EDP: An Efficient Decomposition and Pruning Scheme for Convolutional Neural Network Compression

Xiaofeng Ruan, Yufan Liu, Chunfeng Yuan, Bing Li, Weiming Hu, *Senior Member, IEEE*, Yangxi Li, and Stephen Maybank, *Fellow, IEEE*

***Abstract*— Model compression methods have become popular in recent years, which aim to alleviate the heavy load of deep neural networks (DNNs) in real-world applications. However, most of the existing compression methods have two limitations: 1) they usually adopt a cumbersome process, including pretraining, training with a sparsity constraint, pruning/decomposition, and fine-tuning. Moreover, the last three stages are usually iterated multiple times. 2) The models are pretrained under explicit sparsity or low-rank assumptions, which are difficult to guarantee wide appropriateness. In this article, we propose an efficient decomposition and pruning (EDP) scheme via constructing a compressed-aware block that can automatically minimize the rank of the weight matrix and identify the redundant channels. Specifically, we embed the compressed-aware block by decomposing one network layer into two layers: a new weight matrix layer and a coefficient matrix layer. By imposing regularizers on the coefficient matrix, the new weight matrix learns to become a low-rank basis weight, and its corresponding channels become sparse. In this way, the proposed compressed-aware block simultaneously achieves low-rank decomposition and channel pruning by only one single data-driven training stage. Moreover, the network of architecture is further compressed and optimized by a novel Pruning & Merging (PM) module which prunes redundant channels and merges redundant decomposed layers. Experimental results (17 competitors) on different data sets and networks demonstrate that the proposed EDP achieves a high compression ratio with acceptable accuracy degradation and outperforms state-of-the-arts on compression rate, accuracy, inference time, and run-time memory.**

***Index Terms*— Data-driven, low-rank decomposition, model compression and acceleration, structured pruning.**

Manuscript received November 28, 2019; revised June 24, 2020; accepted August 8, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2018AAA0102802, Grant 2018AAA0102803, Grant 2018AAA0102800, Grant 2018YFC0823003, and Grant 2017YFB1002801; in part by the Natural Science Foundation of China under Grant 61902401, Grant 61972071, Grant 61751212, Grant 61721004, Grant 61972397, Grant 61772225, Grant 61906052, and Grant U1803119; in part by the NSFC-General Technology Collaborative Fund for basic research under Grant U1636218, Grant U1936204, and Grant U1736106; in part by the Beijing Natural Science Foundation under Grant L172051, Grant JQ18018, and Grant L182058; in part by the CAS Key Research Program of Frontier Sciences under Grant QYZDJ-SSW-JSC040; in part by the CAS External Cooperation Key Project; and in part by the NSF of Guangdong under Grant 2018B030311046. The work of Bing Li was supported by the Youth Innovation Promotion Association, CAS. *(Xiaofeng Ruan and Yufan Liu contributed equally to this work.) (Corresponding authors: Chunfeng Yuan; Bing Li.)*

Xiaofeng Ruan and Yufan Liu are with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: ruanxiaofeng2017@ia.ac.cn; yufan.liu@ia.ac.cn).

Chunfeng Yuan is with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China (e-mail: cfyuan@nlpr.ia.ac.cn).

Bing Li is with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and also with PeopleAI Inc., Beijing 100190, China (e-mail: bli@nlpr.ia.ac.cn).

Weiming Hu is with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, also with the CAS Center for Excellence in Brain Science and Intelligence Technology, Beijing 100190, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: wmhu@nlpr.ia.ac.cn).

Yangxi Li is with the National Computer Network Emergency Response Technical Team/Coordination Center of China (CNCERT/CC), Beijing 100029, China (e-mail: liyangxi@outlook.com).

Stephen Maybank is with the Department of Computer Science and Information Systems, Birkbeck College, University of London, London WC1E 7HX, U.K. (e-mail: sjmaybank@dcs.bbk.ac.uk).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2020.3018177

I. Introduction

COMPARED with traditional machine learning algorithms, deep neural network (DNN) models [1]–[4] have achieved a better performance in several fields such as image classification [1], object detection [5], object tracking [6], and video understanding [7]. However, a large number of parameters and FLoating-point OPerations (FLOPs) make it difficult to deploy them in mobile and embedding devices. Therefore, the model compression of DNN is a fundamental problem that has been studied extensively in recent years.

Recently, low-rank decomposition methods [8]–[10], such as singular value decomposition (SVD) [8], have been used for model compression by decomposing an original network layer into two lightweight layers. However, there are some limitations in these low-rank decomposition methods. First, an appropriate hyperparameter for the rank of filters should be selected [8], [11], at the cost of many validation experiments. Second, the compression rates of these methods are limited if the DNN is pretrained without low-rank constraint. Third, these methods fail to remove all the redundant channels and occupy much run-time memory. Moreover, the decomposition of each layer makes the network too deep, which may influence the accuracy.

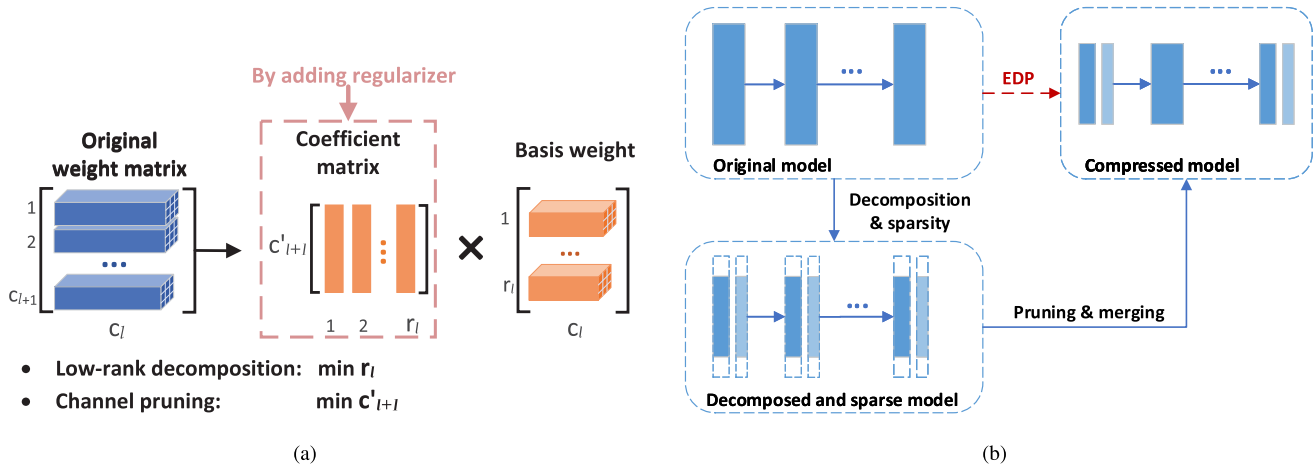


Fig. 1. Illustration of the proposed method. (a) Blue blocks comprise the original weight matrix at the lth layer, with size $c_{l+1} \times c_l$. The orange blocks represent the decomposed matrixes: a basis weight matrix with $r_l \times c_l$ size, and a coefficient matrix with $c'_{l+1} \times r_l$ size. Low-rank decomposition is achieved by minimizing r_l, whereas the sparse output channels are obtained by minimizing c'_{l+1}. (b) Whole optimization process of the proposed method.

Channel pruning methods [12], [13] have been increasingly popular in recent years. They prune unimportant input–output channel-wise connections of DNNs and are more hardware-friendly than traditional weight pruning methods. However, the learning procedure is cumbersome as it includes four stages: pretraining, training with a sparsity constraint, pruning, and fine-tuning. To achieve better performance, the whole procedure may be iterated several times. In addition, channel pruning methods change the input–output dimensions of a layer, which may perform not well in some blocks, such as element-wise addition blocks.

To overcome the above limitations, we propose an efficient decomposition and pruning (EDP) scheme via constructing a compressed-aware block. The proposed block can be used to replace every layer of a convolutional neural network (CNN) in a plug-and-play way so as to efficiently compress the network. The proposed compression procedure contains one single efficient training process and adaptively learns an optimal architecture. Fig. 1(a) shows the architecture and optimization process of the proposed method. Specifically, we embed the compressed-aware block by decomposing the original network layer into two layers: one is a basis weight matrix and the other is a coefficient matrix. By imposing $L_{2,1}$-based constraints on the coefficient matrix, the block can learn to be compact adaptively. On the one hand, the proposed constraints make the columns (i.e., input space) of the coefficient matrix to be sparse. After pruning the redundant connections, the remaining channels of the basis weight matrix can be regarded as the bases of the original weight space. Thus, the low-rank purpose is achieved. On the other hand, the proposed constraints push redundant rows (i.e., output space) of the coefficient matrix to zero. As a result, unimportant connections to the next layer are identified and then pruned. Thus, the remaining channels are sparse. Furthermore, the proposed method can degenerate into two special cases: a low-rank decomposition case and a channel pruning case. The two special cases can be utilized separately in some situations. For example, the low-rank decomposition case does not change the output dimension, which can be used in element-wise addition blocks.

The optimization of the proposed method is based on a proximal gradient algorithm. By setting approximate penalty coefficients of the regularizers, the tradeoff between accuracy and pruned ratio can be adaptively controlled. Moreover, we also find that the early stopping (ES) of these constraints can achieve a better performance. When the network is learned to be sparse and low rank, a Pruning & Merging (PM) module is deployed to prune the redundant filters and merge the redundant decomposed layers. Note that the merging operation can reduce the redundant layers, avoiding the depth of the network being too large. Finally, a lightweight and hardware-friendly DNN model with high performance is obtained. The whole process of the proposed method is shown in Fig. 1(b).

Experiments on several data sets and a range of network architectures show the effectiveness of the proposed method. We can obtain a DNN model with up to 95.59% parameters' compression, 80.11% FLOPs' reduction, 3.3× speedup of inference time, and 1.8× run-time memory saving with VGG-small architecture on CIFAR-10 data set, while keeping acceptable accuracy.

The main contributions are summarized as follows.

1) We propose a compressed-aware block, which can be put in any CNN layers as a plug-in to efficiently compress the network. Low rank and channel sparsity are adaptively achieved by introducing $L_{2,1}$-based constraints on the coefficient matrix.
2) A PM module is presented to remove not only redundant filters but also redundant decomposed layers. Thus, an optimal architecture can be adaptively obtained.
3) The proposed method is flexible. It can achieve low-rank decomposition and channel pruning, either separately or together, when compressing a network.

II. Related Works

Related works for compressing neural networks can be grouped into four categories: weight sparsity (nonstructured) pruning, parameter (weight) quantization, low-rank decomposition, and structured sparsity pruning.

Most of the early studies [14]–[17] on **weight sparsity pruning** focus on the importance of weights. They pruned the unimportant weights via different criteria. Han *et al.* [14] and Guo *et al.* [16] pruned the weights based on the magnitude of parameters. LeCun *et al.* [15] used second-derivative information to identify insignificant weights. In order to make these weights sparse, some regularizations [18], e.g., L_1 and L_2, were imposed on the loss function during the training process. After that, to further improve the performance of the compressed model, Ding *et al.* [19] gradually reduced the redundant weights to zero by directly altering the gradient flow based on momentum stochastic gradient descent (SGD). Besides, some recent works [20]–[22] articulated the lottery ticket hypothesis and learned very sparse networks (winning tickets) which obtained almost the same performance as the original network. Although the storage space can be dramatically reduced, these methods rely on specific operation libraries and hardware. The run-time memory saving is limited because most memory space is consumed by the feature maps rather than the weights.

Parameter (weight) quantization methods [23]–[27] express the floating-point weights by a few bits. For example, XNOR-Net [23], binarized neural networks [24], and binary connect [25] were proposed to quantize the floating-point weights into binary values, and quantized neural networks [26] were trained with low precision weights and activations. Han *et al.* [28] proposed a three-stage compress pipeline, in which they first pruned the model, then quantized the weights, and finally adopted Huffman coding to further increase the compression rate. However, parameter (weight) quantization methods usually result in a moderate accuracy degradation in large DNNs.

Low-rank decomposition [8], [10], [29]–[32] methods have been explored over the past few years. They decomposed the weight matrix of DNN into several pieces, using techniques such as SVD [8], [10] and canonical decomposition/parallel factors decomposition (CP decomposition) [32]. In [29], a decomposition method was presented by separating $k \times k$ filters into $k \times 1$ and $1 \times k$ filters. Kim *et al.* [30] utilized tucker decomposition on a kernel tensor to compress the networks. It consists of three steps: rank selection, low-rank tensor decomposition, and fine-tuning. This method requires additional experiments to select an appropriate rank. Most works did not consider the low-rank constraint in the training process, and thus, the compression rate is limited. Recently, Alvarez and Salzmann [31] introduced the low-rank constraint (i.e., nuclear norm) and the sparse group least absolute shrinkage and selection operator (LASSO) regularizer to train the network, before SVD-based decomposition. However, after training, the operation of SVD is time-consuming, and the decomposed layers occupy much run-time memory.

Structured pruning methods [12], [13], [33]–[35] directly remove redundant neurons and channels rather than irregular weights. Some works [12], [13] imposed LASSO regression to learn the importance of each channel, whereas several works [33], [35] ranked filters by criteria such as the absolute values and pruned the unimportant ones. In addition, some works take the correlations between filters into account. For example, [35] tied any strongly correlated neurons to a common value. Since these methods prune parts of the network structures (e.g., channels) instead of individual weights, they do not need extra libraries or hardware, unlike weight pruning methods. However, almost all these methods need to pretrain stage and fine-tune stage, and some even iterate the multiple stages to further enhance accuracy. Furthermore, channel pruning methods change the input–output dimensions of a layer, which may not match the dimensions in the case of the element-wise addition blocks. For instance, when ResNets are compressed, due to the unequal dimensions of input–output channels, the channel pruning methods cannot prune the last layer of each residual block. This may lead to insufficient compression.

In order to overcome the above weakness, the proposed method integrates low-rank decomposition and structured sparsity pruning into a unified framework. The two components can be performed either separately or together in an efficient training process, without cumbersome stages.

III. Proposed Method

In this section, we first introduce the EDP algorithm which integrates low-rank decomposition and channel pruning to reduce the redundancy. Second, a proximal gradient method is introduced to deal with the optimization of EDP. Then, a PM module is successively presented to further compact the network. The learning procedure is summarized at the end.

A. EDP Algorithm

In the original DNN, the weights of the lth layer are denoted as $\theta_l \in \mathbb{R}^{c_{l+1} \times c_l \times k_l \times k_l}$, where c_l and c_{l+1} are the number of input and output channels, respectively, and k_l is the kernel size. Without the loss of generality, we reorganize the parameters from the original space $\theta_l \in \mathbb{R}^{c_{l+1} \times c_l \times k_l \times k_l}$ to $\theta_l^{2D} \in \mathbb{R}^{c_{l+1} \times c_l k_l k_l}$. Given the input feature map f_l, the output response is obtained by

$$f_{l+1} = \sigma(f_l * \theta_l) \tag{1}$$

where $\sigma(\cdot)$ is an activation function, such as rectifier linear unit (ReLU), and $*$ is the 2-D convolution operator. Fig. 2(a) shows an illustration of the original DNN layer. The input is the feature map f_l with c_l channels, and the output is the feature map f_{l+1} with c_{l+1} channels. The parameter θ_l has c_{l+1} filters with $c_l \times k_l \times k_l$ size.

In this approach, the original layer is decomposed into two layers: one is a basis weight matrix $\theta_l^{'(2D)} \in \mathbb{R}^{r_l \times c_l k_l k_l}$ (tensor $\theta_l' \in \mathbb{R}^{r_l \times c_l \times k_l \times k_l}$), and the other one is a coefficient matrix $\beta_l^{2D} \in \mathbb{R}^{c'_{l+1} \times r_l}$ (tensor $\beta_l \in \mathbb{R}^{c'_{l+1} \times r_l \times 1 \times 1}$), which represents the coefficients of the bases. Therefore, we compute the response of the decomposed layers by

$$f_{l+1} = \sigma(f_l * \theta_l' * \beta_l). \tag{2}$$

Note that the size of β_l^{2D} reflects the number of ranks (i.e., r_l) and channels (i.e., c'_{l+1}). By imposing $L_{2,1}$ regularization on the coefficient matrix β_l^{2D} and its transposition $(\beta_l^{2D})^T$, we obtain the basis weight $\theta_l^{'(2D)}$ with fewer rows r_l and fewer

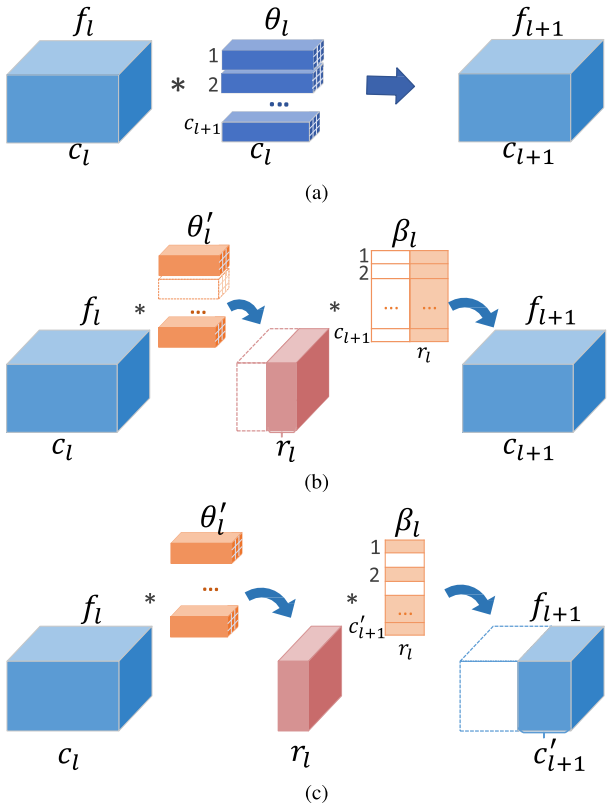


Fig. 2. Illustration of the proposed algorithm. The blocks shown only by outlines are eventually pruned. (a) Original layer. (b) Our layer: Low-rank decomposition. (c) Our layer: Channel pruning.

output channels c'_{l+1}, simultaneously. In this manner, given input–output pairs $(\mathbf{x}, \mathbf{y})$ from data set $\mathcal{D}$, the loss function is formulated as

$$\text{Loss} = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \ell(\mathcal{F}(\Theta, \mathbf{x}), \mathbf{y}) + \Omega(\Theta) \tag{3}$$

where $\Theta = \{\theta'_l, \beta_l\}_{l=1}^{L}$ encompasses all network parameters, and $\mathcal{F}(\cdot)$ is the DNN forward function. The first item $\ell(\cdot)$ is the standard loss function, such as cross-entropy loss. The second item $\Omega(\Theta)$ is a regularization term. In order to encourage the parameters in each layer to have low-rank and sparse channels, $\Omega(\Theta)$ is computed by

$$\Omega(\Theta) = \lambda_1 \sum_{l=1}^{L} ||\beta_l^{2D}||_{2,1} + \lambda_2 \sum_{l=1}^{L} ||(\beta_l^{2D})^T||_{2,1}$$

$$\text{i.e., } ||\beta_l^{2D}||_{2,1} = \sum_{j=1}^{r_l} \sqrt{\sum_{i=1}^{c'_{l+1}} (\beta_l^{2D})^2(i,j)}. \tag{4}$$

Note that λ_1 and λ_2 are the penalty coefficients, and i and j denote the ith row and jth column, respectively. It is worth mentioning that adding constraints on β means less parameter optimization and joint optimization, which can help to promote the training performance. In ablation analysis, it is verified that adding regularizations to β is more effective and efficient than adding regularizations to θ, and joint optimization outperforms separate optimization.

By setting the values of λ_1 and λ_2, we get different special cases, as discussed in detail as follows.

①$\lambda_1 \neq 0, \lambda_2 = 0$: Low-rank decomposition component

In this case, (4) is reduced to

$$\Omega(\Theta) = \lambda_1 \sum_{l=1}^{L} ||\beta_l^{2D}||_{2,1}. \tag{5}$$

Hence, after training the model, a number of columns of β_l^{2D} are forced to be zero. The columns in β_l^{2D} with zero values are pruned, and the corresponding rows of $\theta_l^{'(2D)}$ are also pruned. Fig. 2(b) shows this case: the original parameter θ_l^{2D} is decomposed into a basis weight matrix $\theta_l^{'(2D)}$ with r_l filters and a coefficient matrix β_l^{2D} with c_{l+1} filters (each filter represents a row of the matrix). Note that each filter size of β_l is $r_l \times 1 \times 1$, and the initial value of r_l is c_{l+1}. During the optimization process, the filter number (i.e., r_l) of $\theta_l^{'(2D)}$ is trained to be smaller and smaller so that $\theta_l^{'(2D)}$ becomes a low-rank basis weight matrix. In this case, we achieve low-rank decomposition of a DNN layer by training with (5). It is applicable to every type of DNN, including ResNet [3], since it does not change the input–output dimensions. By using the coefficient matrix β_l^{2D} with $L_{2,1}$ regularizer, the low-rank basis weight $\theta_l^{'(2D)}$ can also be automatically generated without manual rank selection.

For the need of keeping invariant in input–output dimensions, we use the decomposition case to compress the layer. This is because the decomposition case degenerated from the proposed method only decomposes one layer into two layers and does not influence the input–output dimensions.

② $\lambda_2 \neq 0, \lambda_1 = 0$: Channel pruning component

In terms of this case, (4) is reduced to

$$\Omega(\Theta) = \lambda_2 \sum_{l=1}^{L} ||(\beta_l^{2D})^T||_{2,1}. \tag{6}$$

Similarly, β_l^{2D} is trained to have sparse rows by (6). Pruning the rows with zero values leads to a reduction in the number of output channels. As shown in Fig. 2(c), the number of output channels c'_{l+1} is initialized to c_{l+1}. The number of output channels becomes smaller as the training proceeds. Hence, a more compact output f_{l+1} is obtained, with the saving of much run-time memory. In comparison with existing channel pruning methods, such as Slimming [13], this case only needs one whole training process without multistage pruning and achieves a high performance.

③ $\lambda_1 \neq 0, \lambda_2 \neq 0$: EDP

When λ_1 and λ_2 are both nonzero, we make use of the low-rank decomposition component and channel pruning component, simultaneously. The initial values of r_l and c'_{l+1} are both set to c_{l+1}. The DNN is then trained by minimizing the loss function in (3). After pruning, a low-rank basis weight θ'_l with a small number (i.e., r_l) of filters and an output feature map f_{l+1} with a small number (i.e., c'_{l+1}) of channels are obtained. In EDP, the two components compensate for the drawbacks of each other. This ensures an extreme compression rate with only a small decline in performance.

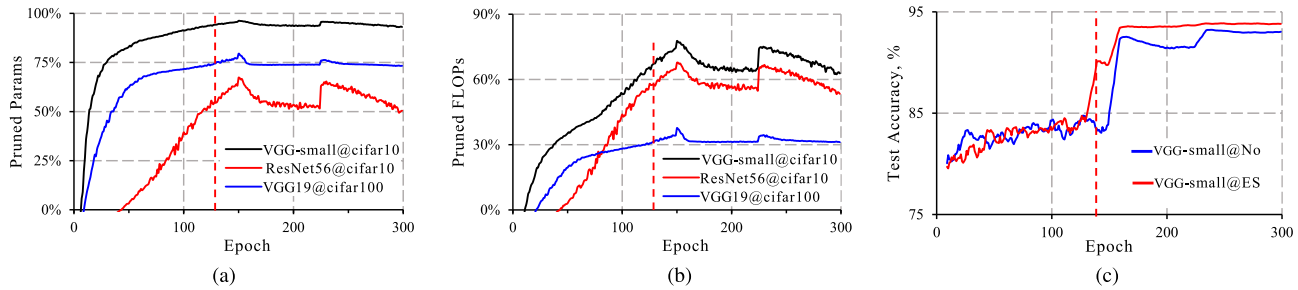


Fig. 3. Analysis of the ES strategy of optimization with regularizations. Note that the red imaginary line represents the ES epoch. No: training without an ES strategy; ES: training with ES strategy. (a) Sparse ratio of parameters. (b) Sparse ratio of FLOPs. (c) Test accuracy comparison.

B. Optimization

Optimization: The optimization objective is to minimize the loss defined in (3)

$$\min_{\Theta} \quad \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \ell(\mathcal{F}(\Theta,\mathbf{x}),\mathbf{y}) + \Omega(\Theta). \tag{7}$$

For the convenience of subsequent analysis, (7) is rewritten as

$$\begin{aligned} &\min_{\Theta} \quad g(\Theta) + \Omega(\Theta) \\ &\text{i.e., } g(\Theta) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \ell(\mathcal{F}(\Theta,\mathbf{x}),\mathbf{y}). \end{aligned} \tag{8}$$

The term $g(\Theta)$ is a widely used DNN loss function (e.g., the cross-entropy loss for classification tasks) which is smooth and convex. In this task, $\ell(\cdot) = -\sum_{c=1}^{M} p_c(x_i)\ log\ q_c(x_i)$, which is the cross-entropy loss. Note that M denotes the number of possible class labels, $q_c(x_i)$ is a predicted probability after the *soft-max* function that observation i is of class c, and $p_c(x_i)$ denotes a binary indicator (0 or 1) of whether class label c is the correct classification for observation i. The second term, $\Omega(\Theta)$, is convex but not differentiable. In order to solve the resulting nonsmooth unconstrained optimization problem, we make use of the proximal gradient descent method [36]. For the first term $g(\Theta)$, the gradient update can be obtained by the quadratic approximation

$$\Theta^{+} = \underset{z}{\text{argmin}} \frac{1}{2\alpha} ||z - (\Theta - \alpha\nabla g(\Theta))||_F^2 + \Omega(z) \tag{9}$$

in which α is the learning rate (LR), Θ^{+} is the next estimate of the network parameters, z is all possible approximation solution of the network parameters that we are going to predict, and Θ is from the previous iteration.

As $\Omega(\Theta)$ is only relevant to $\{\beta_l\}_{l=1}^{L}$, we update $\{\theta_l\}_{l=1}^{L}$ and $\{\beta_l\}_{l=1}^{L}$ separately. In detail, we choose the initial $\Theta^{(0)}$ and then repeat

$$\begin{aligned} \{\theta_l\}^{(n+1)} &= \{\theta_l\}^{(n)} - \alpha\nabla g(\Theta^{(n)}) \\ \{\beta_l\}^{(n)'} &= \{\beta_l\}^{(n)} - \alpha\nabla g(\Theta^{(n)}) \\ &\qquad n = 0, 1, 2, \ldots \end{aligned} \tag{10}$$

$$\begin{aligned} \{\beta_l\}^{(n+1)} &= S_{\alpha\lambda_1}(\{\beta_l\}^{(n)'}) + S_{\alpha\lambda_2}(\{\beta_l^T\}^{(n)'}). \\ &\qquad n = 0, 1, 2, \ldots \end{aligned} \tag{11}$$

Simultaneously, using the soft-thresholding algorithm [37], we can obtain

$$\begin{aligned} &[S_{\alpha\lambda_1}(\beta_l)]_j \\ &= \begin{cases} \beta_l(:,j) - \dfrac{\alpha\lambda_1\beta_l(:,j)}{||\beta_l(:,j)||_2}, & \text{if } ||\beta_l(:,j)||_2 > \alpha\lambda_1 \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{12}$$

where j denotes the column index, and $\beta_l(:,j)$ represents the jth column of β_l. Similarly

$$\begin{aligned} &[S_{\alpha\lambda_2}(\beta_l^T)]_i \\ &= \begin{cases} \beta_l(i,:) - \dfrac{\alpha\lambda_2\beta_l(i,:)}{||\beta_l(i,:)||_2}, & \text{if } ||\beta_l(i,:)||_2 > \alpha\lambda_2 \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{13}$$

Substituting (12) and (13) into (11), the optimization problem can be solved.

Early Stopping: According to the experiments, imposing the constraints throughout the whole training process probably is not optimal. Fig. 3 shows the curves of sparse ratios at different epochs. It manifests that both the parameters' sparse ratio and the FLOPs' sparse ratio reach saturation after certain epochs. In addition, when we early stop the regularizations at the saturation point and continue training the pruned model during the remaining epochs, the accuracy further increases, as shown in Fig. 3(c). Therefore, we leverage the ES strategy of regularizations in the training process. Note that ES is different from fine-tuning. In most of the existing methods, such as Slimming [13] and GrOWL [35], training with regularizations and fine-tuning are two separate training processes. Hyperparameters (e.g., LR) are reset in each training process. On the contrary, the ES is only a small training period in the whole compressed scheme, where the LR continuously decays, and other parameters are continuously updated.

C. PM Module

After optimization, it is usual to prune the redundant parameters of the network. However, in the EDP, it is not enough to only conduct a pruning operation. The decomposition component decomposes one layer into two layers, which brings about a deeper network and some redundant blocks. On the one hand, the deeper network may influence the training convergence. On the other hand, the parameters of the two decomposed layers may be more than that of one single layer. It is because the row number r_l of β_l and the output channel number c'_{l+1}

TABLE I

TEST ACCURACY OF DIFFERENT TRAINING STRATEGY IN DIFFERENT (λ_1, λ_2) SETTINGS

Strategy	Test Accuracy [a]/ Model Size		
	(0.001, 0.001)	(0.003, 0.003)	(0.005, 0.005)
No	94.08% / 3.5M	93.49% / 1.5M	92.66% / 0.7M
ES [b]	94.22% / 4.8M	93.48% / 1.9M	92.96% / 0.9M
ESPM [c]	**94.27%** / 3.3M	**94.06%** / 1.6M	**93.37%** / 0.8M

[a] Using VGG-small@cifar10.
[b] Using early-stopping strategy.
[c] Using PM module based on ES.

Algorithm 1: Proposed Method EDP

Input: Model $network$, Data set $\mathcal{D}$, LR α.
Output: a lightweight network.

```
Decompose network and initialize model parameters Θ^0;
t = 0;
while t < Epoch_ES do
    for each iteration ∈ epoch t do
        Update: Θ ← Θ − α∇g(Θ) using data set D;
        for each β_l ∈ {β_l}_{l=1}^L do
            Update β_l via (11), (12) and (13);
        end
    end
    t = t + 1;
end
Remove the redundant parameters leveraging PM module;
Continue the training until the end (Epoch_end);
Obtain a lightweight network with acceptable accuracy
  degradation.
```

are determined based on learning, unlike the existing low-rank decomposition methods that r_l is set to be much less than c'_{l+1}. Thus, in this case, it is better to merge the two layers and get more compact architecture.

Therefore, we present a PM module to prune the redundant channels and to merge the redundant decomposed layers. Specifically, we first prune the zero-value columns and rows of $\{\beta_l\}_{l=1}^L$ to get a pruned model. Based on the pruned model, if there exists

$$c'_l \times c'_{l+1} \times k_l k_l \leqslant c'_l \times r_l \times k_l k_l + r_l \times c'_{l+1} \times (1 \cdot 1)$$
$$\Longrightarrow r_l \geqslant \frac{c'_l c'_{l+1} k_l k_l}{c'_l k_l k_l + c'_{l+1}} \tag{14}$$

we conduct merging operation on the two decomposed layers and obtain a single convolutional layer with the size of $c'_{l+1} \times c'_l \times k_l \times k_l$. Experimental results in Table I show that the PM module further improves the model performance and brings about the smaller model size. It compensates for the drawback that ES slightly increases the model size.

D. Learning Procedure

We summarize the proposed method in Algorithm 1. In the whole training process, the original network is first decomposed and updated repeatedly by optimizing (3) jointly. At $Epoch_{ES}$, we early stop the regularization $\Omega(\Theta)$ and utilize the PM module to prune the zero-value channels and merge the redundant decomposed layers. After that, we continue training the network until the end. At this moment, we finish the whole training process and get a lightweight network with slight accuracy degradation.

TABLE II

VGG ARCHITECTURE VARIANTS FOR THE CIFAR DATA SET IN THIS ARTICLE. THE CONVOLUTIONAL KERNEL IS 3×3

Network	VGG-Small [40]	VGG-19 in [13]
Conv layers	2x64, Max pooling	2x64, Max pooling
	2x128, Max pooling	2x128, Max pooling
	3x256, Max pooling	4x256, Max pooling
	3x512, Max pooling	4x512, Max pooling
	3x512, Max pooling	4x512, Max pooling
FC layers	512, 512, 10	100

IV. EXPERIMENT

A. Settings

1) Data Sets: We evaluate the proposed method on three data sets: CIFAR-10, CIFAR-100 [38], and ILSVRC 2012 ImageNet [39]. There are 50 000 training images and 10 000 test images with resolution 32×32 in CIFAR-10/100 data sets. CIFAR-10 is drawn from 10 classes, whereas CIFAR-100 is split into 100 categories. All the images in the CIFAR-10 data set are normalized using mean = $[0.4914, 0.4822, 0.4465]$ and $std = [0.2470, 0.2435, 0.2616]$, and CIFAR-100 data set is normalized using $mean = [0.5071, 0.4867, 0.4408]$ and $std = [0.2675, 0.2565, 0.2761]$. ImageNet is a large-scale data set, which contains 1.28 million training images and 50 000 validation images from 1 000 classes. Images are resized to 256×256 size and normalized with $mean = [0.485, 0.456, 0.406]$ and $std = [0.229, 0.224, 0.225]$.

2) Networks: On the CIFAR-10 data set, we evaluate the proposed method using VGG-16 [2] and ResNet56 [3]. As the original VGG-16 [2] is specially designed for ImageNet classification, we use a variation version (i.e., VGG-small) taken from [40] in the experiment. To be specific, the architecture consists of 13 convolutional layers and 2 much smaller fully connected (FC) layers. Considering the convergence and the performance, we adopt batch normalization (BN) [41] after each convolutional layer and remove dropout after the FC layer. On the CIFAR-100 data set, we use VGG-19 as in [2], with one FC layer. These VGG architecture variants are shown in Table II. On the ImageNet data set, we evaluate the method on ResNets [3] (including ResNet 34, 50, and 101) and MobileNet V2 [42].

3) Implementation Details: We implement all experiments using PyTorch [51] on multiple NVIDIA GTX 1080 Ti GPUs in the training process and an Intel Core i7-6850K in the test process. During training, all the networks are trained using SGD. The batch size is set to be 100 on CIFAR-10/100 and 256 on the ImageNet data set. It takes 300 epochs for training on CIFAR-10/100 and 90 epochs on the ImageNet data set. The models are trained using an initial LR of 0.05 on CIFAR-10 and 0.01 on the CIFAR-100/ImageNet data set. The LR is multiplied by 0.1 at 50% and 75% of the training epochs

TABLE III

PERFORMANCE COMPARISON OF VGG-SMALL AND RESNET56 ON **CIFAR-10**. NOTE THAT THE TOP TWO RESULTS ARE HIGHLIGHTED WITH RED AND PINK FONTS, RESPECTIVELY. THE "--" INDICATES THAT THE RESULTS ARE NOT LISTED IN THE ORIGINAL ARTICLE

Model Architecture	Method	Test Accuracy, % (Baseline→Pruned)	Params (M[a])	FLOPs[b] ($\times 10^8$)	R_{Params} %	R_{FLOPs} %
VGG-small	Baseline	93.60	14.98	6.27	0	0
	GrOWL [35]	93.40→92.20	1.31	--	91.26	--
	GrOWL+L_2 [35]	93.40→92.70	1.03	--	93.12	--
	Li *et al.* [43]	93.25→93.40	5.40	4.12	63.90	34.21
	NRE [44]	93.46→93.40	1.09	2.03	92.72	67.64
	SVD [31]	93.60→93.46	2.04	3.60	86.38	42.53
	Slimming [13]	93.60→93.48	2.00	3.54	86.65	43.50
	HRank [45]	93.96→93.43	2.51	2.91	82.90	53.50
	Ours	**93.60→93.52**	**0.66**	**1.25**	**95.59**	**80.11**
ResNet56	Baseline	93.61	0.85	2.51	0	0
	Li *et al.* [43]	93.04→93.06	0.73	1.81	13.70	27.60
	SFP [33]	93.59→93.35	--	1.18	--	52.60
	ASFP [46]	93.59→93.12	--	1.18	--	52.60
	DCP [47]	**93.80→93.49**	0.43	1.26	49.24	49.75
	FPGM [48]	93.59→93.49	--	1.18	--	52.60
	AMC [49]	92.80→91.90	--	1.26	--	50.00
	KSE [50]	93.03→93.23	0.43	1.20	50.00	52.38
	HRank [45]	93.26→93.17	0.49	1.25	42.40	50.00
	Ours	**93.61→93.61**	**0.39**	**1.06**	**54.18**	**57.71**

[a] M: Million.

[b] Note that the FLOPs here are computed by considering both multiplication and addition operations, rather than only considering multiplication operation in some other papers.

TABLE IV

PRUNED RESULTS OF VGG19 ON **CIFAR-100**

Model Architecture	Method	Test Accuracy, % (Baseline→Pruned)	Params (M)	FLOPs ($\times 10^8$)	R_{Params} %	R_{FLOPs} %
VGG19	Baseline	73.55	20.08	7.96	0	0
	Li *et al.* [43]	73.55→73.65	5.52	4.13	72.51	48.12
	Slimming [13]	73.26→73.48	5.00	5.01	75.10	37.06
	SVD [31]	73.55→71.41	5.25	5.73	73.85	28.02
	Ours	**73.55→74.00**	**4.15**	**3.96**	**79.32**	**50.22**

on CIFAR-10/100 and 30 and 60 epoch on the ImageNet data set. We utilize a weight decay of 10^{-4} and a momentum of 0.9. Besides, we set $Epoch_{ES}$ to be 120 (for CIFAR-10/100 data sets) and 10 (for ImageNet data set) in all experiments.

4) Evaluation Metrics: We evaluate the proposed method using the number of network parameters and FLOPs (multiply adds). Note that the FLOPs are counted for the operations on convolutional and FC layers. Some calculations such as BN and other overheads are not accounted for. We also report the ratio of parameters or FLOPs reduction, as given by

$$R_{\text{Params}} = \frac{\text{Pruned Params}}{\text{Original Params}}$$
$$R_{\text{FLOPs}} = \frac{\text{Pruned FLOPs}}{\text{Original FLOPs}}. \quad (15)$$

In addition, the inference time and run-time memory are leveraged to further evaluate the proposed method.

5) Competing Methods: To analyze the effectiveness of EDP, 17 state of the arts are taken into account for comparison, including Li *et al.* [43], SVD [31], ThiNet [52], Slimming [13], NRE [44], GrOWL [35], NISP [53], SFP [33], DCP [47], AMC [49], FPGM [48], SSR [54], ASFP [46], AOFP [55], KSE [50], HRank [45], and DMC [56].

B. Performance Comparisons

In this section, we compare the method on three data sets including CIFAR-10, CIFAR-100, and ImageNet, with VGG, ResNets, and MobileNet V2 architecture. The results of these competing methods are reported according to the original article.

1) CIFAR-10: On CIFAR-10, we compare the EDP with baseline, GrOWL [35], Li *et al.* [43], NRE [44], SVD [31], Slimming [13], SFP [33], ASFP [46], DCP [47], AMC [49], FPGM [48], KSE [50], and HRank [45] in Table III. Among these compared methods, Li *et al.* [43], NRE [44], Slimming [13], SFP [33], DCP [47], ASFP [46], FPGM [48], and HRank [45] are the state-of-the-art channel pruning methods. SVD [31] is a low-rank decomposition method, and

TABLE V
PRUNED RESULTS OF RESNETS ON ILSVRC 2012 IMAGENET

Model Architecture	Method	Test Accuracy, % (Baseline→Pruned)		Accuracy↓, %		R, %	
		Top-1	Top-5	Top-1	Top-5	Params	FLOPs
ResNet34	Li *et al.* [43]	73.23→72.17	-- → --	**1.06**	--	10.8	24.2
	SFP [33]	**73.92**→71.83	**91.62**→90.33	2.09	1.29	--	41.1
	NISP [53]	-- → --	-- → --	**0.92**	--	**43.7**	**43.8**
	ASFP [46]	**73.92**→**72.53**	**91.62**→**91.04**	1.39	**0.58**	--	41.1
	Ours	73.46→**72.33**	91.39→**90.88**	1.13	**0.51**	**45.5**	**44.9**
ResNet50	ThiNet [52]	72.88→72.02	91.14→90.67	**0.86**	**0.47**	33.7	36.8
	SFP [33]	**76.15**→62.14	**92.87**→84.60	14.01	8.27	--	41.8
	NISP [53]	-- → --	-- → --	0.89	--	**44.0**	43.8
	FPGM [48]	**76.15**→74.83	**92.87**→92.32	1.32	0.55	--	**53.5**
	SSR [54]	75.12→72.29	92.30→90.73	2.83	1.57	39.2	51.3
	HRank [45]	**76.15**→**74.98**	**92.87**→**92.33**	1.17	**0.54**	36.7	43.8
	Ours	75.90→**75.34**	92.77→**92.43**	**0.56**	**0.34**	**43.9**	**52.6**
ResNet101	SFP [33]	**77.37**→**77.51**	**93.56**→**93.71**	-0.14	**-0.15**	--	**42.2**
	FPGM [48]	**77.37**→77.32	**93.56**→93.56	0.05	0.00	--	42.2
	AOFP [55]	76.63→76.88	93.29→93.49	**-0.25**	**-0.20**	--	30.1
	Ours	77.17→**77.63**	93.48→**93.62**	**-0.46**	-0.14	**41.0**	**45.4**

GrOWL [35] learns to group parameters with a high correlation. GrOWL+L_2 represents the GrOWL method adding L_2 regularizer in optimization. AMC [49] is an AutoML-based pruning method. KSE [50] is a kernel-level redundancy-based compression method. For VGG-small, shown in Table III, the proposed method achieves an accuracy of **93.52%** and a **95.59%** pruned parameter ratio, performing the best among the comparative methods. Moreover, Slimming [13], the best competing method, achieves a comparable accuracy of 93.48% but a much lower pruned parameter ratio of 86.65%. Compared with the baseline, EDP compresses VGG-small more than 95% parameters and 80% FLOPs, with less than 0.1% accuracy loss. For a more compact architecture ResNet56, EDP still outperforms all the compared methods and compresses ResNet56 more than 54% parameters and 57% FLOPs, without any accuracy degradation.

2) CIFAR-100: On CIFAR-100, the EDP is compared with baseline, Li *et al.* [43], Slimming [13], and SVD [31]. As shown in Table IV, the results completely exceed all the competing methods, not only on the accuracy but also on the ratios of parameters'/ FLOPs' reduction. Among these methods, Li *et al.* [43] achieved high accuracy but has a low compressed ratio. Slimming [13] obtained a relatively small compressed model, but the accuracy is not high. On the contrary, the EDP, containing both the advantages of decomposition and channel pruning, shows its superior effectiveness.

3) ImageNet: On the ImageNet data set, EDP is further compared with Li *et al.* [43], ThiNet [52], NISP [53], AMC [49], SFP [33], SSR [54], ASFP [46], AOFP [55], FPGM [48], HRank [45], and DMC [56] on ResNets with different depths and MobileNet V2, as shown in Tables V and VI. For ResNet34, although ASFP [46] archives the highest accuracy, the pretrained model (i.e., baseline) also has the highest accuracy, and the compressed ratio is low. On the contrary,

TABLE VI
PERFORMANCE COMPARISON OF THE COMPRESSED MOBILENETV2 (A LIGHTWEIGHT NETWORK) ON THE IMAGENET DATA SET

Method	FLOPs ($\times 10^6$)	Test Accuracy, % (Top-1)
0.75x MobileNet V2 [42]	220	69.8
AMC [49]	220	70.8
DMC [56]	162	68.4
Ours	218	**71.0**

the proposed method gets the second high accuracy but achieves the highest compressed ratio and low accuracy loss. For ResNet50, the compressed model achieves the highest test accuracy with a high compressed ratio. Especially, it reaches a very high value of FLOPs reduction ratio, far larger than most other methods (close to or even more than 10%). For ResNet101, a deep residual network, the EDP obtains the highest compressed ratio among the four methods (SFP [33], AOFP [55], FPGM [48], and the EDP). The accuracy of the compressed model even outperforms the baseline, improving 0.46% in top-1 and 0.14% in top-5. For lightweight network compression (i.e., MobileNet V2), as shown in Table VI, the method also obtains the best performance. These results indicate that the method is effective and superior to compared methods on both small-scale and large-scale data sets, due to fusing decomposition and pruning scheme and adaptively learning to compress networks.

C. Ablation Analysis

In Section IV-C, we conduct extensive experiments to further analyze and discuss the effectiveness of the proposed method. Without explicit explanation, the results are obtained by compressing VGG-small on the CIFAR-10 data set. Moreover, the "baseline" results are trained from scratch.

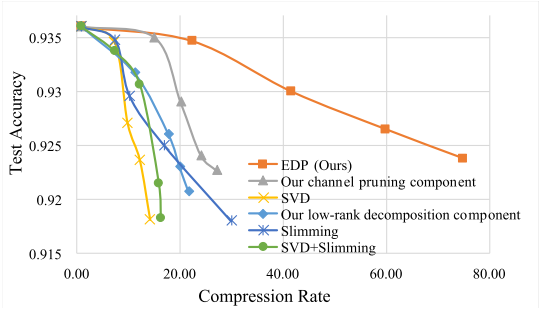


Fig. 4. Accuracy curves at different compression rates. The proposed method works best among different compression methods.

1) Effectiveness of Each Component in EDP: To verify the effectiveness of EDP, we analyze each component of the proposed method. Fig. 4 shows the accuracy of EDP and its individual component at different compression rates. The results of compared methods, such as low-rank decomposition method–SVD [31], channel pruning method–Slimming [13], and SVD + Slimming, are also reported. In particular, SVD performs SVD on the layers of a DNN model which is pre-trained with nuclear norm regularizer. Different compression rates are achieved by choosing the different ranks of the parameter matrix. Slimming imposes L_1 regularization on the scaling factors in BN and then prunes channels with low scaling factors. Further, we combine both SVD and Slimming on one model as a competing method of the full method EDP. According to the results, both of the two components of EDP outperform the competing methods, and the whole proposed method outperforms all the other methods by a large margin.

Concretely, in Fig. 4, "The low-rank decomposition component" is one of the components of the method, and it belongs to case ① in Section III-A. It takes only a single process to achieve the low-rank decomposition component, with less accuracy loss. On the contrary, SVD suffers from rapid performance drop when the network has a high compression rate. Fig. 4 shows that at a similar performance, the model size of the low-rank decomposition component is only **50%–65%** of that of the SVD model. Thus, the low-rank decomposition component is superior to some typical low-rank decomposition method–SVD.

"The channel pruning component" is the second component of the proposed method, and it belongs to case ② in Section III-A. Similar to the low-rank decomposition component, the channel pruning component is automatically achieved during the optimization procedure. On the contrary, for Slimming, the performance of compressed DNN without fine-tuning is extremely low (approximately 10% accuracy). Thus, it takes three stages (i.e., training with regularizer, pruning, and fine-tuning) for sliming to compress a model. The experimental results verify that at a similar performance, the model size of the channel pruning component is only **50**% or even less of that of the Slimming model. Besides, at the same compression rate, the channel pruning exceeds Slimming by **0.5%–2.0%**.

The full method EDP consists of the low-rank decomposition component and channel pruning component, which has the best performance in Fig. 4. EDP integrates these two components by inserting two regularizers both on the proposed coefficient matrix β_l, and the overall framework is optimized uniformly to get an optimal model. Hence, the weights' matrixes of DNN compressed by EDP naturally fulfill the low-rank and channel sparsity properties, which enable a higher compression rate with less performance gap. On the contrary, the competing method SVD+Slimming performs only similar or even worse, compared with individual SVD and individual Slimming. In this case, the arbitrary two-method combination is only able to obtain a suboptimal model and severely harms the performance of the original model.

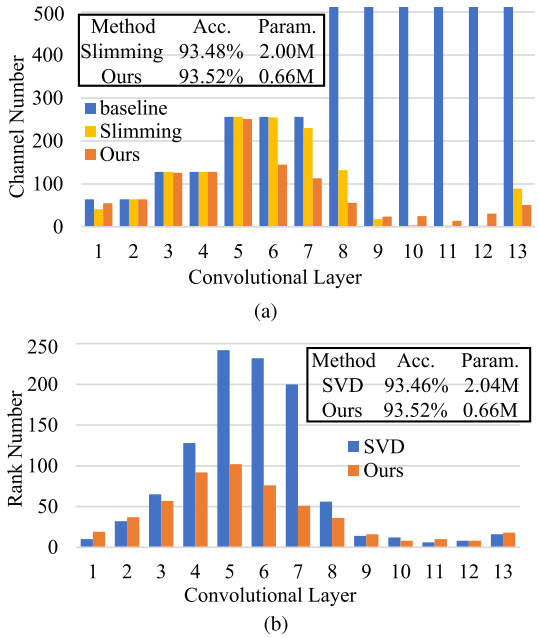


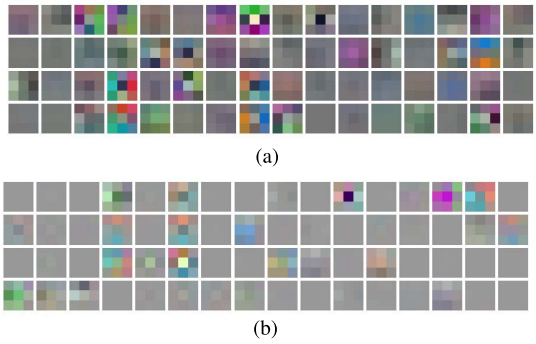
Fig. 5. Layer-wise comparison of rank number and channel number. (a) Channel number. (b) Rank number.

2) Layer-Wise Comparisons: The layer-wise channel numbers and rank numbers of different compressed DNNs are shown in Fig. 5. For the comparison of the channel number in Fig. 5(a), we can see that the proposed method prunes much more channels in deep layers than in shallow layers, which indicates that the last several layers are more redundant. Compared with channel pruning method–Slimming, the method has more sparse channels. Even so, the pruned method with a higher compression rate achieves comparable accuracy. The layer-wise comparison of the rank of weight matrix is also shown in Fig. 5(b), between SVD and the proposed method. It depicts that the ranks of EDP are significantly lower than that of SVD, especially in the middle layers. Hence, the cooperation of the EDP's two components shows the effectiveness to make the weight matrix to be low rank.

3) Visualization: We also give some subjective analysis of the proposed method. Fig. 6 shows the filters of the first convolutional layer of VGG-small trained on CIFAR-10. In particular, the $64\times3\times3\times3$ tensor is represented by 64 RGB images with 3×3 size. Moreover, the values of the filters are normalized to [0, 255]. As shown in Fig. 6, the filters of the proposed method are much sparser compared with the baseline. Since most of the filters are all zero, it is easier

TABLE VII
DETAILED RESULTS OF PARAMETERS AND FLOPS OF VGG-SMALL ON CIFAR-10

Feature map size	Original architecture	Pruned architecture	Compressed (abs. / rel.), %	
			Params	FLOPs
32×32	3 → Conv 3x3 → 64 64 → Conv 3x3 → 64	3 → Conv 3x3 → 33 33 → Conv 3x3 → 35	0.18 / 70.76	8.93 / 70.76
16×16	64 → Conv 3x3 → 128 128 → Conv 3x3 → 128	35 → Conv 3x3 → 70 70 → Conv 3x3 → 70	1.04 / 70.09	12.67 / 70.09
8×8	128 → Conv 3x3 → 256 256 → Conv 3x3 → 256 256 → Conv 3x3 → 256	70 → Conv 3x3 → 140 140 → Conv 3x3 → 141 141 → Conv 3x3 → 141	6.88 / 69.84	21.04 / 69.84
4×4	256 → Conv 3x3 → 512 512 → Conv 3x3 → 512 512 → Conv 3x3 → 512	141 → Conv 3x3 → 164 164 → Conv 3x3 → 50 → Conv 1x1 → 62 62 → Conv 3x3 → 22 → Conv 1x1 → 43	37.39 / 94.94	28.61 / 94.94
2×2	512 → Conv 3x3 → 512 512 → Conv 3x3 → 512 512 → Conv 3x3 → 512	43 → Conv 3x3 → 14 → Conv 1x1 → 31 31 → Conv 3x3 → 9 → Conv 1x1 → 69 69 → Conv 3x3 → 12 → Conv 1x1 → 50	47.14 / 99.76	9.02 / 99.76
1×1	512 → FC → 512 → FC → 10	50 → FC → 512 → FC → 10	1.58 / 88.51	0.04 / 88.51
Total			94.20	80.31



(a) (b)

Fig. 6. Filter visualization results. Note that the filters come from the first layer of VGG-small. (a) Baseline. (b) Proposed method.

for EDP to achieve low rank as well as channel sparsity. We also give a detailed observation of the layer-wise compression results. Table VII shows the details of the compressed VGG-small on CIFAR-10. Note that the absolute compressed ratio in some block is the removed parameter proportion of all network parameters, whereas the relative compressed ratio means the removed parameter proportion of all the block parameters. Compared with the baseline, each layer of the compressed architecture has fewer parameters and FLOPs, which verifies the effectiveness of the compression method. Besides, it can be found that deeper layers have much higher relative compressed ratios of parameters and FLOPs than the shallower layers. This reflects that the redundant parameters are densely distributed in deep layers, which is consistent with the conclusion in [35]. This may mean that the network maintains the diversity and large information of the inputs in the first few convolutional layers.

4) Sensitivity of Hyperparameter Settings: For the hyperparameter (i.e., λ_1 and λ_2) selection of the proposed method, we adopt a "grid-search" approach using the training data, which are tuned exponentially at first and then selected by a fine grid search approach. Especially, for λ_1 and λ_2, the grid search is divided into two steps: one for loose grid search in [0.0001, 0.001, 0.01, 0.1] to find a relatively good FLOPs' accuracy tradeoff, and then, the other for a fine grid search in a small subset near the relatively good point. As shown in Fig. 7, $\lambda_1 = 0.01$ and $\lambda_2 = 0.001$ are selected in the loose grid search stage, and then, we adjust λ_1 in [0.006, 0.007, 0.008, 0.009, 01] and λ_2 in [0.001, 0.002, 0.003, 0.004, 0.005] to choose the optimal hyperparameters. In the loose grid search stage, the hyperparameters are tuned exponentially in a relatively wide range. In the fine grid search stage, we find that the hyperparameters are not sensitive at this range, and the accuracy keeps steady with slight fluctuation. These observations show that the λs are not difficult to be tuned.

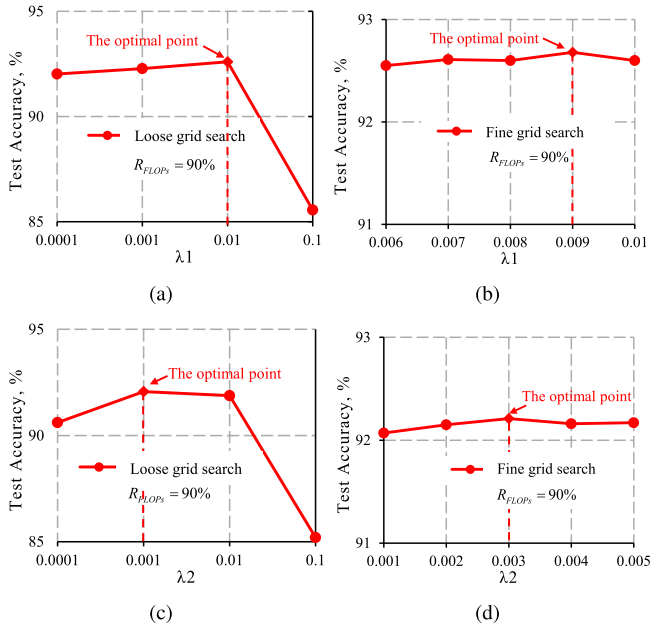


Fig. 7. Test accuracy at different hyperparameter settings. (a) Loose grid search of λ_1. (b) Fine grid search of λ_1. (c) Loose grid search of λ_2. (d) Fine grid search of λ_2.

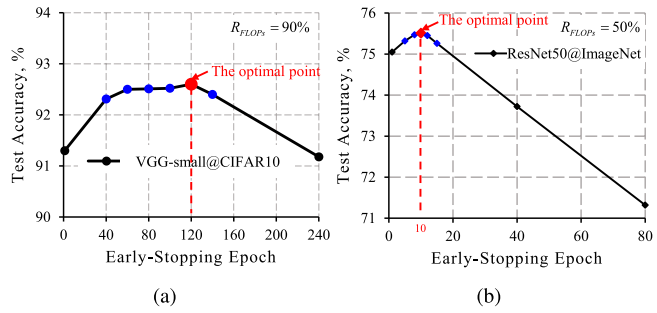


Fig. 8. Sensitivity analysis of ES epoch. (a) Accuracy versus $Epoch_{ES}$ curve with VGG-small on the CIFAR-10 data set at 90% pruned FLOPs rate. (b) Accuracy versus $Epoch_{ES}$ curve with ResNet50 on the ImageNet data set at 50% pruned FLOPs rate.

On the other hand, we also analyze the sensitivity of the ES epoch $Epoch_{ES}$. It is also selected by the "grid-search" approach. In the loose grid search, we search $Epoch_{ES}$ in [1, 120, 240] for CIFAR-10 data set and [1, 40, 80] for ImageNet data set, and then, a fine grid search is conducted in a small subset near a relatively good point. As shown in Fig. 8, $Epoch_{ES} = 120$ (for CIFAR-10) and $Epoch_{ES} = 1$ (for ImageNet) are selected in the loose grid search stage, and then, we adjust $Epoch_{ES}$ in [40, 60, 80, 100, 140] for CIFAR-10 and $Epoch_{ES}$ in [5, 8, 10, 12, 15] for ImageNet to choose the optimal $Epoch_{ES}$. When $Epoch_{ES} = 120$ and 10, the test accuracy reaches the best. Hence, we set $Epoch_{ES}$ to be 120 (for CIFAR-10) and 10 (for ImageNet) in all the experiments. The accuracy at different $Epoch_{ES}$ is shown in Fig. 8. It can be seen that $Epoch_{ES}$ has low sensitivity in the fine grid range and is not difficult to be tuned.

5) Joint Optimization Versus Separate Optimization: The proposed EDP optimizes (7) jointly, with low-rank decomposition component (i.e., the first item of $\Omega(\Theta)$) and channel pruning component (i.e., the second item of $\Omega(\Theta)$). To evaluate the effectiveness of the joint optimization scheme, we compare it with separate optimization. In the experiment of separate optimization, we optimize case ① and case ② in sequence. As shown in Fig. 9, the joint optimization outperforms separate optimization. The test accuracy of joint optimization is 0.5% higher than that of separate optimization at the same pruning rate. Besides, joint optimization has fewer hyperparameters and simpler pruning process, which helps the EDP to be efficient and effective.

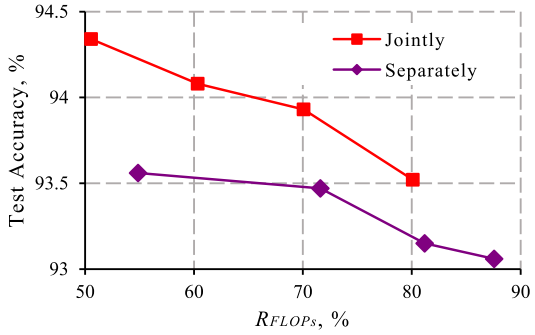


Fig. 9. Accuracy curves at different compression ratios of joint optimization and separate optimization.

TABLE VIII

COMPARISON RESULTS OF ADDING THE REGULARIZATION TO θ AND β WITH VGG SMALL ON THE CIFAR-10 DATA SET. NOTE THAT "OURS (θ)" INDICATES ADDING THE REGULARIZATION TO θ

Method	Accuracy	R_{Params}	R_{FLOPs}	Taining Time
Ours(θ)	92.71%	78.80%	79.98%	18s/epoch
Ours	**93.52%**	**95.59%**	**80.11%**	**15s/epoch**

TABLE IX

TEST ACCURACY AT DIFFERENT PRESET PRUNED RATIOS. IN THE EXPERIMENTS, WE USE VGG-SMALL AND RESNET56 ON THE CIFAR-10 DATA SET AND RESNET50 ON THE IMAGENET DATA SET

Model Architecture @Dataset	R_{FLOPs}	Test Accuracy Top-1	Top-5
VGG-small@CIFAR-10	0	93.60%	
	50%	**94.34%**	
	60%	94.08%	
	70%	93.93%	
	80%	93.52%	
ResNet56@CIFAR-10	0	93.61%	
	50%	**93.79%**	
	60%	93.51%	
	70%	93.14%	
ResNet50@ImageNet	0	75.90%	92.77%
	40%	**75.92%**	**92.83%**
	50%	75.51%	92.54%
	60%	74.48%	92.06%

In addition, in the proposed joint optimization, all the constraints are imposed on β. In order to verify that it is more effective to impose constraints on β instead of θ, we compare the performance of adding regularization to β and θ. As shown in Table VIII, the test accuracy of adding regularizations to θ is 0.81% lower than that of adding regularizations to β. Moreover, the pruning rate and training time are also inferior to the proposed method. Therefore, adding regularizations to β is more effective and efficient than adding regularizations to θ.

TABLE X

TRAINING TIME RESULTS BETWEEN BASELINE AND THE PROPOSED METHOD. IN THE EXPERIMENTS, WE USE VGG-SMALL ON THE CIFAR-10 DATA SET AND RESNET50 ON THE IMAGENET DATA SET. TOTAL TIME↓(%) DENOTES THE TOTAL TIME DROP PERCENTAGE, THE HIGHER THE BETTER

Model Architecture @Dataset	Method	Average Consumption Time $[0;\ Epoch_{ES})$	Average Consumption Time $[Epoch_{ES};\ E)$	Total time↓(%)
VGG-small@CIFAR-10	Baseline	10 s/epoch	10 s/epoch	-10.8%
	Ours	15 s/epoch	8 s/epoch	
ResNet50@ImageNet	Baseline	0.53 h/epoch	0.53 h/epoch	**14.3%**
	Ours	0.63 h/epoch	0.43 h/epoch	

TABLE XI

TRAINING EPOCHS' COMPARISON OF VGG-SMALL ON THE CIFAR-10 DATA SET. WE CONDUCT THE EXPERIMENTS ONLY USING 150 EPOCHS AT DIFFERENT PRUNED RATIOS

Method	Training Epochs Regularization	Training Epochs Finetuning	Total epochs	Accuracy	R_{Params}	R_{FLOPs}
GrOWL [35]	150	50	200	92.20%	91.26%	−−
GrOWL+L_2 [35]	150	50	200	92.70%	93.12%	−−
Li *et al.* [43]	−−	160	160	93.45%	64.00%	34.20%
Slimming [13]	160	160	320	93.48%	86.65%	43.50%
Ours	−−	−−	**150**	93.05%	**93.49%**	**78.17%**
Ours	−−	−−	**150**	**93.51%**	88.13%	70.27%

6) Preset Compressed Ratio: The proposed method adaptively learns a compact model with a proper compressed ratio and high accuracy. When presetting the compressed ratio, EDP can also be deployed and show a good performance in Table IX. Specifically, after obtaining a sparse network, we rank the parameter magnitudes at each layer and prune a certain proportion of parameters that ranked the last. In Table IX, the compressed model even performs better than baseline (i.e., 94.34% versus 93.60% and 93.79%, versus 93.61%). This may be because fewer parameters can avoid over-fitting. At the same pruned FLOPs ratio on CIFAR-10, the test accuracy of ResNet56 decreases more evidently than that of VGG-small, due to Resnet56 more compact than VGG-small. Moreover, the EDP prunes FLOPs to 70% only with the accuracy degradation of less than 0.5%, for VGG-small and ResNet56. On a larger data set (i.e., ImageNet), we prune ResNet50 and obtain better performance than baseline at pruned FLOPs of 40%, which further manifests the effectiveness of the proposed method.

7) Training Complexity Analysis: Although the imposed regularizers may influence the training complexity, the overall framework is efficient. For the training time, we compare the training time per epoch between the method and baseline (the original network architecture). As shown in Table X, baseline costs 0.53 h/epoch during $[0;\ E)$ to train ResNet50 on the ImageNet data set, whereas the method spends 0.63 h/epoch and 0.43 h/epoch during $[0;\ Epoch_{ES})$ and $[Epoch_{ES};\ E)$ epochs, respectively. Although it takes more time for the proposed method during $[0;\ Epoch_{ES})$ epochs, comparable training time costs less at later epochs due to the PM module and ES strategy. Overall, the training time spends only 10.8% hours more than baseline on CIFAR-10 and 14.4% hours less than baseline on the ImageNet data set. This indicates that the training time complexity of the proposed method is comparable with baseline. For the convergence speed, as shown in Table XI, the method spends fewer epochs on training and performs better than others. Therefore, it can be concluded that the proposed training scheme is efficient.

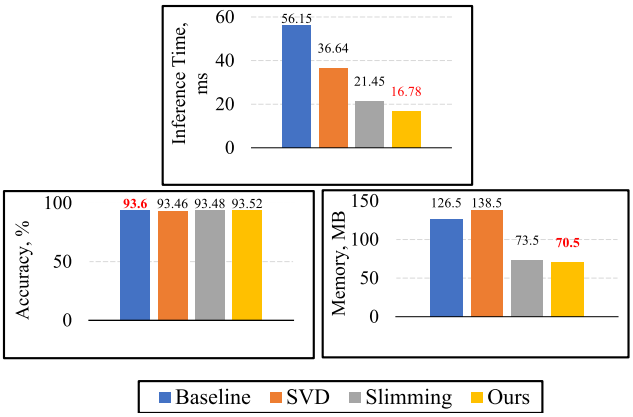


Fig. 10. Inference time and run-time memory comparison using VGG-small on CIFAR-10.

8) Inference Time and Run-Time Memory Consumption: Besides the number of network parameters and FLOPs, we also evaluate the proposed method on inference time and run-time memory. Fig. 10 shows the results of VGG-small on CIFAR-10. For the inference time, Fig. 10 shows that the proposed method accelerates the DNN model by more than 3 times and outperforms all the competing methods. In terms of the run-time memory, it can be found that the run-time memory consumption of the proposed method is the lowest among the competing methods. Moreover, SVD takes 138.50 M CPU memory, even more than the memory occupied by baseline. As SVD is a decomposition method, it generates more layers with feature maps. More feature maps must result in more run-time memory consumption. Fortunately, the channel pruning component decreases the redundant feature maps, overcoming the shortcoming of decomposition.

TABLE XII
OBJECT DETECTION COMPARISON FOR FASTER R-CNN [57] METHOD ON THE PASCAL VOC2007 DATA SET [58]

Method	R_{FLOPs}	Fps [a]	mAP	mAP ↓
Baseline [3]	0	50.4	0.734	−−
Li *et al.* [43]	36.8%	59.8	0.720	0.014
Slimming [13]	37.9%	60.1	0.722	0.012
Ours	**40%**	**60.5**	**0.727**	**0.007**

[a] Fps(images/s) is tested on four NVIDIA GTX 1080 Ti cards.

9) Generalization Ability on Detection Tasks: As described above, the method is effective in image classification. To further analyze the generalization of the method, we conduct experiments with the compressed model. Specially, we use ResNet50 as a backbone network to deploy Faster R-CNN [57] for object detection and then compress Faster R-CNN by reducing 40% FLOPs of the backbone network. In the experimental implementation, we evaluate the performance with inference time (Fps) and mean average precision (mAP) on the PASCAL VOC 2007 data set [58], which consists of about 5K training/validation images and 5K testing images. As shown in Table XII, the pruned model shows a good result. The mAP of the method is slightly lower than the baseline, but the inference speed is faster than the baseline. This demonstrates that the method has good generalization on other tasks.

V. CONCLUSION

In this article, we have proposed an effective decomposition and pruning (EDP) scheme via a compressed-aware block. The block can be used to replace every layer of a CNN in a plug-and-play way so as to efficiently compress the network. EDP takes only one single process to train a DNN and obtain a high compression rate with barely declining accuracy. To be specific, we have embedded the compressed-aware block by decomposing the original network layer into two layers: one represents the weight basis of this layer and the other represents the coefficient matrix. By adding regularizers on the coefficient matrix during training, a low-rank weight basis and sparse channels have been obtained. Moreover, a PM module has been presented to prune redundant channels and merge redundant decomposed layers, which can further compress and optimize the DNN. Experiments have shown that EDP outperforms the state of the arts on pruned ratio, test accuracy, inference time, and run-time memory and is generally applicable to different data sets and networks.

In the future, we will explore the layer pruning for deeper networks. Moreover, we will consider integrating the proposed method with other compression methods (e.g., quantization) in real applications.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

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

[4] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.

[6] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr, "Fully-convolutional siamese networks for object tracking," in *Proc. Eur. Conf. Comput. Vis.*, vol. 2016, pp. 850–865.

[7] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence-video to text," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 4534–4542.

[8] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1269–1277.

[9] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li, "Coordinating filters for faster deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 658–666.

[10] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 10, pp. 1943–1955, Oct. 2016.

[11] S. Lin, R. Ji, C. Chen, and F. Huang, "ESPACE: Accelerating convolutional neural networks via eliminating spatial and channel redundancy," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 1424–1430.

[12] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1389–1397.

[13] 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. (ICCV)*, Oct. 2017, pp. 2736–2744.

[14] 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.

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

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

[17] 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.

[18] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Proc. Adv. Neural Inf. Process. Syst.*, 1992, pp. 950–957.

[19] X. Ding, G. Ding, X. Zhou, Y. Guo, J. Han, and J. Liu, "Global sparse momentum SGD for pruning very deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 6382–6394.

[20] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2018, *arXiv:1803.03635*. [Online]. Available: http://arxiv.org/abs/1803.03635

[21] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," 2018, *arXiv:1810.05270*. [Online]. Available: http://arxiv.org/abs/1810.05270

[22] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," 2018, *arXiv:1810.02340*. [Online]. Available: http://arxiv.org/abs/1810.02340

[23] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[24] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016, *arXiv:1602.02830*. [Online]. Available: http://arxiv.org/abs/1602.02830

[25] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[26] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

[27] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," 2016, *arXiv:1612.01064*. [Online]. Available: http://arxiv.org/abs/1612.01064

[28] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: http://arxiv.org/abs/1510.00149

[29] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," 2014, *arXiv:1405.3866*. [Online]. Available: http://arxiv.org/abs/1405.3866

[30] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, *arXiv:1511.06530*. [Online]. Available: http://arxiv.org/abs/1511.06530

[31] J. M. Alvarez and M. Salzmann, "Compression-aware training of deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 856–867.

[32] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," 2014, *arXiv:1412.6553*. [Online]. Available: http://arxiv.org/abs/1412.6553

[33] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," 2018, *arXiv:1808.06866*. [Online]. Available: http://arxiv.org/abs/1808.06866

[34] G. Huang, S. Liu, L. V. D. Maaten, and K. Q. Weinberger, "CondenseNet: An efficient DenseNet using learned group convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2752–2761.

[35] D. Zhang, H. Wang, M. Figueiredo, and L. Balzano, "Learning to share: Simultaneous parameter tying and sparsification in deep learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–14.

[36] M. Schmidt, N. L. Roux, and F. R. Bach, "Convergence rates of inexact proximal-gradient methods for convex optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 1458–1466.

[37] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imag. Sci.*, vol. 2, no. 1, pp. 183–202, Jan. 2009.

[38] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.

[39] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[40] S. Zagoruyko, "92.45% on cifar-10 in torch," Torch Blog, 2015. [Online]. Available: http://torch.ch/blog/2015/07/30/cifar.html

[41] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*. [Online]. Available: http://arxiv.org/abs/1502.03167

[42] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[43] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Peter Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*. [Online]. Available: http://arxiv.org/abs/1608.08710

[44] C. Jiang, G. Li, C. Qian, and K. Tang, "Efficient DNN neuron pruning by minimizing layer-wise nonlinear reconstruction error," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 2298–2304.

[45] M. Lin *et al.*, "HRank: Filter pruning using high-rank feature map," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1529–1538.

[46] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, and Y. Yang, "Asymptotic soft filter pruning for deep convolutional neural networks," *IEEE Trans. Cybern.*, vol. 50, no. 8, pp. 3594–3604, Aug. 2020.

[47] Z. Zhuang *et al.*, "Discrimination-aware channel pruning for deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 875–886.

[48] 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/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4340–4349.

[49] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: Automl for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 784–800.

[50] Y. Li *et al.*, "Exploiting kernel sparsity and entropy for interpretable CNN compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2800–2809.

[51] A. Paszke *et al.*, "Automatic differentiation in Pytorch," in *Proc. Adv. Neural Inf. Process. Syst. Workshop*, 2017, pp. 1–4. [Online]. Available: pytorch.org

[52] 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. (ICCV)*, Oct. 2017, pp. 5058–5066.

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

[54] S. Lin, R. Ji, Y. Li, C. Deng, and X. Li, "Toward compact ConvNets via structure-sparsity regularized filter pruning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 2, pp. 574–588, Feb. 2020.

[55] X. Ding, G. Ding, Y. Guo, J. Han, and C. Yan, "Approximated oracle filter pruning for destructive CNN width optimization," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1607–1616.

[56] S. Gao, F. Huang, J. Pei, and H. Huang, "Discrete model compression with resource constraint for deep neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1899–1908.

[57] 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.

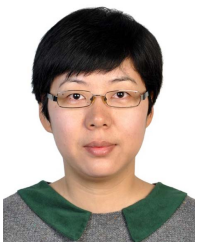
[58] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes challenge: A retrospective," *Int. J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

Xiaofeng Ruan received the B.E. degree in mechanical design, manufacturing, and automation from Dalian Maritime University, Dalian, China, in 2012, and the M.E. degree in aerospace manufacturing engineering from the Harbin Institute of Technology, Harbin, China, in 2014. He is currently pursuing the Ph.D. degree in pattern recognition and intelligent system with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing, China.

His current research interests include pattern recognition, deep learning, and model compression.

Yufan Liu received the B.S. degree from Zhejiang University, Hangzhou, China, in 2015, and the M.S. degree from Beihang University, Beijing, China, in 2018.

She is currently a Research Associate with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing. She has published several articles in international journals and conference proceedings, including the IEEE TRANSACTIONS ON IMAGE PROCESSING, the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), European Conference on Computer Vision (ECCV), and so on. Her research interests include computer vision, model compression, and saliency prediction.



Chunfeng Yuan received the Ph.D. degree from the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing, China, in 2010.

She is currently an Associate Professor with CASIA. Her research interests and publications range from statistics to computer vision, including sparse representation, deep learning, motion analysis, action recognition, and event detection.

Bing Li received the Ph.D. degree from the Department of Computer Science and Engineering, Beijing Jiaotong University, Beijing, China, in 2009.

He is currently a Professor with the Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing. His current research interests include video understanding, color constancy, visual saliency, multiinstance learning, and web content security.

Weiming Hu (Senior Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Engineering, Zhejiang University, Zhejiang, China, in 1998.

From 1998 to 2000, he was a Postdoctoral Research Fellow with the Institute of Computer Science and Technology, Peking University, Beijing, China. He is currently a Professor with the Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing. His research interests are visual motion analysis, recognition of web objectionable information, and network intrusion detection.

Yangxi Li received the Ph.D. degree from Peking University, Beijing, China, in 2012.

He is currently a Senior Engineer with the National Computer Network Emergency Response Technical Team/Coordination Center of China. His research interests lie primarily in multimedia search, information retrieval, and computer vision.

Stephen Maybank (Fellow, IEEE) received the B.A. degree in mathematics from King's College, Cambridge, U.K., in 1976, and the Ph.D. degree in computer science from Birkbeck, University of London, London, U.K., in 1988.

He is currently a Professor with the School of Computer Science and Information Systems, Birkbeck College, University of London. His research interests include the geometry of multiple images, camera calibration, and visual surveillance.