Dictionary Pair-based Data-Free Fast Deep Neural Network Compression

Yangcheng Gao[1,2], Zhao Zhang[1,2,*], Haijun Zhang[3], Mingbo Zhao[4], Yi Yang[5], and Meng Wang[1,2]
[1]School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China
[2]Key Laboratory of Knowledge Engineering with Big Data (Ministry of Education) & Intelligent Interconnected Systems Laboratory of Anhui Province, Hefei University of Technology, Hefei 230009, China
[3]Department of Computer Science, Harbin Institute of Technology (Shenzhen), Xili University Town, Shenzhen, China
[4]Department of Electrical Engineering, City University of Hong Kong, Hong Kong SAR
[5]Centre for Artificial Intelligence, University of Technology Sydney, Sydney, NSW 2007, Australia
*Corresponding author E-mail: cszzhang@gmail.com

2021 IEEE International Conference on Data Mining (ICDM) | 978-1-6654-2398-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICDM51629.2021.00022

***Abstract*—Deep neural network (DNN) compression can reduce the memory footprint of deep networks effectively, so that the deep model can be deployed on the portable devices. However, most of the existing model compression methods cost lots of time, e.g., vector quantization or pruning, which makes them inept to the real-world applications that need fast online computation. In this paper, we therefore explore how to accelerate the model compression process by reducing the computation cost. Then, we propose a new deep model compression method, termed Dictionary Pair-based Data-Free Fast DNN Compression, which aims at reducing the memory consumption of DNNs without extra training and can greatly improve the compression efficiency. Specifically, our proposed method performs tensor decomposition on the DNN model with a fast dictionary pair learning-based reconstruction approach, which can be deployed on different layers (e.g., convolution and fully-connection layers). Given a pre-trained DNN model, we first divide the parameters (i.e., weights) of each layer into a series of partitions for dictionary pair-based fast reconstruction, which can potentially discover more fine-grained information and provide the possibility for parallel model compression. Then, dictionaries of less memory occupation are learned to reconstruct the weights. Extensive experiments on popular DNNs (i.e., VGG-16, ResNet-18 and ResNet-50) showed that our proposed weight compression method can significantly reduce the memory footprint and speed up the compression process, with less performance loss.**

***Index Terms*—Model compression efficiency, dictionary pair-based fast compression of DNNs, fast weight reconstruction, less performance loss.**

I. Introduction

Deep Neural Networks (DNNs)-based models have obtained state-of-the-art performance for a variety of computer vision and image processing applications, due to strong representation ability. However, in practice DNNs usually rely on the development of GPU with high computation capability, since there are millions or even billions of parameters in a DNN model. For example, VGG-16 [1] has a total of 138 million trainable parameters, and it takes two to three weeks to train the VGG model on ImageNet dataset [2] with an Nvidia Titan GPU machine. Thus, due to the large model size and high computational complexity, it is still very challenging to deploy the DNN models on portable devices with limited memory, energy and computation ability for the specific tasks, such as image classification, object detection and image restoration.

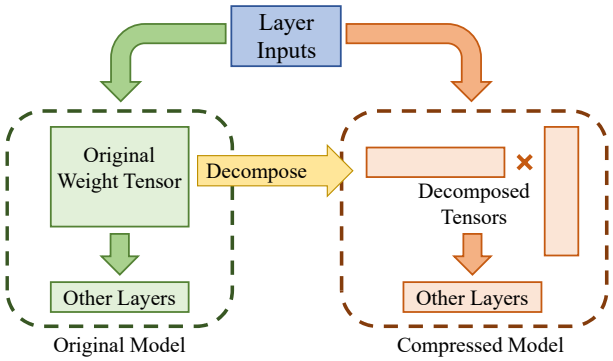

Fig. 1. Tensor decomposition (TD) principle.

To reduce the computational complexity of the deep model and ensure the performance at the same time, researchers have been investigating on the methods of DNN compression, such as vector quantization (VQ) [6], pruning [17] and tensor decomposition (TD) [5]. Specifically, VQ performs similarly as a clustering method like k-means on the weights to group the filters in DNNs. Pruning produces the sparsity in DNNs to reduce the computation and memory access. According to [5], there exists inherent redundancy within DNNs, so that deep model can be compressed through TD. As such, some effective tensor decomposition-based methods have been proposed for DNN compression, such as Singular value decomposition (SVD) [13], QR decomposition [22], CUR [23], Tucker [15], canonical polyadic decomposition (CPD) [10], Tensor Train [24] and Filter Learning [3].

However, existing model compression methods usually did not pay attention to reducing the computational complexity so that the compression process can be speeded up. For example, most existing TD-based methods, especially for the low-rank approximation (LRA) methods [5], need expensive computational process, since the numerous filters in DNNs

978-1-6654-2398-4/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDM51629.2021.00022

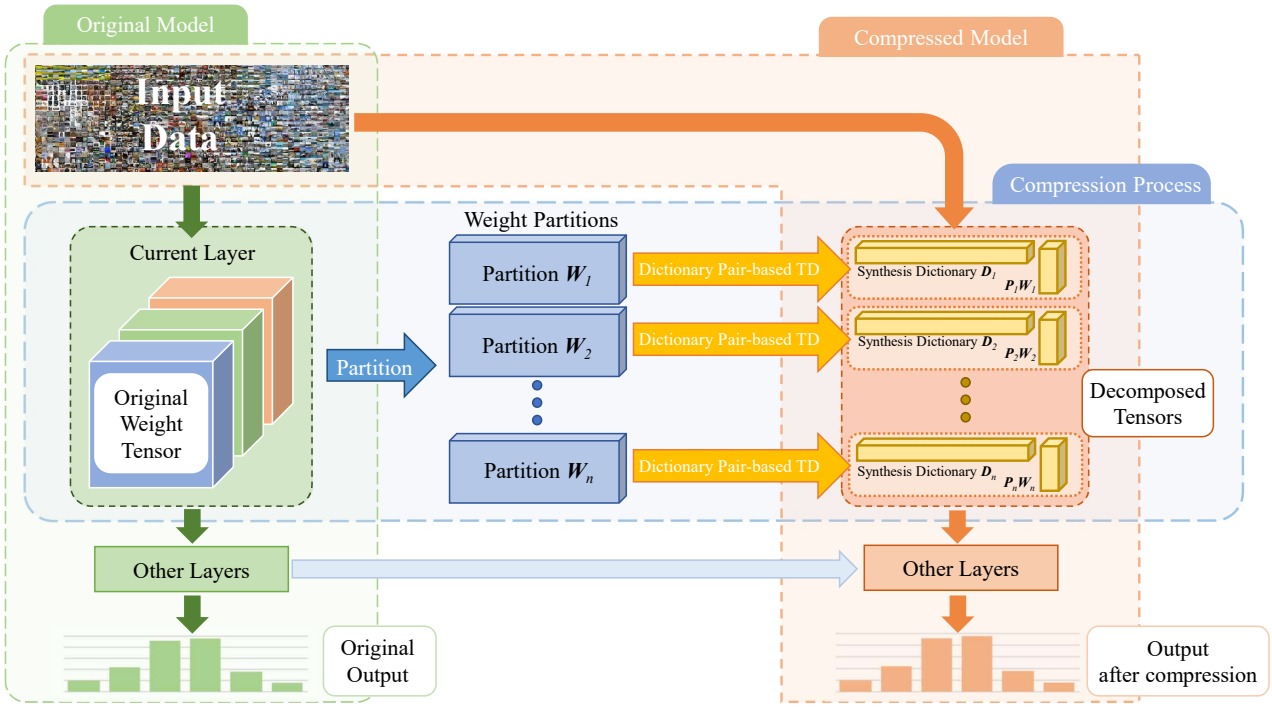

Fig. 2. Whole process of our proposed dictionary pair-based data-free fast DNN compression.

and VQ needs plenty of time and computation resources for clustering. It is also noted that due to the reduced generalization ability after compression, conventional DNN compression methods may need retraining or fine-tuning that takes very long time on large-scale datasets. Besides, existing model compression methods may extremely rely on the original training data. However, in the real-world scenario, original data may be prohibited to be accessed due to some reasons such as privacy protection. As such, we believe a promising model compression method should not only pursue to reduce the memory footprint, but also pay attention to the real needs for fast data-free compression with small amount of calculation. In this paper, we therefore propose a dictionary pair-based fast data-free DNN compression method. Our method aims at finding an efficient tensor decomposition formulated as a dictionary learning problem, without training and with the performance ensured. The main contributions of this paper are summarized as follows:

- Technically, a general data-free fast DNN compression method without training is proposed, which can be deployed on different layers of all existing DNNs. Specifically, our method can obtain significant storage reduction of DNNs, reduce the model compression time and save the computational resources. The whole process of our dictionary pair-based data-free fast DNN compression is illustrated in Fig.2. Given a pre-trained DNN model, our method first divides the parameters in each layer into a series of partitions and then dictionaries of less memory occupation are learned by a fast reconstruction method to approximate and compress the weights.
- A new parameter partition strategy is proposed to divide the weights of DNNs into partitions. With this partition strategy, one can decompose the weight tensors in a more flexible way, since the shape of weight matrix can be defined to adapt to the dictionary pair-based fast reconstruction process. Besides, the partition strategy can also provide the possibility for parallel processing of model compression.
- A dictionary pair-based fast TD approach is presented, which can obtain the robust and compact representation of DNN parameters to improve the performance and model compression efficiency, without the original data. Due to the strong representation ability of the dictionary pair learning-based reconstruction, the tensor decomposition can obtain the dictionaries and representation coefficients with less memory footprint to approximate the weights. As an iterative method with a fast reconstruction in each iteration, our method can significantly reduce the model compression time with the performance ensured, without fine-tuning process.
- Extensive experiments on VGG-16 [1], ResNet-18 [11] and ResNet-50 demonstrated the superior performance of our method in terms of compression time and performance loss. For example, our method can reduce 81.05 % parameters of VGG-16 in terms of Top5 accuracy loss of 0.5 % , within 25 seconds; The Top5 accuracy of ResNet-

18 reaches 88.398 % after our compression within 15 seconds. With a 53.90 % reduction of the model size after the compression within 22.1 seconds, ResNet-50 can also obtain 92.62 % Top5 accuracy.

The rest of this paper is organized as follows. Section II briefly introduces the related work on model compression. In Section III, we present the proposed dictionary-based fast DNN compression method. Section IV evaluates the performance our method in terms of compression time and effectiveness on the related tasks. Finally, the concluding remarks are provided in Section V.

II. Related Work

Model compression plays an important role for the development and applications of DNNs, and many kinds of model compression methods have been proposed for DNNs, such as vector quantization, pruning and tensor decomposition.

A. Vector Quantization (VQ)

VQ [6] performs the model compression similarly as a clustering approach like k-means to group the filters in DNNs so that the filters can be represented as centroids and codebook. VQ can naturally exploit the redundancies among groups of net-work parameters to achieve dramatic compression. For example, an AGB algorithm [6] was proposed for clustering the output activations of ResNet-50 to achieve 20× compression. However, VQ needs much time to learn the codebook and fine-tune the centroids. According to [6], quantizing a ResNet-50 with their AGB method will take about one day on a 16 GB Volta V100 GPU.

B. Pruning

Pruning-based methods produce sparsity in DNNs [25] to reduce the calculation and storage by removing the unimportant components at different levels, e.g., filters [20], connections [7], channels [8] and layers [9]. Note that the drawback of weight and connection pruning is that they both can only produce the sparsity in DNNs, instead of achieving real reduction of memory footprint. As for the channel and layer pruning, they cannot avoid high accuracy loss after model pruning. Besides, pruning methods usually takes long time for model retraining and needs original data.

C. Tensor Decomposition (TD)

TD can decompose the weight tensor of each layer in DNNs into smaller tensors to approximate the original one, so that it can reduce the storage and computational cost. For example, a convolutional filter was proposed as a linear combination of separable 1D filters by dictionary learning [3]; Nonlinear least squares was used to compute the CPD of convolutional filters in DNN [10]; Redundancy of convolutional layer is exploited by SVD-based methods [5], which convert the filter into 3 tensors. However, most of the TD methods will consume large amount computational resources and time.

For the consideration of efficiency, some fast TD-based methods have also been proposed for DNN compression. For example, the approach in [13] found the exact global optimizer of the low-rank decomposition and achieved significant speedup and storage reduction. Another TD-based method based on truncated SVD, named ENC [12], provided a compromise between accuracy and computational complexity, and obtained the optimal rank configuration for kernel decomposition in a short time. However, both of them still require original data for parameter fine-tuning or training from scratch, there-fore they still cannot offer a global model compression speedup on large-scale datasets.

TABLE I
Important Notations Used in the Paper

Notation	Description
$\boldsymbol{W}$	Uncompressed weight tensor
$\boldsymbol{w}_i$	The i-th output channel of $\boldsymbol{W}$
$\boldsymbol{W}_i$	The i-th weight partition of $\boldsymbol{W}$
$\widehat{\boldsymbol{W}}$	Approximation to $\boldsymbol{W}$
$\boldsymbol{X}$	Input of layer
$\boldsymbol{X}_i$	The i-th Input partition corresponding to $\boldsymbol{W}_i$
$\boldsymbol{Y}$	Input of layer
$\{\boldsymbol{D}, \boldsymbol{P}\}$	Dictionary pair
$\boldsymbol{D}$	Synthesis dictionary
$\boldsymbol{D}_i$	The i-th dictionary corresponding to $\boldsymbol{W}_i$
$\boldsymbol{d}_i$	The i-th dictionary word of $\boldsymbol{D}$
$\boldsymbol{P}$	Analysis dictionary
$\boldsymbol{P}_i$	The i-th analysis dictionary corresponding to $\boldsymbol{W}_i$
$\boldsymbol{S}$	Coefficient matrix of the dictionary-based method
$\boldsymbol{s}_i$	The i-th coefficient vector of $\boldsymbol{S}$
$\sigma(\cdot)$	Activation function of layer
B	The batch size of input $\boldsymbol{X}$
C_{in}	Size of input channel
C_{out}	Size of output channel
s	Size of weight partition $\boldsymbol{P}_i$
k	Kernel size of layer
N	Number of dictionary words

III. Dictionary Pair-based Data-Free Fast DNN Compression

In this section, we describe the dictionary pair-based data-free fast DNN compression method in detail.

A. Notation

For clear presentation of our method, we first introduce the important notations used in the paper in Table I.

B. Whole Framework

We first introduce the whole process of our proposed fast DNN compression. In our method, we formulate the tensor decomposition as an optimization problem and represent the original weight tensors using a series of linear combinations learned by a dictionary pair learning-based fast reconstruction process. Specifically, for a given pre-trained DNN model, the original weight tensor of each layer will be efficiently computed as the linear combinations of smaller tensors for compression. It is noteworthy that there is no requirement for fine-tuning or retraining in our method, while these are needed in the conventional model compression methods.

As shown in Fig.2, our proposed fast DNN compression method works on the weights of the pre-trained deep model

layer by layer. The process of our method can be summarized into three steps, i.e., partitioning weight tensors, decomposing them into small ones and approximating the original ones. While decomposing the weight tensor within each layer without extra operations, i.e., fine-tuning or retraining, it can save lots of compression time. Besides, original data are not required in our method, which may be restricted by the privacy protection and access restrictions. In what follows, we describe the detailed steps of our method:

1) **Partition of parameter.** Different from existing approaches that consider the whole weight tensor as a decomposition object, we take the weight partitions as the objects for dictionary pair-based fast reconstruction. Specifically, after extracting the weight tensor from current layer of the pre-trained DNN model, we divide them into a series of weight partitions, which will be detailed shortly. This strategy can deal with the problem in a more fine-grained way and provide the possibility for parallel computing of parameters.
2) **Tensor decomposition on partitions.** This process is the most critical part to the performance of our compression method. Based on a dictionary pair-based fast decomposition process on the partitions, we obtain the dictionaries and coding coefficients to approximate the original weight partitions. In other words, the knowledge in the weights learned by DNN model is transferred into the learned dictionaries and coefficients, so that the weight tensors can be linearly represented with the words in the learned dictionary. Since the dictionaries and coefficients occupy less memory footprint than the original weight partitions, the weight storage of current layer can be reduced.
3) **Iterate to step 1 to partition and decompose the weight tensors of the next layer.** For conventional model compression methods, parameter fine-tuning is usually a necessary process to improve the generalization ability damaged by parameter reduction. To pursue a fast data-free model compression with low computational cost, we discard the fine-tuning or model retraining process, and propose a dictionary pair-based reconstruction method for the partitioned tensor weights.

C. Partition Strategy

The proposed partition strategy divides the weights of DNNs into partitions for tensor decomposition. We illustrate the comparison of the conventional tensor decomposition and the tensor decomposition with our partition strategy in Fig.3.

Given a weight tensor, conventional TD methods directly decompose it into smaller tensors to approximate the original one. In our dictionary pair-based TD process, we choose a partition size and divide the weight matrix into partitions before decomposition. Then, we decompose the weight partitions into smaller tensors to reconstruct them. Note that if we set the partition size to be equal to the size of input channel, we will take the weight matrix of the current layer as a whole. Otherwise, if we set the partition size to 1, the weight matrix will be divided into a column vector. That is, the partition size should be carefully chosen, which will be discussed in the following experiments.

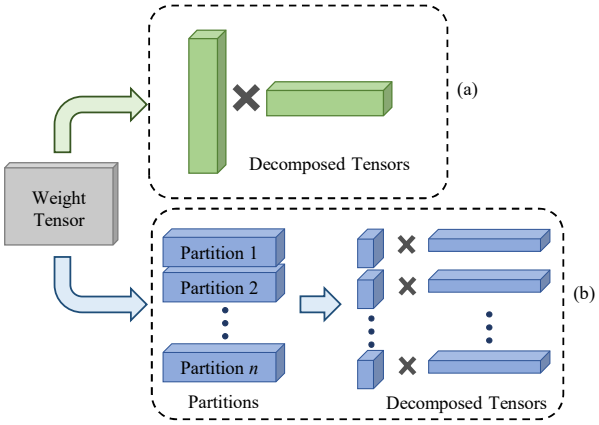

Fig. 3. Comparison of (a) conventional tensor decomposition and (b) tensor decomposition with our partition strategy.

Based on the partition strategy, the shape of partitions can also be flexibly defined to adapt to the dictionary pair-based fast reconstruction process that mainly executes the fine-grained decomposition on the weight partitions. This will potentially lead to better compression results. That is, the partition size will decide the dimension of the learned dictionaries to affect the performance of our method. We will demonstrate the positive effects of the partition strategy later in experiments. Besides, due to the independence of each partition, the partition strategy provides the possibilities for parallel computing, which means that there exists the potential of further improvement. But the parallel computing problem is clearly beyond the scope of this work. Although the partition strategy can be performed on the weight tensor of various layers, it may work better on the fully-connected layers than convolutional layers. Because the partition edges in the convolutional layer may destroy the correlations between adjacent filters which share the same pixels of the input. Thus, we need to choose a proper partition size to avoid this problem.

D. Dictionary Pair-based Fast TD

The goal of the dictionary-based TD is to approximate the original weights of each layer in the DNN model by a dictionary pair learning-based fast reconstruction process.

Given a weight tensor $\boldsymbol{W} \in \mathbb{R}^{C_{in} \times C_{out}}$, dictionary-based TD aims at calculating a dictionary $\boldsymbol{D} \in \mathbb{R}^{C_{in} \times N}$ with words and a set of coefficient vectors $\boldsymbol{s}_i \in \mathbb{R}^{C_{in}}$. Then, each output channel $\boldsymbol{w}_i \in \mathbb{R}^{C_{in}}$ can be approximately represented as $\boldsymbol{w}_i = \boldsymbol{D}\boldsymbol{s}_i$ so that output $\boldsymbol{W}$ is represented by $\boldsymbol{W} = \boldsymbol{D}\boldsymbol{S}$. To learn $\boldsymbol{D}$, a general optimization problem can be defined as follows:

$$\{\boldsymbol{D}, \boldsymbol{S}\} = \arg\min_{\boldsymbol{D},\boldsymbol{S}} \|\boldsymbol{W} - \boldsymbol{D}\boldsymbol{S}\|_F^2 + \lambda \|\boldsymbol{S}\|_p \qquad (1)$$

where $\|\boldsymbol{S}\|_p$ denotes the l_p-norm sparsity constraint on $\boldsymbol{S} \in \mathbb{R}^{N \times C_{out}}$. Conventional l_0 or l_1-norm sparsity constrained dictionary learning methods achieve the sparse representation by learning a synthesis dictionary $\boldsymbol{D}$ and obtaining the sparse coding coefficients $\boldsymbol{S}$.

However, the l_0 or l_1-norm sparsity constraint will make the dictionary-based TD method inefficient in the training phase. Therefore, we use an efficient dictionary pair-based TD method with a group sparse representation ability, inspired by the projective dictionary pair learning (DPL) [4], so that the knowledge in DNNs can be preserved in the learned dictionaries with low reconstruction loss in a short time, due to the fact that costly l_0 or l_1-norm sparsity constraint is avoided.

Different from conventional dictionary learning (1), our unsupervised dictionary pair-based TD method learns a dictionary pair, i.e., a synthesis dictionary $\boldsymbol{D} \in \mathbb{R}^{C_{in} \times N}$ and an analysis dictionary $\boldsymbol{P} \in \mathbb{R}^{N \times C_{in}}$, to achieve the goal of weight approximation. More importantly, the dictionary pair-based TD method can not only greatly save the compression time, but also lead to competitive weight tensor approximation performance. So, we use it to learn dictionaries to approximate the original weight tensors in DNNs and achieve a fast TD. It should be noted that our proposed dictionary pair-based fast reconstruction is different from [4] in twofold. First, their learning mechanisms are different. Specifically, the DPL in [4] is a fully-supervised class-specific method, while our dictionary pair-based fast reconstruction method is unsupervised. Second, their motivations and purposes are different. That is, DPL is mainly proposed for data classification, while our method is proposed for DNN compression.

By learning an analysis dictionary $\boldsymbol{P}$ to obtain the approximated coefficient matrix via embedding, dictionary pair-based reconstruction error minimization and objective function can be defined as follows:

$$\{\boldsymbol{D}, \boldsymbol{P}\} = \arg\min_{\boldsymbol{D},\boldsymbol{P}} \|\boldsymbol{W} - \boldsymbol{DPW}\|_F^2, s.t. \|\boldsymbol{d}_i\|_2^2 \leq 1, \quad (2)$$

where the constraint on $\boldsymbol{d}_i$ provides the stability for the dictionary pair learning process. The analysis dictionary $\boldsymbol{P}$ makes it possible to compute the coefficients directly with original weights.

For optimization, we introduce a variable matrix $\boldsymbol{A} \in \mathbb{R}^{N \times C_{out}}$ and relax the problem in Eq.(2) as follows:

$$\{\boldsymbol{D}, \boldsymbol{P}, \boldsymbol{A}\} = \arg\min_{\boldsymbol{D},\boldsymbol{P},\boldsymbol{A}} \|\boldsymbol{W} - \boldsymbol{DA}\|_F^2 + \tau \|\boldsymbol{PW} - \boldsymbol{A}\|_F^2,$$

$$s.t. \|\boldsymbol{d}_i\|_2^2 \leq 1, \quad (3)$$

where τ is a scalar constant. At the beginning, we perform random initialization on both $\boldsymbol{D}$ and $\boldsymbol{P}$ with the unit Frobenius norm, then $\boldsymbol{A}$ and $\{\boldsymbol{D}, \boldsymbol{P}\}$ can be updated alternatively as follows:

1) Fix $\{\boldsymbol{D}, \boldsymbol{P}\}$, update $\boldsymbol{A}$:

$$\boldsymbol{A} = (\boldsymbol{D}^T\boldsymbol{D} + \tau\boldsymbol{I})^{-1}(\tau\boldsymbol{PW} + \boldsymbol{D}^T\boldsymbol{W}). \quad (4)$$

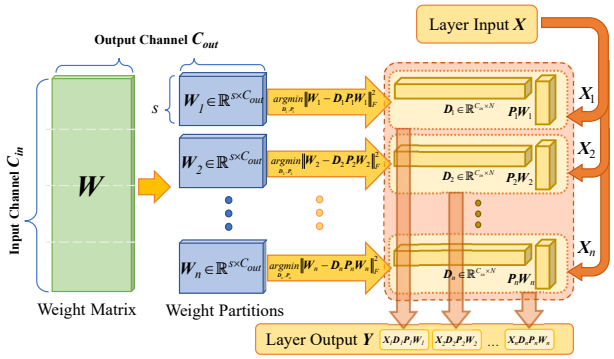

Fig. 4. Dictionary pair based fast compression on a fully-connected layer.

2) Fix $\boldsymbol{A}$, update $\{\boldsymbol{D}, \boldsymbol{P}\}$:

$$\boldsymbol{P} = \tau\boldsymbol{AW}^T(\tau\boldsymbol{WW}^T + \gamma\boldsymbol{I})^{-1}, \quad (5)$$

where γ is a small number to prevent $\boldsymbol{WW}^T$ being non-invertible during the iteration.

For $\boldsymbol{D}$, we can introduce a variable $\boldsymbol{Q}$ to obtain the solution:

$$\begin{cases} \boldsymbol{D}^{(l+1)} = \arg\min_{\boldsymbol{D}} \|\boldsymbol{W} - \boldsymbol{DA}\|_F^2 + \rho \left\|\boldsymbol{D}^{(l+1)} - \boldsymbol{Q} + \boldsymbol{T}\right\|_F^2, \\ \boldsymbol{Q}^{(l+1)} = \arg\min_{\boldsymbol{S}} \rho \left\|\boldsymbol{D}^{(l+1)} - \boldsymbol{Q} + \boldsymbol{T}\right\|_F^2, s.t. \|\boldsymbol{q}_i\|_2^2 \leq 1, \\ \boldsymbol{T}^{(l+1)} = \boldsymbol{T}^l + \boldsymbol{D}^{(l+1)} - \boldsymbol{Q}^{(l+1)}, update\ \rho. \end{cases} \quad (6)$$

The optimization process usually converges rapidly, with the iteration stopping when the difference between the energy in two adjacent iterations is less than 0.01.

By performing the fast reconstruction on the partitioned weight tensors without original data, our fast compression method can not only lead to a competitive TD result, but also reduce the computational cost. The memory footprints of the dictionaries and coefficients are also less than those of the original weights. The storage of DNN models can then be significantly reduced.

E. Fast Compression on Different Layers

It is noteworthy that our compression method is general, which can work on both the fully-connected and convolutional layers, so that we can achieve the global parameter compression for DNN model. In what follows, we will introduce the details on how to do the compression on fully-connected and convolutional layers.

1) Implementation on fully-connected layers: We first show the compression method on fully-connected layers, in which the weights can be regarded as a two-dimensional matrix $\boldsymbol{W} \in \mathbb{R}^{C_{in} \times C_{out}}$ in each layer. Note that we omit the bias in all layers, since it usually occupies small storage and has small impact to the re-construction. Fig. 4 illustates the dictionary pair based fast compression on a fully-connected layer. With the partition strategy, we dividing $\boldsymbol{W}$ into a series of weight partitions $\boldsymbol{W}_i \in \mathbb{R}^{s \times C_{out}}$, we can take each $\boldsymbol{W}_i$ as the object

for dictionary pair-based reconstruction over each partition. Specifically, considering the number of dictionary words N, we perform the fast dictionary pair learning on the partitions to decompose them. Then, each weight partition can be linearly approximated by $\boldsymbol{W}_i \approx \boldsymbol{D}_i\boldsymbol{P}_i\boldsymbol{W}_i$. That is, we obtain a series of dictionaries and coefficients to approximate the partitions.

$$\{\boldsymbol{D}_i, \boldsymbol{P}_i\} = \arg\min_{\boldsymbol{D}_i,\boldsymbol{P}_i} \|\boldsymbol{W}_i - \boldsymbol{D}_i\boldsymbol{P}_i\boldsymbol{W}_i\|_F^2, s.t. \|\boldsymbol{d}_j\|_2^2 \leq 1. \tag{7}$$

In a general fully-connected layer, the output $\boldsymbol{Y} \in \mathbb{R}^{C_B \times C_{out}}$ can be simply formulated as follows:

$$\boldsymbol{Y} = \sigma(\boldsymbol{X}\boldsymbol{W}). \tag{8}$$

Then, we can obtain the output $\boldsymbol{Y}$. After dividing the weight matrix into a series of partitions, with the linear representation of $\boldsymbol{W}_i = \boldsymbol{D}_i\boldsymbol{P}_i\boldsymbol{W}_i$, the new output $\boldsymbol{Y}$ of the layer based on the weight partitions can be approximated as

$$\begin{aligned}\boldsymbol{Y} &= \sigma([\boldsymbol{X}_1\boldsymbol{W}_1, \boldsymbol{X}_2\boldsymbol{W}_2, \ldots, \boldsymbol{X}_n\boldsymbol{W}_n]) \\ &= \sigma([\boldsymbol{X}_1\boldsymbol{D}_1\boldsymbol{P}_1\boldsymbol{W}_1, \boldsymbol{X}_2\boldsymbol{D}_1\boldsymbol{P}_1\boldsymbol{W}_2, \ldots, \boldsymbol{X}_n\boldsymbol{D}_1\boldsymbol{P}_1\boldsymbol{W}_n]).\end{aligned} \tag{9}$$

Since the partition size s is fixed on each layer, the reduction ratio on the weights of each layer α_{layer} can be computed as

$$\alpha_{layer} = 1 - \frac{N(s + C_{out})}{sC_{out}}. \tag{10}$$

Assuming that α_i denotes the reduction ratio of the i-th layer, the global reduction ratio α on the weights of a DNN model can be defined as follows:

$$\alpha = \sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} \left(1 - \frac{N_i(s_i + C_{i_{out}})}{s_i C_{i_{out}}}\right). \tag{11}$$

Note that the size of output channel is decided by the given layer in DNN model, and we can decide the reduction ratio by setting the partition size and number of dictionary words.

2) Implementation on convolutional layers: We then show the compression method on convolutional layers. Although the weight tensors of convolutional layers contain four-dimensions, we can also apply our fast TD approach on it. In each convolutional layer, the weight tensor can be represented as $\boldsymbol{W} \in \mathbb{R}^{C_{in} \times C_{out} \times k \times k}$, which has C_{out} filters of size $k \times k \times C_{in}$. It is naturally to understand that adjacent filters should be highly correlated, since they share the same pixels and have similar information. Moreover, the high correlation between adjacent filters makes it possible to exploit the redundancy among the filters.

For better discovering and utilizing the correlation between adjacent filters, we unfold the weight tensor into a two-dimensional matrix denoted by $\boldsymbol{W}_e \in \mathbb{R}^{C_{out} \times (C_{in} \times k \times k)}$. That is, the c^{th} filter of the original weight matrix is expanded to the row c of partition. To approximate the convolution, we reshape the input $\boldsymbol{X}$ to $\boldsymbol{X}_e$ in a similar way and the result of convolution $\boldsymbol{X} * \boldsymbol{W}$ can be replaced by $\boldsymbol{X}_e\boldsymbol{W}_e$.That is, it is a dual process to the one that uses the bi-level Toeplitz matrices to represent the filters [6]. Then, we can apply the proposed partition strategy on $\boldsymbol{W}_e$ and obtain the weight partitions $\boldsymbol{W}_{ei} \in \mathbb{R}^{s \times (C_{in} \times k \times k)}$ for the dictionary pair-based fast reconstruction learning as explained in Section III-E1 to decompose $\boldsymbol{W}_{ei}$ into $\boldsymbol{D}_{ei}$ and $\boldsymbol{P}_{ei}\boldsymbol{W}_{ei}$.

Different from the fully-connected layer, on convolutional layers, we should pay more attention to the partition size . If we choose a small partition size, the correlation of in-partition filters could be destroyed by the edges of the partitions. If the partition size is too large, the dictionaries learned may not hold enough information to represent the filters. Therefore, we have to tradeoff the representation performance of our dictionary pair-based method and the correlation of in-partition filters to choose the partition size in reality.

The number of dictionary words or atoms is similar to the rank of those LRA methods to compromise between accuracy loss and compressed model size. According to the experiments, we tend to set the number of dictionary words to be relatively larger for the convolutional layers than fully-connected layers, since there usually exists more redundancy on the fully-connected layers. Another reason is that the construction error in current layer will be delivered to the next layer, and amplified at the end.

TABLE II
PRE-TRAINED MODEL INFORMATION

Dataset	Model	Size (MB)	Top1 (%)	Top5 (%)
ImageNet	VGG-16	527.82	73.360	91.516
	ResNet-18	44.59	69.758	89.078
	ResNet-50	97.49	76.130	92.862
CIFAR-10	ResNet-20	1.03	91.48	99.65
	ResNet-56	3.25	93.27	99.70

TABLE III
COMPARISON OF MODEL COMPRESSION TIME

Model	Approach	Time ↓	Finetuning?
ResNet-18	ABG [6]	2h 26min	No
		-	Yes
	Low-rank [13]	28.0s	No
	DCP [14]	250h	Yes
	Ours	**12.0s**	**No**
ResNet-50	ABG [6]	16h 30min	No
		160h 37min	Yes
	Low-rank [13]	56.86s	No
	Hrank [28]	820h	Yes
	Ours	**22.1s**	**No**
VGG-16	ENC-Inf [12]	642.5s	No
	Low-rank [13]	29.16s	No
	Ours	**21.3s**	**No**

IV. EXPERIMENTS

In this section, we evaluate our compression method by comprehensive experiments, and illustrate the comparison results with other related methods. Specifically, we first compare the compression time of each DNN model compression method to evaluate the advantage of computational complexity. Then, we report the performance of each method to show the generalization performance. Finally, we perform the ablation

TABLE IV
PERFORMANCE OF VGG-16 ON IMAGENET

Method	Compressed size (MB)	Reduction Ratio ↑	Top1 Accuracy			Top5 Accuracy		
			Baseline	Compressed	Loss	Baseline	Compressed	Loss
Low-rank [13]	201.09	61.90 %	-	-	-	90.60 %	90.31 %	0.29 %
Tucker [15]	507.34	3.88 %	-	-	-	91.52 %	89.40 %	2.12 %
APG [16]	248.90	52.84 %	70.60 %	68.53 %	2.07 %	89.90 %	88.20 %	1.70 %
SVD [19]	137.22	74.00 %	-	-	-	89.90 %	90.00 %	-0.10 %
APoZ [21] with fine-tuning	213.51	59.55 %	68.36 %	70.44 %	-2.08 %	88.44 %	89.79 %	-1.35 %
APoZ [21] without fine-tuning	213.51	59.55 %	68.36 %	69.29 %	-0.93 %	88.44 %	89.07 %	-0.63 %
PCP [27]	110.60	79.05 %	-	-	-	91.52 %	89.00 %	2.52 %
Ours	**100.00**	**81.05** %	**73.36** %	**71.63** %	**1.73** %	**91.52** %	**90.91** %	**0.61** %

TABLE V
PERFORMANCE OF RESNET-18 ON IMAGENET

Method	Compressed size (MB)	Reduction Ratio ↑	Top1 Accuracy			Top5 Accuracy		
			Baseline	Compressed	Loss	Baseline	Compressed	Loss
DCP [14]	15.60	65.01 %	-	64.11 %	-	-	85.78 %	-
SFP [18]	31.21	30.00 %	70.28 %	67.10 %	3.18 %	89.63 %	87.78 %	1.85 %
Ours	**14.90**	**66.58** %	**69.76** %	**68.57** %	**1.18** %	**89.08** %	**88.40** %	**0.68** %

TABLE VI
PERFORMANCE OF RESNET-50 ON IMAGENET

Method	Compressed size (MB)	Reduction Ratio ↑	Top1 Accuracy			Top5 Accuracy		
			Baseline	Compressed	Loss	Baseline	Compressed	Loss
NISP-B [17]	54.77	43.82 %	72.88 %	71.99 %	0.89 %	-	-	-
SFP [18] with fine-tuning	158.35	30.00 %	76.15 %	74.61 %	1.54 %	92.87 %	92.06 %	0.81 %
SFP [18] without fine-tuning	158.35	30.00 %	76.15 %	62.14 %	14.01 %	92.87 %	84.60 %	8.27 %
ThiNet [20]	47.23	51.56 %	72.88 %	71.01 %	1.87 %	91.14 %	90.02 %	1.12 %
SSR-L2 [36]	46.54	52.26 %	72.88 %	72.16 %	0.72 %	91.14 %	90.85 %	0.29 %
PCP [27]	57.72	40.80 %	74.30 %	73.40 %	0.90 %	92.10 %	91.50 %	0.60 %
EDP [30]	54.69	43.90 %	75.90 %	75.34 %	0.56 %	92.77 %	92.43 %	0.34 %
H-rank [28]	54.60	44.00 %	76.15 %	74.98 %	1.17 %	92.87 %	92.33 %	0.54 %
DCP [14]	47.23	51.55 %	76.01 %	74.99 %	1.02 %	92.93 %	92.20 %	0.73 %
Ours	**44.94**	**53.90** %	**76.13** %	**75.48** %	**0.65** %	**92.86** %	**92.62** %	**0.24** %

TABLE VII
PERFORMANCE OF RESNET-20 ON CIFAR-10

Method	Reduction Ratio ↑	Top1 Accuracy		
		Baseline	Compressed	Loss
SFP [18]	30.00 %	92.2 0%	90.83 %	1.37 %
VCNP [32]	38.00 %	92.01 %	91.66 %	0.35 %
FPGM [37]	40.00 %	92.20 %	90.62 %	1.58 %
Ours	**44.28** %	**91.48** %	**91.55** %	**-0.07** %

TABLE VIII
PERFORMANCE OF RESNET-56 ON CIFAR-10

Method	Reduction Ratio ↑	Top1 Accuracy		
		Baseline	Compressed	Loss
NISP [17]	42.60 %	93.04 %	93.01 %	0.03 %
PFEC [31]	13.70 %	93.06 %	93.04 %	0.02 %
VCNP [32]	45.00 %	93.04 %	92.26 %	0.78 %
KSE [33]	54.73 %	93.03 %	93.23 %	-0.20 %
EDP [30]	45.82 %	93.61 %	93.61 %	0.00 %
H-rank [28]	42.35 %	93.26 %	93.17 %	0.09 %
SFP [18]	40.00 %	93.59 %	93.3 5%	0.24 %
FPGM [37]	40.00 %	93.59 %	93.49 %	0.10 %
Ours	**44.60 %**	**93.27 %**	**93.27 %**	**0.00 %**

study to demonstrate the effects of the dictionary pair-based reconstruction and the partition strategy.

A. Dataset and Experimental Setting

To demonstrate the effectiveness and efficiency of our compression method, we conduct extensive experiments on full ImageNet ILSCVR-12 validation dataset [2] and CIFAR-10 dataset [34]. We evaluate and compare the performance of our dictionary pair-based TD method with others on five popular DNN models, i.e., VGG-16 [1], ResNet-18, ResNet-50, ResNet-20 and ResNet-56 [11]. For different DNN models, we will use different partition strategies and numbers of dictionary words. All of the dictionaries and coefficient matrices are stored in float16 for further compression. The original model accuracy evaluated by the experiments and the original model size are described in Table II.

All the experiments are conducted within the PyTorch framework [29] on a NVIDIA GeForce GTX 1080Ti GPU and an Intel (R) Core (TM) i5-7500 3.40 GHz CPU with 8GB memory. All of the pre-trained models evaluated on ImageNet are provided by the PyTorch model zoo [35]. Our source code is released on GitHub: https://github.com/DiamondSheep/Quantize_DPL.

B. Compression Speed

One of the most significant advantage of our proposed fast compression method is its compression efficiency and low computational cost in the compression process, due to avoiding the time-consuming fine-tuning or retraining process. As such, we would like to compare the compression speed of each method on one machine with the same configurations for the fair comparison.

To demonstrate the efficiency of our scheme, we compare the compression time with five related works, i.e., ABG [6], ENC-Inf [12], Low-rank [13], Hrank [28] and DCP [14]. The results of compression time are described in Table III. We see clearly that our scheme offers a huge advantage in terms of compression speed. Specifically, our compression method finishes the weight compression on all the DNN models within 30 seconds, which is much faster than other compared methods, especially for those with fine-tuning. The Low-rank compression method [13] follows us, whose compression time is also promising. However, it can only compress the convolutional layers, so it cannot achieve a global compression for DNN models. In contrast, our dictionary pair-based fast method can be used to compress the weights of both convolutional layers and fully-connected layers in a short time.

C. Performance Comparison of DNN Models with Weight Compression

In this study, we evaluate the performance of the existing VGG-16 and ResNet models on the ImageNet and CIFAR-10 datasets, in terms of reduced model size and accuracy loss.

1) VGG-16 on ImageNet: The deep model of VGG-16 contains 13 convolutional layers and 3 fully-connected layers, and occupies 527.82MB on device. We compare the performance of our compression method with the following related approaches, including Low-rank [13], Tucker [15], APG [16], SVD [19], APoZ [21] and PCP [27]. The performance comparison results are described in Table V. We see that VGG-16 model deployed with our compression method performs the best among all the compared meth-ods, which reduces the VGG-16 memory footprint to 100MB with 1.73 % top1 accuracy loss and 0.6 % top5 accuracy loss. It is also noteworthy that our method achieves the model reduction and maintains the performance of DNN models without original data in a short time, which demonstrates that our dictionary pair-based data-free method can better preserve the knowledge in DNN models. In other words, our dictionary pair-based fast TD method has a remarkable ability to exploit the redundancy in existing DNN models.

2) ResNet on ImageNet: The ResNet architecture [11] utilized shortcuts to avoid vanish-ing gradients and simplify the model. In this study, we explore the performance of each compression method on both ResNet-18 and ResNet-50 models with well-designed architectures. Specifically, ResNet-18 has one convolutional layer, one fully-connected layer and eight residual blocks including two convolutional layers with a shortcut. ResNet-50 of bottleneck architectures is built with one convolutional layer, one fully-connected layer and 16 bottleneck blocks consisting of three convolutional layers with a shortcut. Different from VGG-16, ResNet models are more compact and harder to be compressed, since there exists less redundancy inside. Besides, the weights of the fully-connected layer in ResNet are less than those of conventional DNN models. As such, it will be more challenging to decompose the weight tensors of those ResNet-based deep models, while preserving their performance at the same time.

In this study, we compare the performance of our compression method with the following several related compression approaches: NISP [17], SFP [18], ThiNet [20], SSR-L2 [36], PCP [27], EDP [30], H-rank [28] and DCP [14]. The experimental results are described in Tables IV and V. We can see that: 1) Our proposed data-free compression method performs the best among all the competitors; 2) For the ResNet-18 model, we can reduce the model size to 14.9MB with 1.18 % top1 accuracy loss and 0.68 % top5 accuracy loss; 3) For the ResNet-50 model, we reduced 53.90 % model storage with 0.65 % top1 accuracy loss and 0.24 % top5 accuracy loss.

3) ResNet on CIFAR-10: For CIFAR-10 dataset, we evaluate two DNN models, i.e., ResNet-20 and ResNet-56, deployed with different model compression methods. We compare the performance of our data-free model compression method with NISP [17], PFEC [31], VCNP [32], KSE [33], EDP [30], H-rank [28], SFP [18] and FPGM [37], in this study. The comparison results in terms of reduced model size and accuracy loss are described in Tables VII and VIII, from which we can observe that our method can still deliver better performance than other compared methods based on both ResNet-20 and ResNet-56. Specifically, our method can reduce 53.90 % storage of the original model with top1 accuracy 91.55 % for ResNet-20, and the compressed model obtains higher accuracy than the original one. For ResNet-56, we achieve a storage reduction of 44.60 % with no accuracy loss. In summary, compared with other competitors, our method achieves the same performance with low computation, and without extra data and fine-tuning process.

D. Ablation Study

The dictionary pair-based fast reconstruction and the partition strategy are two most core contributions of this paper, so we would like to investigate their effects respectively.

1) Effect of dictionary pair-based reconstruction: We first conduct the ablation studies on the effects of the analysis dictionary $\boldsymbol{P}$ by comparing the results with the orignal dictionary-based TD method without $\boldsymbol{P}$. That is, we directly compute a synthesis dictionary $\boldsymbol{D}$ and a coefficient matrix $\boldsymbol{S}$, by replacing

TABLE IX
COMPARISON RESULTS BASED ON RESNET-18

Method	Compressed Size (MB)	Top1 Accuracy	Top5 Accuracy
Ours	**14.9**	**68.57 %**	**88.40 %**
Without $\boldsymbol{P}$	14.9	56.48 %	82.35 %

the term $\boldsymbol{P}$ with $\boldsymbol{S}$ in Eq. (2). For the fair comparison, we take the ResNet-18 model as an example, and evaluate it based on ImageNet dataset with the same configurations, e.g., sparsity constraint, partition size, number of dictionary words and setting of dictionary learning.

For the dictionary-based reconstruction without $\boldsymbol{P}$, we initialize the dictionary $\boldsymbol{D}$ with the original weight partitions rather than random matrix, since we experimentally observe that if we initialize $\boldsymbol{D}$ randomly as our method, it will not reconstruct the weight tensors and the performance of compressed model will be rather bad.

The comparison results of the dictionary-based reconstruction and dictionary pair-based reconstruction are described in Table IX. With the same compressed model size, our method outperforms the one without analysis dictionary in terms of accuracy. Note that this also demonstrates that our dictionary pair-based reconstruction method can preserve more knowledge in DNN models, implying that the approximated coefficient matrix by embedding contains more useful information of the weight partitions.

2) Effect of weight partitions: In this study, we conduct an ablation study to evaluate the effects of the partition strategy, as well as the damage caused by the partitions to the correlation of convolutional filters, we con-duct the ablation study. Specifically, we perform our dictionary pair-based method with different partition size and numbers of dictionary words for the convolutional layers of ResNet-18 model. The comparison results are presented in Fig.5.

From the curves in Fig.5, we find the negatively correlation between compressed model size and accuracy loss. That is, the lower the accuracy loss, the better the compression performance. The results also show that ResNet-18 deployed with our method performs the best when the partition size on convolutional layers is set to 288. In such case, with the same storage reduction, our method can preserve the most useful information of the ResNet-18 model, since its accuracy loss is lower than other settings. In other words, if we choose a smaller partition size than 288, the accuracy loss will be increased due to the damage of filter correlation. In contrast, if we choose a larger partition size, the accuracy loss will also be increased, implying that the dictionary pair-reconstruction process in our method will lose useful information in the weight partitions.

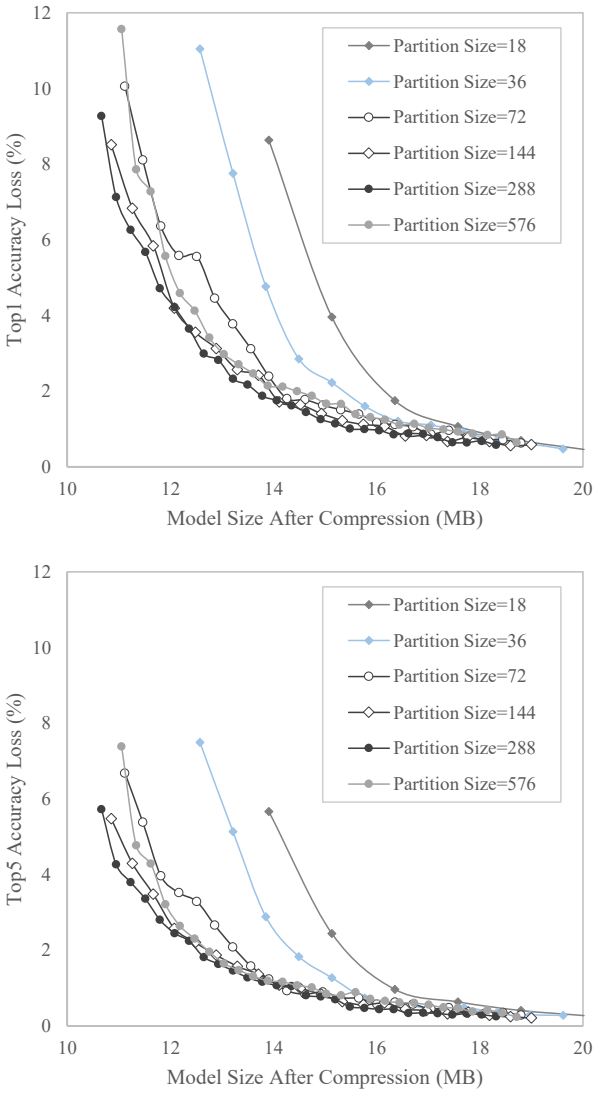

Fig. 5. Performance vs. different partition sizes on ResNet-18

V. CONCLUSION

We have discussed the problem on how to reduce the model size of existing DNN models efficiently, with important information in the weight tensors preserved and without original data. Specifically, we have proposed a general dictionary pair-based data-free fast neural network compression method, which can effectively reduce the memory storage and model size of existing DNN models within a short time, with low performance loss. Given a pre-trained DNN model, we first present a partition strategy to divide the parameters of each layer into weight partitions. Then, by performing the dictionary pair-based fast reconstruction on the partitions, we can potentially preserve the important information and discover more fine-grained information to ensure the performance. Note that our proposed data-free fast compression method can be easily extended to all the DNN models and deployed on both the convolutional and fully-connected layers.

Extensive experiments have demonstrated the effectiveness

of our method in terms of reduced model size and accuracy loss. In future, we will also consider how to deploy the implementation of our compression scheme on hardware devices. In addition, the automatic determination strategy of the number of dictionary words in each layer still needs further investigation.

ACKNOWLEDGMENT

This work is partially supported by National Natural Science Foundation of China (62072151, 62020106007), Anhui Provincial Natural Science Fund for Distinguished Young Scholars (2008085J30), and the Fundamental Research Funds for Central Universities of China (JZ2019HGPA0102). Zhao Zhang is the corresponding author.

REFERENCES

[1] Simonyan, Karen, and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556, 2014.
[2] Deng, Jia, "Imagenet: A large-scale hierarchical image database." IEEE conference on computer vision and pattern recognition, 2009.
[3] Rigamonti, Roberto, "Learning separable filters." Proceedings of the IEEE conference on computer vision and pattern recognition, 2013.
[4] Gu, Shuhang, "Projective dictionary pair learning for pattern classification." Advances in neural information processing systems, 2014.
[5] Denton, Emily, "Exploiting linear structure within convolutional networks for efficient evaluation." arXiv preprint arXiv:1404.0736, 2014.
[6] Stock, Pierre, "And the bit goes down: Revisiting the quantization of neural networks." arXiv preprint arXiv:1907.05686, 2019.
[7] Han, Song, Huizi Mao, and William J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149, 2015.
[8] Howard, Andrew G., "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861, 2017.
[9] Umuroglu, Yaman, "Finn: A framework for fast, scalable binarized neural network inference." Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017.
[10] Lebedev, Vadim, "Speeding-up convolutional neural networks using finetuned cp-decomposition." arXiv preprint arXiv:1412.6553, 2014.
[11] He, Kaiming, "Deep residual learning for image recognition." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2016.
[12] Kim, Hyeji, Muhammad Umar Karim Khan, and Chong-Min Kyung, "Efficient neural network compression." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019.
[13] Tai, Cheng, "Convolutional neural networks with low-rank regularization." arXiv preprint arXiv:1511.06067, 2015.
[14] Liu, Jing, "Discrimination-aware network pruning for deep model compression." IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.
[15] Kim, Yong-Deok, "Compression of deep convolutional neural networks for fast and low power mobile applications." arXiv preprint arXiv:1511.06530, 2015.
[16] Huang, Zehao, and Naiyan Wang, "Data-driven sparse structure selection for deep neural networks." Proceedings of the European conference on computer vision, 2018.
[17] Yu, Ruichi, "Nisp: Pruning networks using neuron importance score propagation." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018.
[18] He, Yang, "Soft filter pruning for accelerating deep convolutional neural networks." arXiv preprint arXiv:1808.06866, 2018.
[19] Kim, Hyeji, and Chong-Min Kyung, "Automatic rank selection for high-speed convolutional neural network." arXiv preprint arXiv:1806.10821, 2018.
[20] Luo, Jian-Hao, Jianxin Wu, and Weiyao Lin, "Thinet: A filter level pruning method for deep neural network compression." Proceedings of the IEEE international conference on computer vision, 2017.
[21] Hu, Hengyuan, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures." arXiv preprint arXiv:1607.03250, 2016.
[22] Aizenberg, Igor, Antonio Luchetta, and Stefano Manetti, "A modified learning algorithm for the multilayer neural network with multi-valued neurons based on the complex QR decomposition." Soft Computing 16.4, pages 563-575, 2012.
[23] Gittens, Alex, and Michael Mahoney, "Revisiting the nystrom method for improved large-scale machine learning." International Conference on Machine Learning. PMLR, 2013.
[24] Oseledets, Ivan V, "Tensor-train decomposition." SIAM Journal on Scientific Computing 33.5, pages 2295-2317, 2011.
[25] Deng, Lei, "Model compression and hardware acceleration for neural networks: A comprehensive survey." Proceedings of the IEEE 108.4, pages 485-532, 2020.
[26] Phan, Anh-Huy, "Stable low-rank tensor decomposition for compression of convolutional neural network." European Conference on Computer Vision, 2020.
[27] Guo, Jinyang, "Model Compression using Progressive Channel Pruning." IEEE Transactions on Circuits and Systems for Video Technology, 2020.
[28] Lin, Mingbao, "Hrank: Filter pruning using high-rank feature map." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020.
[29] Paszke, Adam, "Pytorch: An imperative style, high-performance deep learning library." arXiv preprint arXiv:1912.01703, 2019.
[30] Ruan, Xiaofeng, "EDP: An Efficient Decomposition and Pruning Scheme for Convolutional Neural Network Compression." IEEE Transactions on Neural Networks and Learning Systems, 2020.
[31] Li, Hao, "Pruning filters for efficient convnets." arXiv preprint arXiv:1608.08710, 2016.
[32] Zhao, Chenglong, "Variational convolutional neural network pruning." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019.
[33] Li, Yuchao, "Exploiting kernel sparsity and entropy for interpretable CNN compression." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019.
[34] Krizhevsky, Alex, and Geoffrey Hinton, "Learning multiple layers of features from tiny images." Tech Report, 2009.
[35] https://pytorch.org/docs/stable/torchvision/models
[36] Lin, Shaohui, "Toward compact convnets via structure-sparsity regularized filter pruning." IEEE transactions on neural networks and learning systems 31.2 pages 574-588, 2019.
[37] He, Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019.