction of the model complexity.

GoogLeNet [1]. In Table I, compared to the state-of-the-art HRank [8], our DCFF shows its capacity to maintain a higher accuracy (94.92% vs. 94.53%) while removing more FLOPs (70.1% vs. 67.9%) and parameters (66.3% vs. 64.7%). It is worth noting that HRank heavily relies on expensive model pretraining and fine-tuning. In contrast, our DCFF simply trains a compact model from scratch, resulting in a major advantage of reducing great number of processing time.

ResNet-56/110 [2]. We continue to train compact ResNets using different depths of 56 and 110. We can see from Table I that, with more reductions of both FLOPs and parameters, the proposed DCFF well retains the performance of the original ResNet-56 and further increases the accuracy of ResNet-110 by 0.30%. These results are significantly better than off-the-shelf counterparts.

D. Results on ILSVRC-2012

We also conduct experiments on the large-scale ILSVRC-2012 for training compact ResNet-50 [2] in Table II. For fair comparison, we perform our DCFF with different pruning rates such that the accuracy can be compared under a similar complexity reduction.

ResNet-50 [2]. The compared SOTAs for ResNet-50 in Table II are HRank [8], LFPC [57], ResRep [39], AutoPruner [37] and CURL [30]. Compared with them, our DCFF achieves higher test accuracy while more FLOPs and parameters are reduced. For example, our DCFF achieves 75.18% top-1 and 92.56% top-5 accuracies after pruning 45.3% FLOPs and removing 40.7% parameters, which are better than ABCPruner that retains the accuracies of 74.84% and 92.31% after reducing 40.8% FLOPs and 33.8% parameters. In comparison with CURL that obtains 73.39% top-1 and 91.46% top-5 accuracies with the reductions of 73.2% FLOPs and 73.9% parameters, our DCFF retains better top-1 accuracy of 73.81% and top-5 accuracy of 91.59%, and meanwhile, it reduces more FLOPs of 75.1% and more parameters of 74.3%. These results verify the effectiveness of our dynamic-coded filter fusion in training a compact CNN model even on a large-scale dataset.
TABLE II
QUANTITATIVE RESULTS ON ILSVRC-2012. WE REPORT THE TOP-1 AND TOP-5 ACCURACY, THE FLOPS, THE AMOUNT OF PARAMETERS, AND THE PRUNING RATE OF THE COMPACT MODELS. ∗ SHOWS THE LEARNING RATE WITH THE COSINE SCHEDULER. DCP∗ DENOTES OUR REPRODUCED RESULTS BY REMOVING THE RECONSTRUCTION ERROR.

| Method          | Top1-acc | Top5-acc | FLOPs | Pruning Rate | Parameters | Pruning Rate |
|-----------------|----------|----------|-------|--------------|------------|--------------|
| ResNet-50 [2]   | 76.15%   | 92.96%   | 4.14B | 23.55M       | 0.0%       |              |
| ThNet-30 [27]   | 76.42%   | 88.30%   | 1.10B | 73.4%        | 8.66M      | 66.1%        |
| HRank [8]       | 69.10%   | 89.58%   | 0.98B | 76.3%        | 8.27M      | 67.6%        |
| DCCF (Ours)     | 71.54%   | 90.57%   | 2.33B | 43.7%        | 13.60M     | 39.0%        |
| SSS-26 [33]     | 71.82%   | 90.87%   | 1.55B | 62.6%        | 13.37M     | 47.7%        |
| APPruner [40]   | 73.52%   | 91.51%   | 1.79B | 56.8%        | 11.24M     | 56.0%        |
| LPPC [57]       | 74.18%   | 91.92%   | 1.60B | 61.4%        |            |              |
| NPPM [62]       | 75.96%   | 92.75%   | 2.32B | 56.0%        |            |              |
| DCP [63]        | 74.99%   | 92.20%   | 2.13B | 51.5%        | 14.19M     | 55.5%        |
| DCP∗ [63]       | 74.13%   | 91.87%   | 2.13B | 51.5%        | 14.19M     | 55.5%        |
| DCCF (Ours)     | 74.21%   | 91.93%   | 1.49B | 63.8%        | 10.58M     | 58.6%        |
| SSS-32 [33]     | 74.18%   | 91.92%   | 2.82B | 31.9%        | 18.60M     | 27.2%        |
| CP [26]         | 72.30%   | 90.80%   | 2.73B | 34.1%        | -          | -            |
| SFP [31]        | 74.61%   | 92.06%   | 2.41B | 41.8%        | -          | -            |
| ABCPruner [40]  | 74.84%   | 92.31%   | 2.45B | 40.8%        | 16.92M     | 33.8%        |
| DCCF (Ours)     | 75.18%   | 92.56%   | 2.25B | 45.3%        | 15.16M     | 40.7%        |
| ResRep∗ [39]    | 75.49%   | 92.55%   | 1.55B | 62.1%        | -          | -            |
| DCCF∗ (Ours)    | 75.60%   | 92.55%   | 1.52B | 63.0%        | 11.05M     | 56.8%        |
| AutoPruner∗ [37]| 73.05%   | 91.25%   | 1.33B | 66.4%        | 12.69M     | 50.4%        |
| DCCF∗ (Ours)    | 74.85%   | 92.41%   | 1.11B | 73.2%        | 6.67M      | 73.9%        |
| CURL∗ [30]      | 73.39%   | 91.46%   | 1.02B | 75.1%        | 6.56M      | 74.3%        |
| DCCF∗ (Ours)    | 73.81%   | 91.59%   | -     | -            | -          | -            |
| MobileNet-V2 [44]| 72.00%   | 90.12%   | 3.00M | 0.0%         | 3.50M      | 0.0%         |
| MetaPruning [61]| 68.20%   | 87.96%   | 14.0M | 53.3%        | 2.62M      | 25.1%        |
| DCCF (Ours)     | 68.60%   | 88.13%   | 14.0M | 53.3%        | 2.62M      | 25.1%        |

Note that recent methods including NPPM [62] and DCP [63] seem to perform better in Table II. Here, we would like to stress some factualities. Regarding NPPM, no parameter reduction has been reported. Empirically, more parameters lead to better performance. It is hard to tell if better performance of NPPM comes from more parameters. As for DCP, the performance gap becomes much smaller. Nevertheless, our DCCF reaches more complexity reduction of 58.60% versus 51.55% in FLOPs and 63.8% versus 55.50% in parameters. Also, we would like to stress that DCP implicitly utilizes knowledge distillation where a reconstruction error between the pre-trained model (teacher) and the pruned model (student) is built. For a more fair comparison, we reproduce DCP based on the released code but removing the reconstruction error, denoted as DCP∗. Consequently, under the same complexity reduction, we obtain 74.13% in the top-1 accuracy and 91.87% in the top-5 accuracy. Also, the pruning cost of DCP is very expensive where a total of 8 days are used to finish our reproduction including 3 days for pre-training a ResNet-50, 3 days for pruning the network layer-wisely and 2 days for fine-tuning the pruned model. In contrast, it takes around 2 days to complete each experiment of our DCCF.

**MobileNet-V2 [44].** In contrast to ResNet-50, compression on MobileNet is more challenging for its extremely compact design. For fairness, we inherit the per-layer pruning ratio from MetaPruning [61] such that the pruned model resides in the same complexity. We perform our DCCF and manifest the compared results in Table II. We can see our DCCF achieves lower top-1 errors than its counterparts. As against MetaPruning that has only 68.20% accuracy in the top-1, DCCF returns higher performance of 68.60%.

![Fig. 4](image-url)  
Top-1 accuracy of ResNet-56 for the variants of DCCF on CIFAR-10. DCCF A uses ℓ1-norm as filter importance. DCCF B has no filter fusion. DCCF C uses a fixed temperature t = 1.

**E. Performance Analysis**

To analyze the proposed method, we develop three variants of DCCF, including: (1) DCCF A: We measure the filter importance using the intrinsic property-based ℓ1-norm to replace our inter-similarity-based importance criterion defined in (3). (2) DCCF B: The filter fusion proposed in Section III-C is removed. We simply preserve the high-score filters for training while the low-score filters are discarded. (3) DCCF C: We replace the training-adaptive temperature parameter t in (4) with a constant t = 1. Note that, to pursue fair comparison among all variants, we adopt the same pruning rate for ResNet-56 in Table I, that is, 55.9% FLOPs are reduced and 55.0% parameters are removed. Then, we report the top-1 accuracy in Fig. 4.

Fig. 4 shows that DCCF achieves the best performance, with the top-1 accuracy of 93.26%. Then, by replacing our KL-divergence-based filter importance, which reflects the inter-similarity among filters with ℓ1-norm that essentially measures...
the intrinsic property of each filter, DCFF₄ decreases the performance to 92.36%, resulting in 0.9% accuracy drop. It well demonstrates that the inter-similarity-based evaluation can capture the relative importance of filters more accurately. Further, we explore the effect of our filter fusion. As can be observed, without the involvement of the filter fusion (DCFFₑ), the accuracy decreases to 91.39%, showing that low-score filters also do benefit to the accuracy performance of compact networks.

Lastly, we illustrate the necessity of using the training-adaptive temperature parameter $t$. Setting $t = 1$ (DCFFₑ) leads to a significant accuracy drop of 2.43% in comparison with our training-adaptive scheme. To dive into a deeper analysis, in Fig. 5, we visualize the high-score filters in different training epochs. The high-score filters drastically change at the beginning of the network training for both the temperature designs. As the training goes on, with $t = 1$, the high-score filters still retain a drastic change which damages the network performance as discussed in Section III-B, whilst our training-adaptive formulation gradually fixes the relative importance of filters in the late training stages and thus stabilizes the network training.

**F. Ablation Study**

In this subsection, we continue to provide some ablations including the distance measurement in (2), filter fusion in (3), the temperature $T_e$ in (4), as well as per-layer pruning ratio. All experiments are performed upon ILSVRC-2012 using ResNet-50 of 2.25B FLOPs and 15.16M parameters from Table II.

Recall in (2), we measure filter distance using euclidean distance. In Table III, we compare with other distance measurements including Manhattan distance, Correlation distance, and Cosine distance. The results manifest that the euclidean distance shows the best result among all. The potential reason for the poor performance of compared methods might be attributed to our observation that these distance measurements change more drastically than the commonly-used euclidean distance, leading to unstable network training. Therefore, across the paper, the euclidean distance is adopted to perform all experiments.

**TABLE III**

| Euclidean | Manhattan | Correlation | Cosine |
|----------|-----------|-------------|--------|
| 75.18    | 74.72     | 74.91       | 74.93  |

We continue with the analyses on our filter fusion method. Eq. (3) measures the filter importance. For comparison, we introduce two variants: First, we inversely measure the importance of each filter as $-I_k$. Second, the filter importance is randomly measured in each forward propagation. From Table IV, we can see that the compared fusion methods lead to performance degradation. In particular, the performance of “Randomly” severely drops to 55.63%, indicating that filter fusion requires a heuristic guidance. As analyzed in Section III-B, a large $I_k$ indicates $w_k$ is more representative, which however is removed by “Reversely”, therefore, performance drop occurs as well.

We finally study the influence of temperature parameter $T_e$. According to (4), $T_e$ is expected to be infinite for the purpose of a one-hot distribution vector. Table V manifests the performance w.r.t. different values of $T_e$ in the experimental implementation. For a small $T_e$, we observe severe performance drops since it destroys one-hot distribution and then unstabilizes the relative importance score. Better accuracy can be obtained by increasing $T_e$ until around $10^4$ which is large enough to maintain a one-hot distribution and therefore the performance becomes stable.

As stated in the beginning of Section IV, we manually determine the per-layer pruning ratios. Nevertheless, recent structure searching based ABCPruner [40] and global ranking based EagleEye [58] can be a supplementary means for determining per-layer pruning ratios without human involvement. To verify this, we borrow the per-layer pruning ratios from the released checkpoints of ResNet-50 [40], [58] and perform our DCFF. Table VI provides performance comparison between our manually defined per-layer pruning ratios and these based on structure searching [40]/global ranking [58]. As can be seen,
TABLE VI
PERFORMANCE COMPARISON OF PRUNED RESNET-50 BETWEEN MANUALLY DEFINED PER-LAYER PRUNING RATIO AND STRUCTURE SEARCHING [40]
GLOBAL RANKING [58]: WE REPORT THE TOP-1 ACCURACY ON ILSVRC-2012

| Tc  | Accuracy | FLOPs | Parameters |
|-----|----------|-------|------------|
| DCFF_manually | 71.54 | 363M | 7.40M |
| DCFF_manually | 72.19 | 363M | 7.35M |
| DCFF_manually | 74.21 | 1490M | 10.58M |
| DCFF_manually | 73.78 | 1490M | 9.10M |
| DCFF_manually | 74.21 | 1490M | 10.58M |
| DCFF_manually | 74.73 | 1794M | 11.24M |
| DCFF_manually | 75.18 | 2250M | 15.16M |
| DCFF_manually | 74.83 | 1891M | 11.75M |
| DCFF_manually | 75.18 | 2250M | 11.75M |
| DCFF_manually | 75.79 | 2256M | 18.02M |
| DCFF_manually | 73.18 | 1040M | 6.99M |
| DCFF_manually | 75.18 | 2250M | 15.16M |
| DCFF_manually | 75.60 | 2070M | 14.41M |

structure searching and global ranking mostly return better performance under similar model complexity reduction. Therefore, the proposed method can be effectively boosted by appropriately combining these studies that are devoted to better per-layer pruning ratios.

V. CONCLUSION

In this paper, a novel dynamic-coded filter fusion (DCFF) is introduced to train compact CNNs. The method successfully realizes the CNN pruning without the dependency on a computation-heavy pretrained model and the introduction of hyperparameter-sensitive sparsity constraints. To this end, we first maintain a distribution as a proxy of each filter, on top of which, an inter-similarity importance evaluation is devised to measure the relative importance of filters. The distribution proxy gradually approximates to a one-hot vector as its temperature parameter approaches infinity, leading to a dynamic-coded importance evaluation. Furthermore, instead of simply abandoning low-score filters, we propose to fuse all filters using the assigned distribution proxy as our preserved filters in the forward propagation. In the backward, the original filters are updated by the SGD optimizer. After a simple network training from scratch, we preserve the fused filters as our compact CNN model without any sparse constraint. Our DCFF not only advances in its simple implementation, but also shows superior ability to derive more compact models with better classification performance when compared to many recent competitors.

REFERENCES

[1] C. Szegedy et al., “Going deeper with convolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 1–9.
[2] 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.
[3] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in Proc. Adv. Neural Inf. Process. Syst., 2015, pp. 91–99.
[4] V. Ramanathan, R. Wang, and D. Mahajan, “DLWL: Improving detection for low-shot classes with weakly labelled data,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2020, pp. 9342–9352.
[5] H. Nam and B. Han, “Learning multi-domain convolutional neural networks for visual tracking,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 4293–4302.
[6] P. Li, J. Shi, and S. Shen, “Joint spatial-temporal optimization for stereo 3D object tracking,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2020, pp. 6877–6886.
[7] H. Li, A. Kadav, I. Durandovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” in Proc. Int. Conf. Learn. Representations, 2017.
[8] M. Lin et al., “HRank: Filter pruning using high-rank feature maps,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2020, pp. 1529–1538.
[9] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, “Compressing neural networks with the hushing trick,” in Proc. Int. Conf. Mach. Learn., 2015, pp. 2285–2294.
[10] Y. Gong, L. Liu, M. Yang, and L. Bourdev, “Compressing deep convolutional networks using vector quantization,” in Proc. Int. Conf. Learn. Representations, 2015.
[11] C. Zhu, S. Han, H. Mao, and W. J. Dally, “Trained ternary quantization,” in Proc. Int. Conf. Learn. Representations, 2017.
[12] L. Lin et al., “Rotated binary neural network,” in Proc. Adv. Neural Inf. Process. Syst., 2020, pp. 7474–7485.
[13] J. Yang et al., “Quantization networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2019, pp. 7308–7316.
[14] Z. Qu, Z. Zhou, Y. Cheng, and L. Thiele, “Adaptive loss-aware quantization for multi-bit networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2020, pp. 7988–7997.
[15] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, “Predicting parameters in deep learning,” in Proc. Adv. Neural Inf. Process. Syst., 2013, pp. 2148–2156.
[16] M. Jaderberg, A. Vedaldi, and A. Zisserman, “Speeding up convolutional neural networks with low rank expansions,” in Proc. Brit. Mach. Vis. Conf., 2014.
[17] H. Kim, M. U. K. Khan, and C.-M. Kyung, “Efficient neural network compression,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2019, pp. 12 569–12 577.
[18] A. Polino, R. Pascanu, and D. Alistarh, “Model compression via distillation and quantization,” in Proc. Int. Conf. Learn. Representations, 2018.
[19] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, “GAN compression: Efficient architectures for interactive conditional GANs,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2020, pp. 5284–5294.
[20] Q. Guo et al., “Online knowledge distillation via collaborative learning,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2020, pp. 11 020–11 029.
[21] M. Lin et al., “Filter sketch for network pruning,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 12, pp. 7091–7100, Dec. 2022.
[22] M. Lin et al., “Network pruning using adaptive exemplar filters,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 12, pp. 7357–7360, Dec. 2022.
[23] J. Guo, W. Ouyang, and D. Xu, “Multi-dimensional pruning: A unified framework for model compression,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2020, pp. 1508–1517.
[24] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, “Network trimming: A data-driven neuron pruning approach towards efficient deep architectures,” 2016, arXiv:1607.03250.
[25] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” in Proc. Int. Conf. Learn. Representations, 2017.
[26] 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.
[27] J.-H. 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.
[28] R. Yu et al., “NISP: Pruning networks using neuron importance score propagation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 9194–9203.
[29] A. Dubey, M. Chatterjee, and N. Ahuja, “Coreset-based neural network compression,” in Proc. Eur. Conf. Comput. Vis., 2018, pp. 454–470.
[30] J.-H. Luo and J. Wu, “Neural network pruning with residual-connections and limited-data,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2020, pp. 1458–1467.
[31] 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.
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.

Z. Huang and N. Wang, “Data-driven sparse structure selection for deep neural networks,” in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 304–320.

C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, “Variational convolutional neural network pruning,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2780–2789.

Y. Li, S. Gu, C. Mayer, L. V. Gool, and R. Timofte, “Group sparsity: The hinge between filter pruning and decomposition for network compression,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 8018–8027.

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

J. H. Luo and J. Wu, “AutoPruner: An end-to-end trainable filter pruning method for efficient deep model inference,” *Pattern Recognit.*, vol. 107, 2020, Art. no. 107461.

J. Li et al., “OICSR: Out-in-channel sparsity regularization for compact deep neural networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 7046–7055.

X. Ding et al., “ResRep: Lossless CNN pruning via decoupling remembering and forgetting,” in *Proc. IEEE Conf. Comput. Vis.*, 2021, pp. 4510–4520.

M. Lin, R. Yi, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, “Channel pruning via automatic structure search,” in *Proc. Int. Joint Conf. Artif. Intell.*, 2020, pp. 673–679.

A. Krizhevsky et al., “Learning multiple layers of features from tiny images,” Tech. Rep., 2009.

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

O. Russakovsky et al., “ImageNet large scale visual recognition challenge,” *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.

S. Vadera and S. Ameni, “Methods for pruning deep neural networks,” *IEEE Access*, vol. 10, pp. 63 280–63 300, 2022.

Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 598–605.

B. Hassibi and D. G. Stork, “Second order derivatives for network pruning: Optimal brain surgeon,” in *Proc. Adv. Neural Inf. Process. Syst.*, 1993, pp. 164–171.

S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

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

A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, “Net-Trim: Convex pruning of deep neural networks with performance guarantees,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3177–3186.

Z. Liu, J. Xu, X. Peng, and R. Xiong, “Frequency-domain dynamic pruning for convolutional neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 1043–1053.

J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” in *Proc. Int. Conf. Learn. Representations*, 2019.

T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi, “Dynamic model pruning with feedback,” in *Proc. Int. Conf. Learn. Representations*, 2020.

S. Han et al., “EIE: Efficient inference engine on compressed deep neural network,” *ACM SIGARCH Comput. Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

F. Meng et al., “Pruning filter in filter,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, Art. no. 1479.

Y. He, Y. Ding, L. Zhu, H. Zhang, and Y. Yang, “Learning filter pruning criteria for deep convolutional neural networks acceleration,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2009–2018.

B. Li, B. Wu, J. Su, and G. Wang, “EagleEye: Fast sub-net evaluation for efficient neural network pruning,” in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 639–654.

I. Loschchovil and F. Hutter, “SGDR: Stochastic gradient descent with warm restarts,” in *Proc. Int. Conf. Learn. Representations*, 2017.

J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, “Slimmable neural networks,” in *Proc. Int. Conf. Learn. Representations*, 2019.

Z. Liu et al., “MetaPruning: Meta learning for automatic neural network channel pruning,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 3296–3305.

S. Gao, F. Huang, W. Cai, and H. Huang, “Network pruning via performance maximization,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 9270–9280.

J. Liu et al., “Discrimination-aware network pruning for deep model compression,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 8, pp. 4035–4051, Aug. 2022.

**Mingbalo Lin** received the BS degree from Fuzhou University, Fuzhou, China, in 2016, and the MS and PhD degrees in intelligence science and technology from Xiamen University, Xiamen, China, in 2022. He is currently a senior researcher with the Tencent Youtu Lab, Shanghai, China. His publications on top-tier conferences/journals include IEEE Transactions on Pattern Analysis and Machine Intelligence, International Journal of Computer Vision, IEEE Transactions on Neural Networks and Learning Systems, CVPR, NeurIPS, AAAI, IJCAI, ACM MM and so on. His current research interest is to develop efficient vision model, as well as information retrieval.

**Bohong Chen** is currently working toward the BS degree with Xiamen University, China. His research interests include computer vision, and neural network compression and acceleration.

**Fei Chao** (Member, IEEE) received the BSc degree in mechanical engineering from Fuzhou University, Fuzhou, China, in 2004, the MSc degree with distinction in computer science from the University of Wales, Aberystwyth, U.K., in 2005, and the PhD degree in robotics from the Aberystwyth University, Wales, U.K., in 2009. He is currently an associate professor with the School of Informatics, Xiamen University, Xiamen, China. He has authored/co-authored more than 50 peer-reviewed journal and conference papers. His current research interests include developmental robotics, machine learning, and optimization algorithms.

**Rongrong Ji** (Senior Member, IEEE) is a Nanjing Distinguished professor with Xiamen University, the deputy director of the Office of Science and Technology with Xiamen University, and the director of Media Analytics and Computing Lab. He was awarded as the National Science Foundation for Excellent Young Scholars (2014), the National Ten Thousand Plan Young Talents (2017), and the National Science Foundation for Distinguished Young Scholars (2020). His research falls in the field of computer vision, media analysis, and machine learning. He has published more than 50 papers in ACM/IEEE Transactions, including IEEE Transactions on Pattern Analysis and Machine Intelligence and International Journal of Computer Vision, and more than 100 full papers on top-tier conferences, such as CVPR and NeurIPS. His publications have got more than 10k citations in Google Scholar. He was the recipient of the Best Paper Award of ACM Multimedia 2011. He has served as area chairs in top-tier conferences such as CVPR and ACM Multimedia. He is also an Advisory member for Artificial Intelligence Construction in the Electronic Information Education Committee of the National Ministry of Education.