Neural Network Compression Based on Tensor Ring Decomposition

Kun Xie, *Member, IEEE*, Can Liu, Xin Wang, *Senior Member, IEEE*, Xiaocan Li, Gaogang Xie, *Member, IEEE*, Jigang Wen, and Kenli Li, *Senior Member, IEEE*

***Abstract*— Deep neural networks (DNNs) have made great breakthroughs and seen applications in many domains. However, the incomparable accuracy of DNNs is achieved with the cost of considerable memory consumption and high computational complexity, which restricts their deployment on conventional desktops and portable devices. To address this issue, low-rank factorization, which decomposes the neural network parameters into smaller sized matrices or tensors, has emerged as a promising technique for network compression. In this article, we propose leveraging the emerging tensor ring (TR) factorization to compress the neural network. We investigate the impact of both parameter tensor reshaping and TR decomposition (TRD) on the total number of compressed parameters. To achieve the maximal parameter compression, we propose an algorithm based on prime factorization that simultaneously identifies the optimal tensor reshaping and TRD. In addition, we discover that different execution orders of the core tensors result in varying computational complexities. To identify the optimal execution order, we construct a novel tree structure. Based on this structure, we propose a top-to-bottom splitting algorithm to schedule the execution of core tensors, thereby minimizing computational complexity. We have performed extensive experiments using three kinds of neural networks with three different datasets. The experimental results demonstrate that, compared with the three state-of-the-art algorithms for low-rank factorization, our algorithm can achieve better performance with much lower memory consumption and lower computational complexity.**

***Index Terms*— Neural network compression, tensor ring (TR) factorization.**

NOMENCLATURE

Symbol	Definition
$\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$	d-order tensor $\mathcal{X}$.
$\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$	Matrix $\mathbf{X}$ with the size of $I_1 \times I_2$.
$\mathbf{x} \in \mathbb{R}^{I_1}$	Vector $\mathbf{x}$ with the size of I_1.
$\mathcal{X}_{i_1,i_2,\ldots,i_d}$	$(i_1, i_2, \ldots, i_d)$ element of tensor $\mathcal{X}$.
B	Batch size of training samples.
$\mathrm{CR}_{-\mathrm{Conv}}$ / $\mathrm{CR}_{-\mathrm{FC}}$	Parameter compression ratio of convolutional layers or fully connected layers.
$Speedup_{-\mathrm{Conv}}$ / $Speedup_{-\mathrm{Conv}}$	Speedup in computational complexity of convolutional layers or fully connected layers.

I. INTRODUCTION

RECENTLY, deep neural networks (DNNs) have made great breakthroughs and found applications in various domains, including natural language processing [1], [2], [3], semantic segmentation [4], as well as image and video recognition [5]. These networks typically consist of millions of parameters, requiring hundreds of megabytes for storage, and cost a huge amount of time for processing. Moreover, each evolution in the architecture of neural networks brings a continuous increase in the number of parameters, further exacerbating the storage and processing requirements. As a result, the deployment of DNNs on resource-limited conventional desktops and portable devices becomes challenging. Therefore, it is critical to reduce the number of parameters and the complexity of processing for the practical use of DNNs.

Considering that a large number of parameters in the neural network are redundant [6], a series of studies have been conducted to compress the neural network, using methods such as low-rank factorization [7], [8], [9], [10], pruning [11], [12], slimming [13], and knowledge distillation [14], [15]. Among these network compression techniques, the ones based on low-rank factorization are the most effective. They factorize the parameters of the neural network into smaller matrices/tensors through matrix/tensor decomposition, which can efficiently reduce the number of parameters without compromising the accuracy of the neural network [16].

Among the compression algorithms that utilize low-rank factorization, those based on tensor factorizations are promising to achieve better performance compared with techniques using the matrix factorization (MF). This is because tensors are higher order extensions of two-order matrices, and tensor models have a stronger ability to represent complex and high-order information hidden in the parameters [17]. Doubtlessly, the obvious differences among the inner structures of different factorization algorithms may be one of the most important reasons that determine the performance of the corresponding compressed DNNs. We notice that unlike traditional tensor factorizations such as CP decomposition [18], [19] and Tucker

Manuscript received 18 March 2023; revised 8 October 2023 and 22 January 2024; accepted 26 March 2024. Date of publication 30 April 2024; date of current version 1 March 2025. This work was supported in part by the National Natural Science Foundation of China/Research Grants Council (RGC) Joint Research Scheme under Grant 62321166652, in part by the National Science Fund for Distinguished Young Scholars under Grant 62025201, in part by the Key Research and Development Program of Hunan Province of China under Grant 2023GK2001, and in part by the Natural Science Foundation of Hunan Province under Grant 2024JJ3014. *(Corresponding author: Can Liu.)*

Kun Xie, Can Liu, Xiaocan Li, and Kenli Li are with the College of Computer Science and Electronics Engineering, Hunan University, Changsha 410002, China (e-mail: xiekun@hnu.edu.cn; canliu@hnu.edu.cn; hnulxc@hnu.edu.cn; lkl@hnu.edu.cn).

Xin Wang is with the Department of Electrical and Computer Engineering, The State University of New York at Stony Brook, Stony Brook, NY 11794 USA (e-mail: x.wang@stonybrook.edu).

Gaogang Xie is with the Institute of Computer Network Information Center, Chinese Academy of Sciences, Beijing 100083, China (e-mail: xie@ict.ac.cn).

Jigang Wen is with the School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China (e-mail: wenjigang@hnust.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNNLS.2024.3383392, provided by the authors.

Digital Object Identifier 10.1109/TNNLS.2024.3383392

decomposition [20], [21], [22], the structure of the emerging tensor ring (TR) decomposition [23] is more flexible. It can represent a high-order tensor by the circular product of a sequence of third-order core tensors, which makes TRD powerful to capture complex information hidden in the parameters of neural networks. Therefore, we design our compression scheme based on TRD.

Although promising, designing an efficient compression scheme based on TRD poses several challenging issues that need to be addressed.

1) *How to decompose the parameters in neural networks through TRD to achieve the maximum compression ratio?* To compress the neural network exploiting TRD, we should first reshape the kernel tensor and weight matrix in the network to a high-order tensor. The size of the reshaped tensor directly impacts the sequence of core tensors obtained by the TRD, which further impacts the compression ratio. To maximize the compression ratio, we need to identify the optimal core tensor sequence by jointly optimizing the procedures of parameter reshaping and TRD.
2) *How to minimize the computational complexity under TRD?* After the parameters are compressed and represented by a sequence of core tensors under TRD, to support forward propagation, we should perform tensor contractions upon these core tensors. We observe that different contraction execution orders of these core tensors result in different computational complexities. To minimize the computational complexity, we need to identify the optimal execution order.

To address the above challenging issues, we make the following technical contributions.

1) For both convolutional layers and fully connected layers in DNNs, we derive the relationship between the number of parameters and the sequence of core tensors under TRD. Based on which, we propose a prime factorization-based algorithm to identify the optimal core tensor sequence by jointly optimizing the procedures of parameter reshaping and the TRD so that to achieve the maximum parameter compression ratio. We are not aware of any existing studies investigating to compress neural networks considering such joint optimization.
2) To identify the optimal execution order when performing tensor contractions, we build a novel tree structure. Based on this structure, we propose a top-to-bottom splitting algorithm to schedule the contraction execution order of core tensors, thereby minimizing the computational complexity of the compressed network.
3) We implement our algorithm in three popular neural networks with three different datasets. The experimental results demonstrate that compared with three state-of-the-art algorithms based on low-rank factorization, our algorithm can achieve better performance with much lower memory consumption and lower computational complexity.

The rest of this article is organized as follows. In Section II, we discuss related work and basic preliminaries on neural network compression. In Section III, we outline the parameter minimization problem in compressing the neural network based on TRD and present our solution based on prime factorization. In Section IV, we present our algorithm for the scheduling of the execution order of the core tensors to minimize the computational complexity. We provide the complete solution and evaluate our proposed techniques in Sections V and VI, respectively. Finally, we conclude the work in Section VII.

II. Related Work and Preliminary

A. Related Works

Among low-rank factorization compression algorithms, MF-based methods [7], [24], [25] decompose the large weight matrix into smaller matrices. Although they can achieve good compression performance, as 2-D matrix can only capture two-order information in parameters, the precision of the neural network may be reduced. For instance, the singular value decomposition-based MF method in [7] can reduce up to 30% parameters in the weight matrix, but it may also lead to a loss of up to 10% in accuracy [9].

Tensors, as a high-order extension of the two-order matrix, can capture complex and high-dimensional information [26]. A few literature studies have investigated network compression based on low-rank tensor factorization [18], [20], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36]. For instance, Liu and Ng [30] propose a tucker decomposition-based algorithm to compress DNNs by considering both the nonlinear response and the multilinear low-rank constraint in the kernel tensor. The work by Wang et al. [33] compresses the network by utilizing CP decomposition, which offers both space and computational efficiency through accelerated multiplication. These initial studies show that, compared with MF-based algorithms, tensor factorization-based methods can achieve a higher compression ratio with less accuracy loss [37].

However, methods based on CP decomposition and tucker decomposition may inevitably suffer from the "curse of dimensionality" [38]. That is, as the spatial dimension increases, the volume of indexes also increases. Tensor train decomposition (TTD) can avoid the "curse of dimensionality" by representing a tensor as linked core tensors with restricted orders [27]. However, the tensor train (TT) model highly depends on permutations of tensor dimensions [39], due to its use of strictly sequential multilinear products over latent cores, which leads to difficulties in finding the optimal TT representation.

In order to alleviate these drawbacks, a novel tensor decomposition method based on TRD has been proposed in [23]. TRD removes the unit rank constraints for the boundary tensor cores. In addition, the multilinear products between tensor cores are not required to strictly follow a specific order, and the tensor core can be circularly shifted. Therefore, TRD can offer a more powerful and generalized representation [40].

A limited number of recent studies [41], [42], [43] have investigated the effectiveness of neural network compression using TRD. For instance, the work by Yu et al. [43] proposes a low-rank sparse TR completion method by imposing the Frobenius norm regularization on its latent space. Although these studies are effective, none of them have considered the significant influence of core tensor size and execution order on the compression ratio and computational complexity. Without a proper design, the compression performance may still be not high.

In this article, we discover that both the reshaping and TRD of the weight tensor impacts the total number of compressed parameters. Furthermore, we also find that different execution

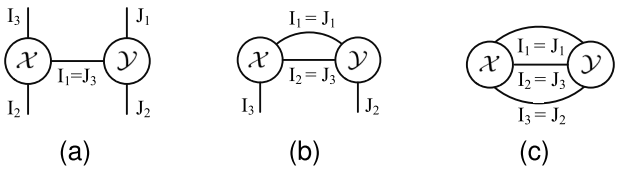


Fig. 1. Examples of tensor contraction. (a) Tensor contraction of two three-order tensors in a single common order yields a four-order tensor $\mathcal{W} \in \mathbb{R}^{I_2 \times I_3 \times J_1 \times J_2}$. (b) Tensor contraction of two three-order tensors in two orders yields a matrix $\mathbf{W} \in \mathbb{R}^{I_3 \times J_2}$. (c) Tensor contraction of two three-order tensors in all orders yields a scalar.

orders of the core tensor result in varying computational complexity in the neural network. Based on these observations, we propose a prime factorization-based algorithm in Section III to maximally compress the number of parameters, and a scheduling algorithm in Section IV to determine the execution order that can minimize the computational complexity in the neural network.

B. Preliminary

This section briefly reviews some preliminaries, a list of the frequently used symbols is provided in the Nomenclature.

1) Tensor Contraction: Tensor contraction is a fundamental and important operation that is considered as a higher dimensional analog of matrix multiplication, inner product, and outer product. The tensor contraction of tensors $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and $\mathcal{B} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_M}$ over the order with the same dimension $I_n = J_m$ (where $1 \leq n \leq N$ and $1 \leq m \leq M$), yields an $(N + M - 2)$-order tensor $\mathcal{C} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N \times J_1 \times \cdots \times J_{m-1} \times J_{m+1} \times \cdots \times J_M}$, where the common order is reduced, the entries in $\mathcal{C}$ can be computed as follows:

$$\mathcal{C}_{i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N,j_1,\ldots,j_{m-1},j_{m+1},\ldots,j_M} = \sum_{i_n=1}^{I_n} \mathcal{A}_{a_{i_1,\ldots,i_{n-1},i_n,i_{n+1},\ldots,i_N}} \mathcal{B}_{b_{j_1,\ldots,j_{m-1},j_m,j_{m+1},\ldots,j_M}}. \quad (1)$$

The operation of (1) is referred to as a contraction of two tensors in a single common order, but tensors can be contracted in several orders or even in all orders, Fig. 1 shows the examples of tensor contraction.

2) Tensor Ring Decomposition: For a given d-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, the TRD aims to represent it by a sequence of three-order tensors that are multiplied circularly. More specifically, a tensor $\mathcal{X}$ can be decomposed into d core tensors $\mathcal{X}^i \in \mathbb{R}^{R_i \times I_i \times R_{i+1}}$, $i = 1, 2, \ldots, d$, expressed as follows:

$$\mathcal{X}_{i_1,i_2,\ldots,i_d} = \sum_{r_1,\ldots,r_d=1}^{R_1,\ldots,R_d} \mathcal{X}^1_{r_1,i_1,r_2} \mathcal{X}^2_{r_2,i_2,r_3}, \ldots, \mathbf{X}^d_{r_d,i_d,r_{d+1}} \quad (2)$$

where $\mathcal{X}_{i_1,i_2,\ldots,i_d}$ denotes $(i_1, i_2, \ldots, i_d)$th element of the tensor, and $R_1, R_2, \ldots, R_d$ are the TR ranks. Note that any two adjacent core tensors, $\mathcal{X}^i$ and $\mathcal{X}^{i+1}$, have an equal dimension R_{i+1} on their corresponding order, and $R_{d+1} = R_1$ in particular.

Fig. 2 shows the diagram representation of a TRD of a d-order tensor $\mathcal{X}$. As shown in Fig. 2, to store the original tensor $\mathcal{X}$, we need to store all the entries, and the total number of entries is $\prod_{i=1}^{d} I_i$. After TRD, the original tensor can be represented by a set of small three-order tensors, and the number of entries to be stored become $\sum_{i=1}^{d} R_i I_i R_{i+1}$ correspondingly, where $R_{d+1} = R_1$. As R_i is usually a smaller value, $\sum_{i=1}^{d} R_i I_i R_{i+1}$ is largely smaller than $\prod_{i=1}^{d} I_i$.

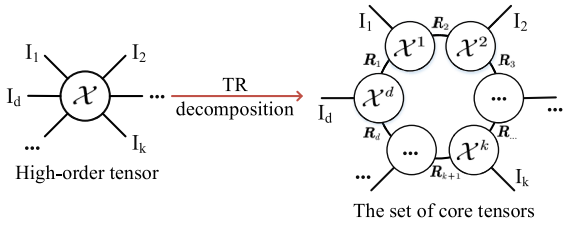


Fig. 2. TRD.

Therefore, compressing parameters in neural network through TRD is a feasible solution.

III. Compressing Neural Network With TRD

A typical DNN generally consists of convolutional layers and fully connected layers. For example, VGG16 consists of 13 convolutional layers and three fully connected layers. In this section, we investigate the compression ratio of both the convolutional and fully connected layers, which is dependent on the size of the reshaped tensor and the corresponding TRD. To achieve the maximum parameter compression ratio, we propose an algorithm based on prime factorization. This algorithm simultaneously determines the optimal tensor reshaping and TRD. We first demonstrate our compression algorithms for convolutional layers in Section III-A and for fully connected layers in Section III-B. Then, we present our prime factorization-based algorithm in Section III-C.

A. Compressing the Convolutional Layer

For a convolutional layer with a three-order input tensor $\mathcal{X} \in \mathbb{R}^{H \times W \times I}$, the three-order output tensor $\mathcal{Y} \in \mathbb{R}^{H' \times W' \times O}$ can be calculated by mapping the input with the convolution of a four-order kernel tensor $\mathcal{W} \in \mathbb{R}^{I \times O \times K \times K}$. The mapping can be described as follows:

$$\mathcal{Y}_{h',w',o} = \sum_{i=1}^{I} \sum_{k_1,k_2=1}^{K} \mathcal{W}_{i,o,k_1,k_2} \mathcal{X}_{h,w,i} \quad (3)$$

where K, I, and O are the spatial channels, input channels, and output channels, respectively.

In visual scenarios, the dimensions of I and O are relatively big. For example, in the LeNet5 model, the kernel size of the second convolutional layer is $(5 \times 5 \times 32 \times 64)$. In this case, the dimensions of I and O correspond to 32 and 64, respectively. To conduct the convolutional layer compression, a straightforward way of applying TRD is to directly decompose the kernel tensor $\mathcal{W} \in \mathbb{R}^{I \times O \times K \times K}$, expressed as follows:

$$\mathcal{W}_{i,o,k_1,k_2} = \sum_{r_1,\ldots,r_4=1}^{R_1,\ldots,R_4} \mathcal{W}^1_{r_1,i,r_2} \mathcal{W}^2_{r_2,o,r_3} \mathcal{W}^3_{r_3,k_1,r_4} \mathcal{W}^4_{r_4,k_2,r_5}$$

where $\mathcal{W}^1 \in \mathbb{R}^{R_1 \times I \times R_2}$, $\mathcal{W}^2 \in \mathbb{R}^{R_2 \times O \times R_3}$, $\mathcal{W}^3 \in \mathbb{R}^{R_3 \times K \times R_4}$, and $\mathcal{W}^4 \in \mathbb{R}^{R_4 \times K \times R_5}$ are the core tensors, and $R_1, \ldots, R_5$ are the TR ranks with $R_5 = R_1$. After compression, the number of parameters in the convolutional layer is $R_1 K R_2 + R_2 K R_3 + R_3 I R_4 + R_4 O R_5$. However, with big I and O, the compression performance is still not high.

To achieve a higher compression ratio, we propose to reshape the convolutional kernel, expressed as follows:

$$\mathcal{W} \in \mathbb{R}^{I \times O \times K \times K} \rightarrow \mathcal{W} \in \mathbb{R}^{I_1 \times \cdots \times I_m \times O_1 \times \cdots \times O_n \times K \times K} \quad (4)$$

where the reshaping satisfies $\prod_{i=1}^{m} I_i = I$ and $\prod_{j=1}^{n} O_j = O$.

Along with the kernel tensor reshaping, to support tensor calculations, the input and output should be also reshaped, expressed as follows:

$$\mathcal{X} \in \mathbb{R}^{H\times W\times I} \to \mathcal{X} \in \mathbb{R}^{H\times W\times I_1\times\cdots\times I_m} \tag{5}$$

$$\mathcal{Y} \in \mathbb{R}^{H'\times W'\times O} \to \mathcal{Y} \in \mathbb{R}^{H'\times W'\times O_1\times\cdots\times O_n}. \tag{6}$$

After reshaping, the convolution operation in (3) can be rewritten as follows:

$$\mathcal{Y}_{h',w',o_1,\ldots,o_n} = \sum_{i_1,\ldots,i_m=1}^{I_1,\ldots,I_m}\sum_{k_1,k_2=1}^{K} \mathcal{W}_{i_1,\ldots,i_m,o_1,\ldots,o_n,k_1,k_2}\mathcal{X}_{h,w,i_1,\ldots,i_m}. \tag{7}$$

To compress the convolutional layer, we exploit TRD to decompose the reshaped kernel tensor as follows:

$$\begin{aligned}\mathcal{W}_{i_1,\ldots,i_m,o_1,\ldots,o_n,k_1,k_2} &= \sum_{r_1,\ldots,r_{m+n+2}=1}^{R_1,\ldots,R_{m+n+2}} \mathcal{W}^1_{r_1,i_1,r_2},\ldots,\mathcal{W}^m_{r_m,i_m,r_{m+1}}\mathcal{W}^{m+1}_{r_{m+1},o_1,r_{m+2}},\ldots,\\ &\quad\times\mathcal{W}^{m+n}_{r_{m+n},o_n,r_{m+n+1}}\mathcal{W}^{m+n+1}_{r_{m+n+1},k_1,r_{m+n+2}}\mathcal{W}^{m+n+2}_{r_{m+n+2},k_2,r_{m+n+3}}\end{aligned} \tag{8}$$

where $\mathcal{W}^{m+n+1} \in \mathbb{R}^{R_{m+n+1}\times K\times R_{m+n+2}}$, $\mathcal{W}^{m+n+2} \in \mathbb{R}^{R_{m+n+2}\times K\times R_{m+n+3}}$, $\mathcal{W}^i \in \mathbb{R}^{R_i\times Q_i\times R_{i+1}}$ are the core tensors, and

$$Q_i = \begin{cases} I_i, & 1\le i\le m \\ O_{i-m}, & m+1\le i\le m+n.\end{cases}$$

$R_1, R_2, \ldots, R_{m+n+3}$ are the TR ranks with $R_{m+n+3} = R_1$.

Using the core tensors obtained in (8), the convolution operation in (7) can be rewritten as follows:

$$\begin{aligned}\mathcal{Y}_{h',w',o_1,\ldots,o_n} &= \sum_{r_{m+1},r_{m+n+1}=1}^{R_{m+1},R_{m+n+1}} \mathcal{W}^O_{r_{m+1},o_1,\ldots,o_n,r_{m+n+1}}\\ &\quad\times\left(\sum_{k_1,k_2=1}^{K}\sum_{r_1=1}^{R_1}\mathcal{W}^K_{r_{m+n+1}\times k_1\times k_2\times r_{m+n+3}}\right.\\ &\quad\left.\times\left(\sum_{i_1,i_2,\ldots,i_m=1}^{I_1,I_2,\ldots,I_m}\mathcal{X}_{h,w,i_1,i_2,\ldots,i_m}\mathcal{W}^I_{r_1,i_1,i_2,\ldots,i_m,r_{m+1}}\right)\right)\end{aligned} \tag{9}$$

where

$$\mathcal{W}^I_{r_1,i_1,\ldots,i_m,r_{m+1}} = \sum_{r_2,\ldots,r_m=1}^{R_2,\ldots,R_m}\mathcal{W}^1_{r_1,i_1,r_2},\ldots,\mathcal{W}^m_{r_m,i_m,r_{m+1}} \tag{10a}$$

$$\mathcal{W}^O_{r_{m+1},o_1,\ldots,o_n,r_{m+n+1}} = \sum_{r_{m+2},\ldots,r_{m+n}=1}^{R_{m+2},\ldots,R_{m+n}}\mathcal{W}^{m+1}_{r_{m+1},o_1,r_{m+2}},\ldots,\mathcal{W}^{m+n}_{r_{m+n},o_n,r_{m+n+1}} \tag{10b}$$

and

$$\mathcal{W}^K_{r_{m+n+1},k_1,k_2,r_{m++n+3}} = \sum_{r_{m+n+2}=1}^{R_{m+n+2}}\mathcal{W}^{m+n+1}_{r_{m+n+1},k_1,r_{m+n+2}}\times\mathcal{W}^{m+n+2}_{r_{m+n+2},k_2,r_{m+n+3}}. \tag{11}$$

We refer these three tensors $\mathcal{W}^I$, $\mathcal{W}^O$, and $\mathcal{W}^K$ as the intermediate tensors in the remainder of this article.

According to (9), the original convolution operation in (7) can be divided into three sequential suboperations, as expressed in (12)–(14). Fig. 3 illustrates the relationship between the original convolution operation and the three suboperations

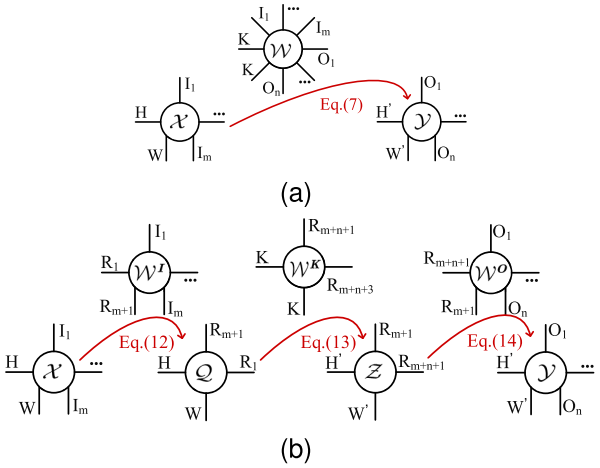


Fig. 3. (a) Original convolution operation in (7). (b) Divide original convolution operation in (7) into suboperations in (12)–(14).

$$\mathcal{Q}_{h,w,r_1,r_{m+1}} = \sum_{i_1,\ldots,i_m=1}^{I_1,\ldots,I_m}\mathcal{X}_{h,w,i_1,\ldots,i_m}\mathcal{W}^I_{r_1,i_1,\ldots,i_m,r_{m+1}} \tag{12}$$

$$\mathcal{Z}_{h',w',r_{m+1},r_{m+n+1}} = \sum_{k_1,k_2=1}^{K}\sum_{r_1=1}^{R_1}\mathcal{Q}_{h,w,r_1,r_{m+1}}\times\mathcal{W}^K_{r_{m+n+1}\times k_1\times k_2\times r_{m+n+3}} \tag{13}$$

$$\mathcal{Y}_{h',w',o_1,\ldots,o_n} = \sum_{r_{m+1},r_{m+n+1}=1}^{R_{m+1},R_{m+n+1}}\mathcal{Z}_{h',w',r_{m+1},r_{m+n+1}}\times\mathcal{W}^O_{r_{m+1},o_1,\ldots,o_n,r_{m+n+1}}. \tag{14}$$

B. Compressing the Fully Connected Layer

For a fully connected layer with an input feature vector $\mathbf{x} \in \mathbb{R}^I$, the output feature vector $\mathbf{y} \in \mathbb{R}^O$ can be calculated by

$$\mathbf{y} = \mathbf{W}\mathbf{x} \tag{15}$$

where $\mathbf{W} \in \mathbb{R}^{I\times O}$ is the weight matrix.

To compress the weight matrix exploiting TRD, we should first reshape the weight matrix $\mathbf{W} \in \mathbb{R}^{I\times O}$ into a tensor form, which can be expressed as follows:

$$\mathbf{W} \in \mathbb{R}^{I\times O} \to \mathcal{W} \in \mathbb{R}^{I_1\times\cdots\times I_m\times O_1\times\cdots\times O_n} \tag{16}$$

where $\mathcal{W}$ is the weight tensor reshaped with I_i and O_j satisfying $\prod_{i=1}^m I_i = I$ and $\prod_{j=1}^n O_j = O$.

Along with the reshaping of the weight matrix, the input and output should be reshaped into tensor forms to support tensor calculations. This can be expressed as follows:

$$\mathbf{x} \in \mathbb{R}^I \to \mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_m}$$

$$\mathbf{y} \in \mathbb{R}^O \to \mathcal{Y} \in \mathbb{R}^{O_1\times\cdots\times O_n}.$$

After the reshaping, the operation in (15) can be rewritten as follows:

$$\mathcal{Y}_{o_1,\ldots,o_n} = \sum_{i_1,\ldots,i_m=1}^{I_1,\ldots,I_m}\mathcal{W}_{i_1,\ldots,i_m,o_1,\ldots,o_n}\mathcal{X}_{i_1,\ldots,i_m}. \tag{17}$$

To compress the fully connected layer, we exploit TRD to decompose the reshaped weight tensor, which can be expressed as follows:

$$\mathcal{W}_{i_1,\ldots,i_m,o_1,\ldots,o_n} = \sum_{r_1,\ldots,r_{m+n}=1}^{R_1,\ldots,R_{m+n}} \mathcal{W}^1_{r_1,i_1,r_2}\mathcal{W}^2_{r_2,i_2,r_3},\ldots, \mathcal{W}^m_{r_m,i_m,r_{m+1}}\mathcal{W}^{m+1}_{r_{m+1},o_1,r_{m+2}},\ldots,\mathcal{W}^{m+n}_{r_{m+n},o_n,r_{m+n+1}} \tag{18}$$

where $\mathcal{W}^i \in \mathbb{R}^{R_i\times Q_i\times R_{i+1}}$ are the core tensors

$$Q_i = \begin{cases} I_i, & 1\le i\le m \\ O_{i-m}, & m+1\le i\le m+n. \end{cases}$$

$R_1, R_2, \ldots, R_{m+n+1}$ are the TR ranks with $R_{m+n+1} = R_1$.

Using the core tensors obtained in (18), operation in (17) can be rewritten as follows:

$$\mathcal{Y}_{o_1,\ldots,o_n} = \sum_{r_{m+1},r_{m+n+1}=1}^{R_{m+1},R_{m+n+1}} \mathcal{W}^O_{r_{m+1},o_1,\ldots,o_n,r_{m+n+1}} \times \left(\sum_{i_1,\ldots,i_m=1}^{I_1,\ldots,I_m} \mathcal{X}_{i_1,\ldots,i_m}\mathcal{W}^I_{r_1,i_1,\ldots,i_m,r_{m+1}}\right) \tag{19}$$

where

$$\mathcal{W}^I_{r_1,i_1,\ldots,i_m,r_{m+1}} = \sum_{r_2,\ldots,r_m=1}^{R_2,\ldots,R_m} \mathcal{W}^1_{r_1,i_1,r_2},\ldots,\mathcal{W}^m_{r_m,i_m,r_{m+1}} \tag{20a}$$

$$\mathcal{W}^O_{r_{m+1},o_1,\ldots,o_n,r_{m+n+1}} = \sum_{r_{m+2},\ldots,r_{m+n}=1}^{R_{m+2},\ldots,R_{m+n}} \mathcal{W}^{m+1}_{r_{m+1},o_1,r_{m+2}},\ldots, \mathcal{W}^{m+n}_{r_{m+n},o_n,r_{m+n+1}}. \tag{20b}$$

According to (19), the original operation in (17) can be divided into two sequential suboperations in (21) and (22). Fig. 4 illustrates the relationship between the original operation and the two suboperations

$$\mathcal{Z}_{r_1,r_{m+1}} = \sum_{i_1,\ldots,i_m=1}^{I_1,\ldots,I_m} \mathcal{X}_{i_1,\ldots,i_m}\mathcal{W}^I_{r_1,i_1,\ldots,i_m,r_{m+1}} \tag{21}$$

$$\mathcal{Y}_{o_1,\ldots,o_n} = \sum_{r_{m+1},r_{m+n+1}=1}^{R_{m+1},R_{m+n+1}} \mathcal{Z}_{r_1,r_{m+1}}\mathcal{W}^O_{r_{m+1},o_1,\ldots,o_n,r_{m+n+1}}. \tag{22}$$

C. Achieving Maximum Compression Ratio With Prime Factorization

As discussed in Sections III-A and III-B, achieving network compression involves reshaping the kernel tensor in the convolutional layer and the weight matrix in the fully connected layer. However, a given kernel tensor $\mathcal{W} \in \mathbb{R}^{I\times O\times K\times K}$ or a weight matrix $\mathbf{W} \in \mathbb{R}^{I\times O}$ can be reshaped into multiple candidate high-order tensors that satisfy $\prod_{i=1}^m I_i = I$ and $\prod_{j=1}^n O_j = O$. Different high-order tensors can be decomposed into varying sequences of core tensors through TRD, resulting in different numbers of parameters in the compressed network. To achieve the maximum compression ratio, the tensor reshaping and TRD should be jointly considered.

In (8), the high-order kernel tensor $\mathcal{W}$ in the convolutional layer is decomposed into a sequence of core tensors, including

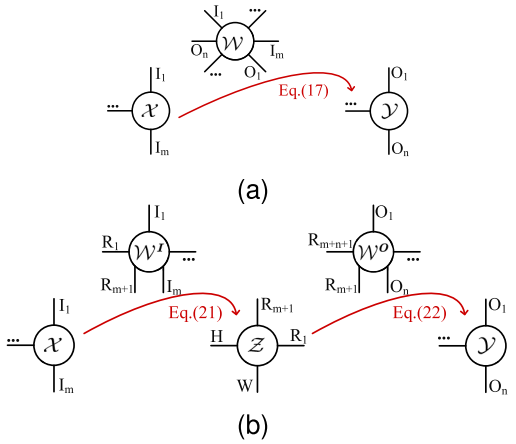


Fig. 4. (a) Operation in (17). (b) Divide (17) into suboperations as (21) and (22).

$\mathcal{W}^{m+n+1} \in \mathbb{R}^{R_{m+n+1}\times K\times R_{m+n+2}}$, $\mathcal{W}^{m+n+2} \in \mathbb{R}^{R_{m+n+2}\times K\times R_{m+n+3}}$, and $\mathcal{W}^i \in \mathbb{R}^{R_i\times Q_i\times R_{i+1}}$, where

$$Q_i = \begin{cases} I_i, & 1\le i\le m \\ O_{i-m}, & m+1\le i\le m+n \end{cases}, \quad R_{m+n+3} = R_1.$$

Similarly, in (18), the high-order weight tensor $\mathcal{W}$ in the fully connected layer is decomposed into a sequence of core tensors, each represented as $\mathcal{W}^i \in \mathbb{R}^{R_i\times Q_i\times R_{i+1}}$, where

$$Q_i = \begin{cases} I_i, & 1\le i\le m \\ O_{i-m}, & m+1\le i\le m+n \end{cases}, \quad R_{m+n+1} = R_1.$$

After performing the TRD, instead of storing the large high-order tensor, we only need to store the set of small core tensors as the parameters of the compressed neural network. Defining P_{Conv} and P_{FC} as the number of parameters in the compressed convolutional layer and compressed fully connected layer, respectively, we have

$$\begin{aligned} P_{\text{Conv}} &= \sum_{i=1}^m (R_i I_i R_{i+1}) + \sum_{j=1}^n (R_{m+j} O_j R_{m+j+1}) \\ &\quad + R_{m+n+1} K R_{m+n+2} + R_{m+n+2} K R_{m+n+3} \\ P_{\text{FC}} &= \sum_{i=1}^m (R_i I_i R_{i+1}) + \sum_{j=1}^n (R_{m+j} O_j R_{m+j+1}). \end{aligned}$$

To maximize the compression ratio, we should solve the following problems:

$$\begin{aligned} \min_{\{I_1,O_1\},\ldots,\{I_m,O_n\}} & \sum_{i=1}^m (R_i I_i R_{i+1}) + \sum_{j=1}^n (R_{m+j} O_j R_{m+j+1}) \\ & + R_{m+n+1} K R_{m+n+2} + R_{m+n+2} K R_{m+n+3} \\ \text{s.t.} & \prod_{i=1}^m I_i = I, \prod_{j=1}^n O_j = O. \end{aligned} \tag{23}$$

$$\begin{aligned} \min_{\{I_1,O_1\},\ldots,\{I_m,O_n\}} & \sum_{i=1}^m (R_i I_i R_{i+1}) + \sum_{j=1}^n (R_{m+j} O_j R_{m+j+1}) \\ \text{s.t.} & \prod_{i=1}^m I_i = I, \prod_{j=1}^n O_j = O. \end{aligned} \tag{24}$$

Given the TR ranks, $R_{m+n+1}KR_{m+n+2} + R_{m+n+2}KR_{m+n+3}$ in (23) is fixed. Therefore, if the minimization problem in (24) is solved, the above two problems can be solved.

To solve the problem in (24), we observe that the objective functions $\sum_{i=1}^{m}(R_i I_i R_{i+1})$, $\sum_{j=1}^{n}(R_{m+j} O_j R_{m+j+1})$ and the constraints $\prod_{i=1}^{m} I_i = I$, $\prod_{j=1}^{n} O_j = O$ are all independent. Consequently, the minimization problem can be further decoupled into two subproblems

$$\begin{cases} \min\limits_{I_1,\ldots,I_m} \sum\limits_{i=1}^{m}(R_i I_i R_{i+1}), & \text{s.t. } \prod_{i=1}^{m} I_i = I \quad (25a) \\ \min\limits_{O_1,\ldots,O_n} \sum\limits_{j=1}^{n}(R_{m+j} O_j R_{m+j+1}), & \text{s.t. } \prod_{j=1}^{n} O_j = O \quad (25b) \end{cases}$$

where the first corresponds to a decomposition problem in the input domain (i.e., related to I), and the second corresponds to another decomposition problem in the output domain (i.e., related to O).

Following the setting of study [41], [42], [44], [45], we set $R_1 = R_2 = \cdots = R_{m+n+3} = R$ for simplicity, and the scalar R is referred as the TR rank in the remainder of this article. Then, the problem in (25a) and (25b) can be rewritten as follows:

$$\begin{cases} \min\limits_{I_1,\ldots,I_m} \sum\limits_{i=1}^{m} I_i & \text{s.t. } \prod_{i=1}^{m} I_i = I \quad (26a) \\ \min\limits_{O_1,\ldots,O_n} \sum\limits_{i=1}^{n} O_i & \text{s.t. } \prod_{j=1}^{n} O_j = O. \quad (26b) \end{cases}$$

Clearly, (26a) and (26b) can be formulated as integer factorization problems. These problems involve decomposing a composite number into smaller factors, while minimizing the sum of these factors.

Before presenting our solution of (26a) and (26b), we first provide Theorem 1.

Theorem 1: Suppose that there are two positive integers a and b greater than 2, we must have $a \times b > a + b$.

The detailed proof is presented in Appendix A.

According to Theorem 1, we can deduce that if I and O in (26a) and (26b) can be decomposed as the product of some small factors that can no longer be decomposed, the storage cost can be minimized.

In number theory, prime factors are integers that can no longer be decomposed into other smaller integers, besides 1 and itself. A prime factorization is a way of decomposing a number as a product of its prime factors. An example of prime factorization is $60 = 2 \times 3 \times 2 \times 5$, where 2, 3, and 5 are prime factors.

Therefore, we propose to solve the minimization problem in (26a) and (26b) by using prime factorization. Theorem 2 confirms the feasibility of exploiting prime factorization to solve our problems in (26a) and (26b).

Theorem 2: For a given integer N that satisfies $N = \prod_{i=1}^{n} N_i$, where N_i is the factor and $i = 1, 2, \ldots, n$, the sum of these factors $\sum_{i=1}^{n} N_i$ is the smallest when all the factors are prime factors.

The detailed Proof of Theorem 2 is presented in the Appendix B.

Fig. 5 shows an example to illustrate our solution. Fig. 5(a) shows a small fully connected layer with weight matrix $\mathbf{W} \in \mathbb{R}^{980\times 35}$, the number of parameters in the network is 34 300. According to Theorem 2, we perform prime factorization on 980 and 35 to obtain the set of prime factors used to determine the dimensions of the weight tensor, which is $\mathcal{W} \in \mathbb{R}^{2\times2\times5\times7\times5\times7}$, as shown in Fig. 5(b). Then, we decompose the weight tensor exploiting TRD to get the set of core tensors $\mathcal{W}^1 \in \mathbb{R}^{R\times2\times R}$, $\mathcal{W}^2 \in \mathbb{R}^{R\times2\times R}$, $\mathcal{W}^3 \in \mathbb{R}^{R\times5\times R}$, $\mathcal{W}^4 \in \mathbb{R}^{R\times7\times R}$, $\mathcal{W}^5 \in \mathbb{R}^{R\times7\times R}$, $\mathcal{W}^6 \in \mathbb{R}^{R\times5\times R}$, and $\mathcal{W}^7 \in \mathbb{R}^{R\times7\times R}$.

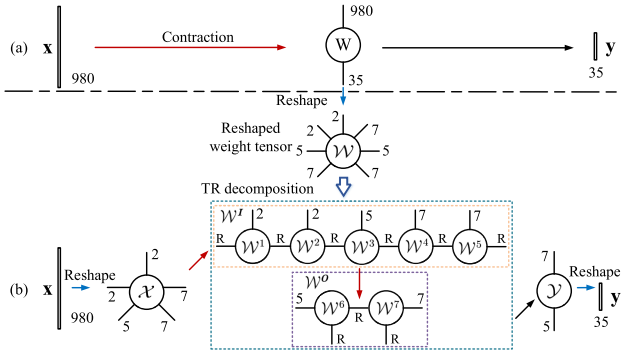


Fig. 5. (a) Example of a fully connected layer. (b) Fully connected layer compression based on TRD.

D. Parameter Compression Ratio

For a convolutional layer with a four-order kernel tensor $\mathcal{W} \in \mathbb{R}^{K\times K\times I\times O}$ and a fully connected layer with a weight matrix $\mathbf{W} \in \mathbb{R}^{I\times O}$, the number of parameters are $KKIO$ and IO, respectively.

Under TRD, the prime factorization of I and O determines the sequence of core tensors. According to Theorem 2, if I and O are decomposed as products of m and n prime factors, respectively, we will get $m+n$ core tensors denoted by $\mathcal{W}^i \in \mathbb{R}^{R\times K_i\times R}$

$$K_i = \begin{cases} I_i, & \text{when } 1 \le i \le m \\ O_{i-m}, & \text{when } m+1 \le i \le m+n \end{cases}$$

where I_i $(i = 1, 2, \ldots, m)$ are the prime factors of I, and O_j $(j = 1, 2, \ldots, n)$ are the prime factors of O.

Therefore, the parameter compression ratios $\mathrm{CR}_{-\mathrm{Conv}}$ and $\mathrm{CR}_{-\mathrm{FC}}$ of the convolutional layer and the fully connected layer can be, respectively, expressed as follows:

$$\mathrm{CR}_{-\mathrm{Conv}} = \frac{KKIO}{R^2\left(2K + \sum_{i=1}^{m} I_i + \sum_{j=1}^{n} O_j\right)} \quad (27)$$

$$\mathrm{CR}_{-\mathrm{FC}} = \frac{\mathrm{IO}}{R^2\left(\sum_{i=1}^{m} I_i + \sum_{j=1}^{n} O_j\right)}. \quad (28)$$

In the example of Fig. 5, the value of I, O, m, n, I_1, I_2, I_3, I_4, I_5, O_1, and O_2 are 980, 35, 5, 2, 2, 2, 5, 7, 7, 5, and 7, respectively. The parameters in the fully connected layer before and after compressions are 34 300 and $35R^2$. When $R = 2$, the compression ratio in this example is up to 245.

IV. Minimizing the Complexity of Computation

In this section, we investigate the computational cost of the compressed network, which is determined by the sequence in which the core tensors are contracted. To minimize the computational cost, we build a novel tree structure and propose a top-to-bottom splitting algorithm to schedule the contracting execution order of core tensors.

A. Computational Cost of Compressed Network

After network compression, to perform forward propagation over the convolutional layer, the computational cost consists of three parts.

1) *Part 1:* Performing tensor contractions upon core tensors $\mathcal{W}^i \in \mathbb{R}^{R_i \times I_i \times R_{i+1}}$ to obtain $\mathcal{W}^I$ and performing tensor contractions upon core tensors $\mathcal{W}^j \in \mathbb{R}^{R_j \times I_j \times R_{j+1}}$ to obtain $\mathcal{W}^O$ where $1 \le i \le m, 1 \le j \le n$.
2) *Part 2:* Performing tensor contractions upon core tensors $\mathcal{W}^{m+n+1} \in \mathbb{R}^{R_{m+n+1} \times K \times R_{m+n+2}}$ and $\mathcal{W}^{m+n+2} \in \mathbb{R}^{R_{m+n+2} \times K \times R_{m+n+3}}$ to obtain the tensor $\mathcal{W}^K$.
3) *Part 3:* Performing (12)–(14) to obtain the output of the convolutional layer.

To perform forward propagation over the fully connected layer, the computational cost consists of two parts.

1) *Part 1:* Performing tensor contractions upon core tensors $\mathcal{W}^i \in \mathbb{R}^{R_i \times I_i \times R_{i+1}}$ to obtain $\mathcal{W}^I$, and performing tensor contractions upon core tensors $\mathcal{W}^j \in \mathbb{R}^{R_j \times I_j \times R_{j+1}}$ to obtain $\mathcal{W}^O$ where $1 \le i \le m, 1 \le j \le n$.
2) *Part 2:* Performing (21) and (22) to obtain the output of the fully connected layer.

In this article, we use the number of additions and multiplications floating point operations (FLOPs) [41] as the metric to measure the complexity of computation.

Given the TR ranks, the number of FLOPs of part 2 in the convolutional layer is a fixed value. The number of FLOPs in (12)–(14), (21), and (22) are $2R^2HWI$, $2R^3KKHW$, $2R^2H'W'O$, $2R^2I$, and $2R^2O$, respectively. Apparently, as H, W, I, O, K, H', and W' are fixed values in a given model, the computational costs for part 3 in the convolutional layer and part 2 in the fully connected layer are fixed as well.

Thus, to minimize the computational complexity in both the convolution layer and the fully connected layer, we should optimize the procedure of tensor contractions in part 1 while obtaining $\mathcal{W}^I$ and $\mathcal{W}^O$.

B. Computational Complexity Under Different Contraction Order

The tensors $\mathcal{W}^I$ and $\mathcal{W}^O$ are obtained by executing a sequence of tensor contractions upon multiple core tensors. In the examples shown in Fig. 6, we use the tensor $\mathcal{W}^I \in \mathbb{R}^{2\times2\times5\times7\times7}$ from Fig. 5 as an example to demonstrate how different orders of tensor contractions for the core tensors can result in varying computational costs.

In Fig. 6, the core tensors are $\mathcal{W}^1 \in \mathbb{R}^{R\times2\times R}$, $\mathcal{W}^2 \in \mathbb{R}^{R\times2\times R}$, $\mathcal{W}^3 \in \mathbb{R}^{R\times5\times R}$, $\mathcal{W}^4 \in \mathbb{R}^{R\times7\times R}$, and $\mathcal{W}^5 \in \mathbb{R}^{R\times7\times R}$. In Fig. 6(a), core tensors $\mathcal{W}^1 \in \mathbb{R}^{R\times2\times R}$ and $\mathcal{W}^2 \in \mathbb{R}^{R\times2\times R}$ are contracted first, resulting in an $R \times 2 \times 2 \times R$ tensor with contraction cost being $2 \times 2 \times 2 \times R^3 = 8R^3$ FLOPs. Then by further contracting core tensors $\mathcal{W}^3 \in \mathbb{R}^{R\times5\times R}$, $\mathcal{W}^4 \in \mathbb{R}^{R\times7\times R}$ and $\mathcal{W}^5 \in \mathbb{R}^{R\times7\times R}$ sequentially, we obtain the large root tensor in Fig. 6(a). The total number of FLOPs in Fig. 6(a) is obtained by summing up the FLOPs involved in each tensor contraction, resulting in a value of $2288R^3$.

In Fig. 6(c), the contractions are performed in a different order, leading to a distinct computational cost. The total number of FLOPs in this case is $2106R^3$.

C. Scheduling the Execution Order of the Core Tensors

In Section IV-B, we already know that given a set of core tensors, different contraction execution orders upon the core tensors result in different computational costs. To minimize the computational cost in the compressed neural network, we should well schedule the contraction execution order of the core tensors. We call this problem as a computational scheduling problem.

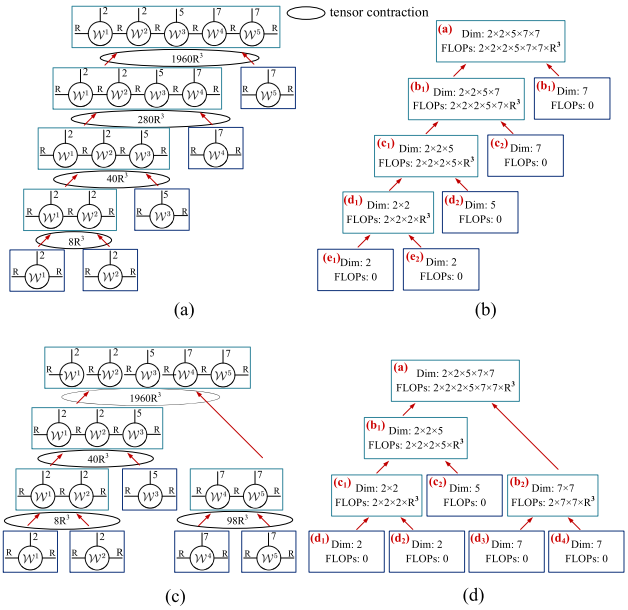


Fig. 6. FLOPs under different execution schemes. (a) and (c) Two different execution schemes. (b) and (d) Tree diagrams of (a) and (c), respectively.

Before we design our scheduling algorithm, we first provide Fig. 6(b) and (d) as the tree diagrams of Fig. 6(a) and (c), respectively. Each box corresponds to a node in the tree that has two attributes.

1) Dim, denotes the dimensions of the tensor, e.g., the dimensions $R \times 2 \times 2 \times 5 \times 7 \times 7 \times R$ of the node (a) is abbreviated as $2 \times 2 \times 5 \times 7 \times 7$.
2) FLOPs, denotes the number of FLOPs incurred by tensor contraction to get that node, e.g., the FLOPs of the node (a) is abbreviated as $2 \times 2 \times 2 \times 5 \times 7 \times 7 \times R^3$.

Following features can be observed in the two tree diagrams.

1) The Dim of all nonleaf nodes is the product of the Dim of their left child node and right child node, e.g., (c_1) and (c_2) are, respectively, the left child node and right child node of node (b_1). We have (b_1)"."Dim = (c_1).Dim $\times$ (c_2)"."Dim.
2) The number of FLOPs of all nonleaf nodes is the product of the Dim of their left child node and right child node multiplied by $2R^3$, e.g., (b_1)"."FLOPs = (c_1)"."Dim $\times$ (c_2)"."Dim $\times$ $2R^3$.
3) The number of FLOPs of all nonleaf nodes is the product of their own Dims multiplied by $2R^3$, e.g., (b_1).FLOPs = (b_1).Dim $\times$ $2R^3$.
4) The root node corresponds to the final contraction result $\mathcal{W}^I$ and $\mathcal{W}^O$. Given the same set of core tensors, different trees have the same final contraction result, thus the same root.
5) Leaf nodes correspond to the core tensors. Different trees have the same set of leaf nodes. Although the leaf node is not a contraction result, we set the FLOPs of the leaf node to 0 for the convenience of presentation.

Based on the tree model, one execution order corresponds to one tree. The computational cost of one execution order is the sum of FLOPs of all nonleaf nodes in the tree. Consequently, within the tree model, the computational scheduling problem can be reformulated as a tree-building problem: given leaf

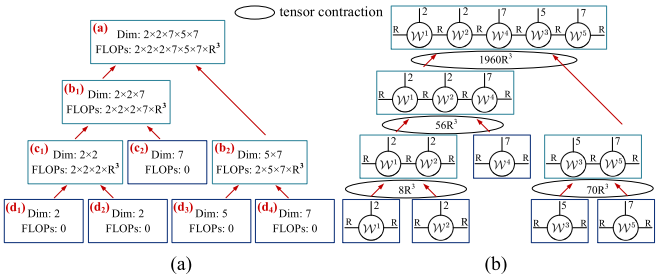


Fig. 7. FLOPs under optimal execution order. (a) Tree diagram of the optimal execution order. (b) FLOPs under optimal execution order.

nodes, build a binary tree to achieve the lowest computational cost.

Before we present our tree-building algorithm, we provide Theorem 3.

Theorem 3: Suppose that there are two positive integers a and b greater than 2. When the product of a and b is a fixed constant, the smaller the difference $|a-b|$ between the two integers, the smaller the sum $a+b$ of the two numbers.

The detailed Proof of Theorem 3 is presented in the Appendix C.

When examining the trees in Fig. 6(b) and (d), we can observe two perspectives on the tree construction process. First, from a bottom–up perspective, the tree building can be seen as executing a sequence of tensor contractions. Second, from a top–bottom viewpoint, the tree building can be seen as executing a sequence of splitting operations, gradually splitting parent nodes from the root node to the leaf nodes (core tensors).

Based on Theorem 3, we propose an algorithm to build the tree by splitting the node from the top to the bottom in the following way.

1) Splits start from the root, splitting ends until reaching the leaf nodes (core tensors).
2) Each splitting satisfies a *splitting principle*: minimizing the difference between the Dim of its left child and the right child.

Given the same root node and core tensors in Fig. 6, under our algorithm, the tree can be built as shown in Fig. 7. The number of FLOPs under the tree in Fig. 7 is $2094R^3$, which is less than that under Fig. 6. From the tree, we can easily obtain the contraction order from the bottom to the top.

D. Speedup

For a convolutional layer with a three-order input tensor $\mathcal{X} \in \mathbb{R}^{H\times W\times I}$, the three-order output tensor $\mathcal{Y} \in \mathbb{R}^{H'\times W'\times O}$ can be mapped with the convolution of a four-order kernel tensor $\mathcal{W} \in \mathbb{R}^{K\times K\times I\times O}$, in which the complexity of computation in terms of the number of FLOPs is $2H'W'OK^2I$. For a fully connected layer with input $\mathbf{X} \in \mathbb{R}^I$ and weight matrix $\mathbf{W} \in \mathbb{R}^{I\times O}$, the complexity of the computation in terms of the number of FLOPs of obtaining the output $\mathbf{y} \in \mathbb{R}^O$ is $2IO$.

In the beginning of Section IV, we have listed the main parts that contribute to the computational complexity of the compressed network. Among these, only the computational complex of calculating $\mathcal{W}^I$ and $\mathcal{W}^O$ is not fixed.

Before we provide the computational complexity in a compressed network, we first present Theorems 4 and 5.

Theorem 4: For m TR core tensors $\mathcal{W}^i \in \mathbb{R}^{R\times I_i\times R}, i = 1,2,\ldots,m$ that satisfy $\prod_{i=1}^m I_i = I$, to recover the tensor $\mathcal{W}^I \in \mathbb{R}^{R\times I_1\times I_2\times\cdots\times I_m\times R}$ following the contraction order obtained by our tree splitting algorithm, the computational complexity in terms of the number of FLOPs is between $2R^3I$ and $4R^3I$.

Theorem 5: For n TR core tensors $\mathcal{W}^j \in \mathbb{R}^{R\times O_j\times R}, j = 1,2,\ldots,n$ that satisfy $\prod_{j=1}^n O_j = O$, to recover the tensor $\mathcal{W}^O \in \mathbb{R}^{R\times O_1\times O_2\times\cdots\times O_n\times R}$ following the contraction order obtained by our tree splitting algorithm, the computational complexity in terms of the number of FLOPs is between $2R^3O$ and $4R^3O$.

The detailed Proof of Theorem 4 is presented in Appendix D. We can prove Theorem 5 following the Proof of Theorem 4.

Thus, compared with the execution in the original convolutional layer and the fully connected layer with the kernel tensor $\mathcal{W} \in \mathbb{R}^{K\times K\times I\times O}$ and weight matrix $\mathbf{W} \in \mathbb{R}^{I\times O}$, the speedup in our scheduled scheme can be presented as follows:

$$\begin{aligned}\text{Speedup}_{-\text{Conv}} &= BH'W'OK^2I/\big(2R^3(I+O)+BR^2HWI\\ &\quad +BR^3H'W'K^2+BR^2H'W'O\big)\\ \text{Speedup}_{-\text{FC}} &= \frac{BIO}{(2R^3+BR^2)(I+O)}\end{aligned}$$

where B is the batch size of training samples.

In the example of Fig. 5, for the fully connected layer with input $\mathbf{X} \in \mathbb{R}^{980}$ and weight matrix $\mathbf{W} \in \mathbb{R}^{980\times 35}$, the number of FLOPs of obtaining the output $\mathbf{y} \in \mathbb{R}^{35}$ in the original execution is 68 600. Under our scheme, we first perform the prime factorization on 980 and 35 to obtain the set of prime factors used to determine the dimensions of the weight tensor, which is $\mathcal{W} \in \mathbb{R}^{2\times2\times5\times7\times7\times5\times7}$. Then, we decompose this weight tensor to obtain the set of core tensors including $\mathcal{W}^1 \in \mathbb{R}^{R\times2\times R}, \mathcal{W}^2 \in \mathbb{R}^{R\times2\times R}, \mathcal{W}^3 \in \mathbb{R}^{R\times5\times R}, \mathcal{W}^4 \in \mathbb{R}^{R\times7\times R}, \mathcal{W}^5 \in \mathbb{R}^{R\times7\times R}, \mathcal{W}^6 \in \mathbb{R}^{R\times5\times R}$, and $\mathcal{W}^7 \in \mathbb{R}^{R\times7\times R}$. Finally, we perform our tree-based contraction scheduling algorithm to recover the intermediate tensors $\mathcal{W}^I \in \mathbb{R}^{R\times2\times2\times5\times7\times7\times R}$ and $\mathcal{W}^O \in \mathbb{R}^{R\times5\times7\times R}$ to obtain the output tensor, in which the number of FLOPs is $2030R^2+2164R^3$. When $R=2$, the speedup is 2.7.

V. Refined Factorization and Complete Solution

A. Refined Factorization

In Section III-C, to solve problems in (26a) and (26b), we propose a prime factorization-based algorithm to obtain the optimal set of core tensors to achieve the maximal parameter compression ratio. In this section, based on the prime factorization, we propose a refined factorization which can achieve the same compression ratio compared with the prime factorization while having lower computational complexity.

Take the fully connected layer as an example, given a weight matrix $\mathbf{W} \in \mathbb{R}^{I\times O}$, we denote the prime factorization of I and O as $I = \prod_{i=1}^m I_i$ and $O = \prod_{j=1}^n O_j$, where I_i $(i = 1,2,\ldots,m)$ and O_j $(j = 1,2,\ldots,n)$ are prime factors. We have these prime factors that can minimize $\sum_{i=1}^m I_i$ and minimize $\sum_{i=1}^m O_i$, and thus achieve the maximal parameter compression ratio. The set of core tensors under this factorization is listed as $\mathcal{W}^i \in \mathbb{R}^{R\times K_i\times R}$, where

$$K_i = \begin{cases} I_i, & \text{when } 1\le i\le m\\ O_{i-m}, & \text{when } m+1\le i\le m+n.\end{cases}$$

That is, each I_i $(i = 1,2,\ldots,m)$ and O_j $(j = 1,2,\ldots,n)$ corresponds to a dimension of core tensor decomposed.

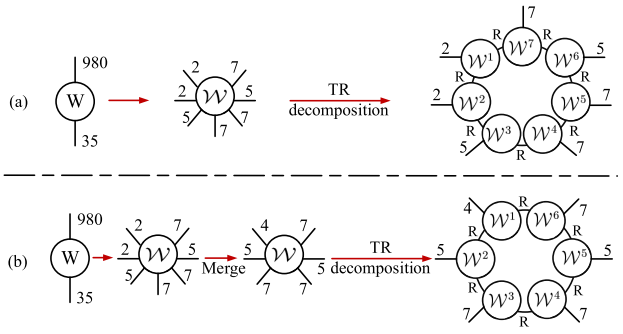


Fig. 8. Parameters before (a) and after merge (b).

We notice that 2 is the smallest prime factor that has the feature $2 \times 2 = 4 = 2 + 2$. Taking advantage of this feature, we propose a *merge principle* to obtain a refined factorization: if there exist even prime factors being 2 in I_i $(i = 1, 2, \ldots, m)$ or O_j $(j = 1, 2, \ldots, n)$, we can merge each pair of prime factors $I_p = I_q = 2$ $(1 \leq p, q \leq m)$ or $O_{p'} = O_{q'} = 2$ $(1 \leq p', q' \leq n)$. Easily, we have the same number of parameters under the prime factorization and the refined factorization.

We give an example in Fig. 8 to illustrate that the numbers of parameters under the prime factorization and the refined factorization are the same.

Under the prime factorization in Fig. 8(a), the weight matrix will be reshaped into tensor $\mathcal{W} \in \mathbb{R}^{2\times2\times5\times7\times7\times5\times7}$, which will be decomposed into 7 core tensors $\mathcal{W}^1 \in \mathbb{R}^{R\times2\times R}$, $\mathcal{W}^2 \in \mathbb{R}^{R\times2\times R}$, $\mathcal{W}^3 \in \mathbb{R}^{R\times5\times R}$, $\mathcal{W}^4 \in \mathbb{R}^{R\times7\times R}$, $\mathcal{W}^5 \in \mathbb{R}^{R\times7\times R}$, $\mathcal{W}^6 \in \mathbb{R}^{R\times5\times R}$, and $\mathcal{W}^7 \in \mathbb{R}^{R\times7\times R}$, which leads to $35R^2$ parameters.

Under the refined factorization in Fig. 8(b), the weight matrix will be reshaped to tensor $\mathcal{W} \in \mathbb{R}^{4\times5\times7\times7\times5\times7}$ which will be decomposed into six core tensors $\mathcal{W}^1 \in \mathbb{R}^{R\times4\times R}$, $\mathcal{W}^2 \in \mathbb{R}^{R\times5\times R}$, $\mathcal{W}^3 \in \mathbb{R}^{R\times7\times R}$, $\mathcal{W}^4 \in \mathbb{R}^{R\times7\times R}$, $\mathcal{W}^5 \in \mathbb{R}^{R\times5\times R}$, and $\mathcal{W}^6 \in \mathbb{R}^{R\times7\times R}$, which leads to $35R^2$ parameters, the same to that under the prime factorization.

B. Complete Solution

Our complete solution to achieve a compressed neural network with maximal parameter compression ratio under low computational complexity consists of two parts.

1) *Obtaining the Optimal Core Tensors in the Compressed Neural Network:* First, the prime factorization-based algorithm in Section III-C is applied to the kernel tensor of the convolutional layer and weight matrix of the fully connected layer, and then, the merge principle in Section V-A is followed to obtain the refined optimal core tensors.
2) *Minimizing the Computational Complexity in the Compressed Neural Network:* The tree-based splitting algorithm in Section IV-C is applied to identify the best contraction execution order of the refined optimal core tensors.

To implement our scheme, our training process first applies our prime factorization algorithm to obtain optimal size of core tensors, then schedules the contraction sequence of the core tensors. After that, the whole compressed network is trained by minimizing a loss function to calculate the values of the core tensors. During inference, we execute the trained compressed network.

VI. Experiments

We validate our solution using three different neural network models.

1) *LeNet-300-100 [46]:* LeNet-300-100 is a fully connected network with three fully connected layers, where the weight matrices are 784×300, 300×100, and 100×10, respectively.
2) *LeNet5 [47]:* LeNet5 is a convolutional neural network consisting of two convolution layers which have the dimensions of $5 \times 5 \times 1 \times 32$ and $5 \times 5 \times 32 \times 64$, and two fully connected layers in which the weight matrices are 3136×1024 and 1024×10.
3) *VGG16 [48]:* VGG16 is a deep convolutional neural network that consists of 16 weight layers, including 13 convolutional layers and three fully connected layers.

We train the above models using three different datasets.

1) *MNIST Dataset [49]:* The MNIST dataset comprises 10ten class handwritten digits with a size of 28×28, such as 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, in which the training set contains 60 000 examples, and the test set has 10 000 examples.
2) *FASHION-MNIST Dataset [50]:* The FASHION-MNIST dataset is a benchmark dataset of the front look of 70 000 grayscale fashion product images with a size of 28×28, in which 60 000 are used for training and 10 000 are used for testing.
3) *CIFAR-100 Dataset [51]:* The CIFAR-100 dataset is a large-scale dataset used for evaluating and comparing image classification tasks. It consists of 50 000 training images and 10 000 testing images, each with a resolution of 32×32 pixels. The images are classified into 100 different fine-grained classes, with each class containing 500 training images and 100 testing images.

We implement five low-rank factorization-based compression algorithms for performance evaluation.

1) MF [24], which compresses the fully connected layer exploiting single value decomposition (SVD).
2) TTD-FC [37], which converts the weight matrices of the fully connected layers into the tensor train format [27] to reduce the number of parameters.
3) TTD [52], which formulates the convolutional layer as a matrix-by-matrix multiplication and reshape the 4-D kernel tensor into a matrix. Then, it compresses both the convolutional layer and the fully connected layer using TTD.
4) TRD [42], which compresses the network based on TRD. Different from our proposed algorithm, the kernel tensor and weight matrix in [42] are reshaped based on the experience or experiment, resulting in low compression ratio. After the set of core tensors is obtained, the intermediate tensors $\mathcal{W}^I$ and $\mathcal{W}^O$ are recovered without scheduling, which causes a lot of computational overhead.
5) Our proposed TRD [prime factorization-based TRD (PFTRD)].

All experiments on LeNet-300-100 and LeNet5 were implemented on Intel[1] Core[2] i7-9700 CPU at 3.00 GHz, and all experiments on VGG16 were implemented on an Nvidia

[1]Registered trademark.
[2]Trademarked.

TABLE I
PARAMETERS IN LENET-300-100

	Parameters under Different Methods		
TR Rank	Random	PFTRD	Uncompressed
2	0.81K	0.77K	266.61K
5	2.94K	2.69K	266.61K
10	10.51K	9.51K	266.61K
15	23.14K	20.89K	266.61K
20	40.81K	36.81K	266.61K

TABLE II
PARAMETERS IN LENET5

	Parameters under Different Methods		
TR Rank	Random	PFTRD	Uncompressed
2	1.70K	1.63K	3274.634K
5	4.68K	4.28K	3274.634K
10	15.33K	13.73K	3274.634K
15	33.08K	29.48K	3274.634K
20	57.93K	51.53K	3274.634K

TABLE III
PARAMETERS IN VGG16

	Parameters under Different Methods		
TR Rank	Random	PFTRD	Uncompressed
2	15.6K	14.94K	34.01M
5	31.77K	27.69K	34.01M
10	89.52	73.22K	34.01M
15	185.77K	149.09K	34.01M
20	320.52K	255.32K	34.01M

TABLE IV
FLOPS IN LENET300-100

	FLOPs under Different Methods			
TR Rank	Random 1	Random 2	Tree	Uncompressed
2	2608831	2591743	2590639	104467195
5	16979713	16712713	16695463	104467195
10	74171263	72035263	71897263	104467195
15	181255013	174046013	173580263	104467195
20	347856463	330768463	329664463	104467195

TABLE V
FLOPS IN LENET5

	FLOPs under Different Methods			
TR Rank	Random 1	Random 2	Tree	Uncompressed
2	162600716	162536348	162533708	5367979184
5	1670605274	1669599524	1669558274	5367979184
10	11376667724	11368621724	11368291724	5367979184
15	36216603974	36189448724	36188334974	5367979184
20	83278458524	83214090524	83211450524	5367979184

TABLE VI
FLOPS IN VGG16

	FLOPs under Different Methods			
TR Rank	Random 1	Random 2	Tree	Uncompressed
2	3472541388	3472528716	3472129740	485505437094
5	23490466890	23490268890	23484034890	485505437094
10	74196017676	74194433676	74144561676	485505437094
15	181648758598	181643412598	181475094598	485505437094
20	336508122492	336495450492	336096474492	485505437094

GTX 1660 GPU. These networks were trained using TensorFlow [53]. The uncompressed networks were trained from initialization. Using the trained network, we further apply the compression algorithms and retrain the network.

A. Validation of the Prime Factorization-Based TRD

In Section III-C, to identify the optimal core tensor sequence by jointly optimizing the procedure of parameter reshaping and TRD, we propose a prime factorization-based method. In this section, we present experimental results to validate our prime factorization-based TRD.

We compare our PFTRD with another TRD(denoted by Random). Tables I–III list the number of parameters under different TR ranks in LeNet300-100, LeNet5 and VGG16, respectively. Obviously, with the same TR rank, the number of parameters under prime factorization-based TRD is less than that under the Random.

B. Verification of Execution Order Scheduling Algorithm

In Section IV-C, to minimize the computational complexity, we propose a novel tree-based splitting algorithm to schedule the contraction order of the core tensors. In this section, we present experimental results to validate our tree-based splitting algorithm.

We compare our tree-based splitting algorithm (denoted by Tree) with two random splittings (denoted by Random 1 and Random 2) using core tensors obtained by our prime factorization-based TRD. Tables IV–VI list the number of FLOPs under different TR ranks in LeNet300-100, LeNet5, and VGG16, respectively. As expected, under the same TR rank, the number of FLOPs under our tree-based splitting algorithm is less than that under the two random contractions.

C. Comparison Results

1) Parameters, Compression Ratio, FLOPs, Speedup, Memory Consumption, and Accuracy Results: We list these comparison results of compressing LeNet300-100, LeNet5, and VGG16 models under different compression algorithms.

a) LeNet300-100 compression on MNIST: The model was trained for 40 epochs using a mini-batch size of 64. Stochastic gradient descent (SGD) with a momentum of 0.9 and a decaying learning rate of 0.99 was employed for optimization.

Table VII lists the parameter setting for the two-layer fully connected network LeNet300-100. We list the number of parameters and FLOPs on each layer before and after compression. For example, for the first layer of LeNet300-100, the weight matrix is 784×300. By applying our TRD, the matrix is reshaped to a high order tensor $(4 \times 4 \times 7 \times 7) \times (3 \times 4 \times 5 \times 5)$, which leads to the number of parameters being $39R^2$, and FLOPs being $2168R^2 + 2350R^3$.

As LeNet300-100 is a two-layer fully connected network, we provide the comparisons of our PFTRD with fully connected layer compression algorithms including MF, TTD-FC, and TRD. Table VIII lists the compression results. As shown in the table, we present the number of parameters (#params), parameter compression ratio (CR_P), the complexity of computation in terms of FLOPs (FLOPs), speedup ratio (speedup), memory consumption (Mem_usage) and *Top-k*

TABLE VII
LENET300-100 COMPRESSION

	Uncompressed LeNet300-100			PFTRD-compressed LeNet300-100		
Layer	Size	#params	FLOPs	Size	#params	FLOPs
1	784×300	235.2K	470.4K	$(4 \times 4 \times 7 \times 7) \times (4 \times 3 \times 5 \times 5)$	$39R^2$	$2168R^2 + 2350R^3$
2	300×100	30K	60K	$(4 \times 3 \times 5 \times 5) \times (4 \times 5 \times 5)$	$31R^2$	$800R^2 + 910R^3$
3	100×10	1K	2K	$(4 \times 5 \times 5) \times (2 \times 5)$	$21R^2$	$220R^2 + 260R^3$
Total	-	266.2K	532.4K	-	$91R^2$	$3188R^2 + 3520R^3$

TABLE VIII
LENET300-100 COMPRESSION RESULTS

Method	#params	CR_P	FLOPs	Speed-up	Mem_usage	Accuracy% (k=1)	Accuracy% (k=3)	Accuracy% (k=5)
Uncompressed	266.61K	$1\times$	104467195	$1\times$	62.18MB	97.34	99.81	99.98
MF(R=2)	3.6k	$74\times$	1304503	$80\times$	27.93 MB	80.36	90.45	98.72
MF(R=5)	8.38k	$31.8\times$	3179047	$32.8\times$	28.5 MB	90.72	95.76	99.12
MF(R=10)	16.35k	$16.3\times$	6303287	$16.5\times$	29.62 MB	93.45	98.38	99.33
TTD-FC(R=2)	0.75k	$353.6\times$	107420895	$0.97\times$	82.3 MB	86.64	92.53	98.87
TTD-FC(R=5)	2.18k	$122.3\times$	120561315	$0.86\times$	86.19 MB	92.13	96.69	99.29
TTD-FC(R=10)	5.73k	$46.5\times$	143178135	$0.73\times$	92.13 MB	94.96	98.95	99.45
TRD(R=2)	0.79k	$337.4\times$	2623983	$39.8\times$	58.86 MB	89.33	93.67	99.09
TRD(R=5)	2.79k	$95.5\times$	17211513	$6\times$	59.95 MB	94.6	97.71	99.54
TRD(R=10)	9.91k	$26.9\times$	76022463	$1.3\times$	67.43 MB	95.78	99.25	99.61
PFTRD(R=2)	0.77k	$344.5\times$	2590639	$40.3\times$	58.62 MB	90.09	93.86	99.14
PFTRD(R=5)	2.69k	$99.1\times$	16695463	$6.3\times$	59.77 MB	95.25	98.74	99.76
PFTRD(R=10)	9.51k	$28\times$	71897263	$1.45\times$	67.34 MB	96.18	99.52	99.8

TABLE IX
LENET5 COMPRESSION

	Uncompressed LeNet-5			PFTRD-compressed LeNet-5		
Layer	Size	#params	FLOPs	Size	#params	FLOPs
1	$5 \times 5 \times 1 \times 32$	0.8K	1254.4K	$5 \times 5 \times 1 \times (4 \times 4 \times 2)$	$21R^2$	$51744R^2 + 39280R^3$
2	$5 \times 5 \times 32 \times 64$	51.2K	20070.4K	$5 \times 5 \times (4 \times 4 \times 2) \times (4 \times 4 \times 4)$	$32R^2$	$75264R^2 + 10040R^3$
3	3136×1024	3211.3K	6422.6K	$(4 \times 4 \times 4 \times 7 \times 7) \times (4 \times 4 \times 4 \times 4 \times 4)$	$46R^2$	$8320R^2 + 8770R^3$
4	1024×10	10.2K	20.5K	$(4 \times 4 \times 4 \times 4 \times 4) \times (2 \times 5)$	$27R^2$	$2068R^2 + 2260R^3$
Total	-	3273.5K	27767.9K	-	$126R^2$	$137396R^2 + 60350R^3$

accuracy (Accuracy% ($k = 1$), Accuracy% ($k = 3$), and Accuracy% ($k = 5$)] under different ranks.

Top-k accuracy is a metric used in machine learning for evaluating models that measures whether the true label of an observation is in the top k predictions (classes) from the model. As can be seen from Table VIII, the larger the value of k, the higher the probability of the true label being included. For example, the *top-1* accuracy, *top-3* accuracy, and *top-5* accuracy of our PFTRD with $R = 5$ are 95.25, 98.74, and 99.76, respectively. In our application scenario, the exact rank of the prediction is not only our concern, we also care about the model's ability to narrow down the possible correct classes. Thus, in subsequent experiments of LeNet5 and VGG16 models, we only listed the accuracy when $k = 3$. In the following articles, accuracy refers to the accuracy when $k = 3$.

As expected, in the compressed network, with the increase of the rank, the parameters, FLOPs, memory consumption, and accuracy increase, and the compression ratio and speedup ratio decrease. By setting $R = 5$, our PFTRD achieves a very good performance, with a compression ratio of up to $99.1\times$, a speedup of $6.3\times$, a memory consumption of 59.77 MB, and a high accuracy of 98.74%.

b) LeNet5 compression on FASHION-MNIST: The model was trained for 30 epochs using a minibatch size of 128, and the Adam optimizer with a learning rate of 0.001 was employed for optimization.

Table IX lists the number of parameters and the number of FLOPs on each layer before and after compression for LeNet5. The compression seems to be more effective since LeNet5 is a deeper neural network with more parameters.

We provide the comparisons of our PFTRD with TTD and TRD. Since LeNet5 is a network consisting of convolutional layers and fully connected layers, we do not compare MF and TTD-FC as they cannot compress convolutional layers. Like Table VIII, we list compression results under different ranks on LeNet5 in Table X. By setting $R = 4$, our PFTRD achieves a compression ratio of $1039.5\times$, a speedup of $5.827\times$, a memory consumption of 748.25, and an accuracy of 96.83. To achieve a similar high accuracy, the parameters, FLOPs, and memory consumption of TTD and TRD are much larger than those of our PFTRD. When TTD achieves an accuracy of 96.25, the parameter, FLOPs, and memory consumption are 1520, 466 858 824 488, and 772.9 larger than those of our PFTRD, respectively.

c) VGG16 compression on CIFAR-100: The model was trained for 30 epochs using a mini-batch size of 64, and the Adam optimizer with a learning rate of 0.001 was employed for optimization.

TABLE X
LENET5 COMPRESSION RESULTS

Method	#params	CR_P	FLOPs	Speed-up	Mem_usage (MB)	Accuracy%
Uncompressed	3274.634K	1×	5367979184	1×	852.73	98.74
TTD(r=2)	1.62k	2021.4×	7374601364	0.727×	1427.33	92.75
TTD(r=4)	2.88k	1137×	7489186684	0.716×	1509.76	94.87
TTD(r=6)	4.67k	701.2×	7557472820	0.710×	1521.15	96.25
TRD(r=2)	1.7k	1926.2×	162696988	32.993×	252.22	93.26
TRD(r=4)	3.4k	963.1×	922477852	5.819×	748.58	95.69
TRD(r=6)	6.24k	524.7×	2743924124	1.956×	1680.79	97.4
PFTRD(r=2)	1.63k	2008.9×	162533708	33.026×	251.85	94.25
PFTRD(r=4)	3.15k	1039.5×	921173660	5.827×	748.25	96.83
PFTRD(r=6)	5.67k	577.5×	2739524780	1.959×	1679.34	98.29

TABLE XI
VGG16 COMPRESSION

	Uncompressed VGG16			PFTRD-compressed VGG16		
Layer	Size	#params	FLOPs	Size	#params	FLOPs
block1	$3\times3\times3\times64$	1.73K	3538.9K	$3\times3\times3\times(4\times4\times4)$	$24R^2$	$137216R^2+18512R^3$
	$3\times3\times64\times64$	36.86K	75497.5K	$3\times3\times(4\times4\times4)\times(4\times4\times4)$	$30R^2$	$262144R^2+18752R^3$
block2	$3\times3\times64\times128$	73.73K	37748.7K	$3\times3\times(4\times4\times4)\times(4\times4\times4\times2)$	$32R^2$	$196608R^2+5072R^3$
	$3\times3\times128\times128$	147.46K	75497.5K	$3\times3\times(4\times4\times4\times2)\times(4\times4\times4\times2)$	$34R^2$	$131072R^2+5216R^3$
block3	$3\times3\times128\times256$	294.91K	37748.7K	$3\times3\times(4\times4\times4\times2)\times(4\times4\times4\times4)$	$36R^2$	$98304R^2+2032R^3$
	$3\times3\times256\times256$	589.82K	75497.5K	$3\times3\times(4\times4\times4\times4)\times(4\times4\times4\times4)$	$38R^2$	$65536R^2+2304R^3$
	$3\times3\times256\times256$	589.82K	75497.5K	$3\times3\times(4\times4\times4\times4)\times(4\times4\times4\times4)$	$38R^2$	$65536R^2+2304R^3$
block4	$3\times3\times256\times512$	1.18M	37748.7K	$3\times3\times(4\times4\times4\times4)\times(4\times4\times4\times4\times2)$	$40R^2$	$18112R^2+2016R^3$
	$3\times3\times512\times512$	2.36M	75497.5K	$3\times3\times(4\times4\times4\times4\times2)\times(4\times4\times4\times4\times2)$	$42R^2$	$32768R^2+2592R^3$
	$3\times3\times512\times512$	2.36M	75497.5K	$3\times3\times(4\times4\times4\times4\times2)\times(4\times4\times4\times4\times2)$	$42R^2$	$32768R^2+2592R^3$
block5	$3\times3\times512\times512$	2.36M	18874.4K	$3\times3\times(4\times4\times4\times4\times2)\times(4\times4\times4\times4\times2)$	$42R^2$	$20480R^2+2376R^3$
	$3\times3\times512\times512$	2.36M	18874.4K	$3\times3\times(4\times4\times4\times4\times2)\times(4\times4\times4\times4\times2)$	$42R^2$	$8192R^2+2376R^3$
	$3\times3\times512\times512$	2.36M	18874.4K	$3\times3\times(4\times4\times4\times4\times2)\times(4\times4\times4\times4\times2)$	$42R^2$	$8192R^2+2376R^3$
fc	512×4096	2.1M	4.2M	$(4\times4\times4\times4\times2)\times(4\times4\times4\times4\times4\times4)$	$42R^2$	$9216R^2+9664R^3$
	4096×4096	16.78M	33.56M	$(4\times4\times4\times4\times4\times4)\times(4\times4\times4\times4\times4\times4)$	$48R^2$	$16384R^2+17024R^3$
	4096×100	409.6K	819.2K	$(4\times4\times4\times4\times4\times4)\times(4\times5\times5)$	$38R^2$	$8392R^2+8752R^3$
Total	-	34M	1483.4M	-	$610R^2$	$1110920R^2+103960R^3$

TABLE XII
VGG16 COMPRESSION RESULTS

Method	#params	CR_P	FLOPs	Speed-up	Mem_usage (MB)	Accuracy%
Uncompressed	34.01M	1×	485505437094	1×	29284.71	67.01
TTD(r=2)	15.2k	2237.5×	486264657660	0.998×	25620.68	57.32
TTD(r=5)	27,17k	1251.8×	488079908920	0.994×	26689.41	61.47
TTD(r=10)	62.77k	541.8×	490342859378	0.99×	27226.61	64.98
TRD(r=2)	15.4k	2208.4×	3473088716	139.79×	3948.28	58.24
TRD(r=5)	30.57k	1112.5×	23498984390	20.66×	6127.76	62.79
TRD(r=10)	84.72k	401.4×	74264065676	6.53×	22733.26	65.35
PFTRD(r=2)	14.94k	2276.4×	3472129740	139.83×	3945.61	60.03
PFTRD(r=5)	27.69k	1228.2×	23484034890	20.67×	6123.43	64.62
PFTRD(r=10)	73.22k	464.5×	74144561676	6.54×	22716.6	66.45

Table XI lists the number of parameters and the number of FLOPs on each layer before and after compression for VGG16. The parameters on VGG16 before and after compression are $34M$ and $610R^2$, suggesting high compression potential.

Like the convolutional neural network LeNet5 above, we provide the comparisons of our PFTRD with TTD and TRD. We list compression results under different ranks on VGG16 in Table XII. By setting $R = 5$, our PFTRD achieves a compression ratio of 1228.2×, a speedup of 20.67×, a memory consumption of 6123.43 MB, and an accuracy of 65.62. To achieve a similar high accuracy, the parameters and FLOPs of TTD and TRD are much larger than that of our PFTRD. When TTD achieves the accuracy of 64.98, the parameters, FLOPs, and memory consumption are 35 080, 466 858 824 488, and 21 103.18 larger than that of our PFTRD, respectively.

2) Inference Time Visualization Results: Fig. 9(a)-(c) presents the visualization results of inference time for our PFTRD compared with other baselines in LeNet300-100, LeNet5, and VGG16, respectively.

As the figure shows, with the increase of the rank, the inference time also increases. This increasing trend aligns with the changes in FLOPs as shown in Tables VIII, X and XII, as FLOPs also increase with time. When achieving a similar

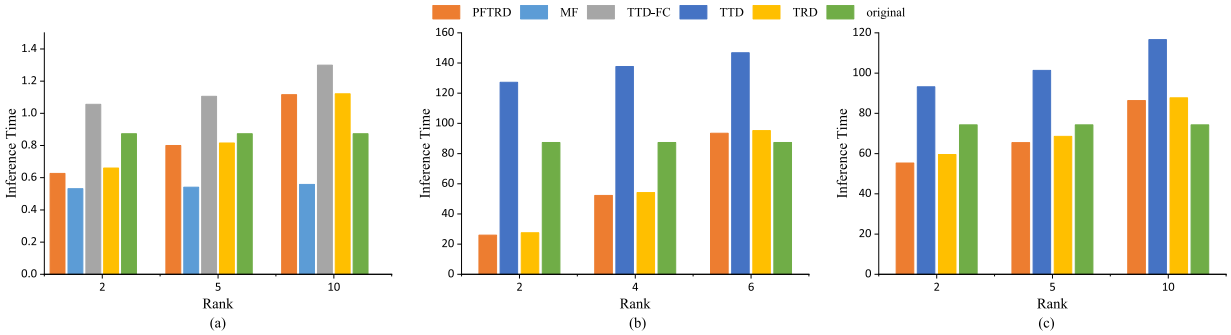


Fig. 9. Inference time of three models. (a) LeNet300-100 on MNIST. (b) LeNet5 on FASHION-MNIST. (c) VGG16 on CIFAR-100.

high accuracy, our PFTRD exhibits the shortest inference time compared with other baselines. For example, in LeNet5, when achieving an accuracy of around 96%, the inference time of our PFTRD is 52.26, while the inference times of the other two baselines are 146.84 and 54.17, respectively, both greater than ours. In Table X, when rank is 6, our PFTRD has smaller FLOPs than the uncompressed network, while the inference time in Fig. 9(b) is slightly longer than that of the uncompressed network. This is because FLOPs measure the theoretical computational complexity, whereas the actual speed of the model is influenced by other factors such as parallelism [52], [54], [55], [56]. How to exploit system's architecture to achieve fast speed mapping at small complexity is an open issue [57], [58]. In order to show the open issue, unlike the other neural network compression studies [24], [52], [59] which do not list inference time, we list the inference time in this article.

In sum, the experimental results demonstrate that PFTRD can achieve high accuracy with the lower computational complexity using the lowest memory cost.

VII. Conclusion

This article proposes a neural network compression algorithm based on the newly emerged TRD. To compress the neural network through TRD, we should first reshape the parameters into a tensor form, and then perform TRD on the reshaped tensor. We discover that both the procedure of tensor reshaping and the TRD impact the parameter compression performance. To maximally compress the parameters thus achieving the lowest memory consumption, we propose an algorithm based on prime factorization to simultaneously identify the optimal tensor reshaping and the TRD. Discovering that different execution orders of the core tensors brings different computational complexities, to identify the optimal execution order, we build a novel tree structure, based on which, we propose a top-to-bottom splitting algorithm to schedule the execution of core tensors to minimize the computational complexity. Extensive experimental results on three neural network models demonstrate the high effectiveness of our proposed scheme to achieve low memory consumption at the lowest computational complexity.

Appendix

A. Proof of Theorem 1

Theorem 1: Suppose that there are two positive integers a and b greater than 2, we must have $a \times b > a + b$.

Proof: When $a = b$, to prove $a \times b > a + b$, we can transform the formula above to $a^2 - (a + a) > 0$, that is $a(a - 2) > 0$. Because $a = b > 2$, so the formula $a \times b > a + b$ always holds.

When $a \neq b$, $a \times b > a + b$ in the theorem can be written as $a \times b - (a + b) > 0$. Suppose that $a < b$, let $c = b - a > 0$, we can transform the formula above to $a(c + a) - (a + a + c) > 0$, that is $a^2 + (c - 2)a - c > 0$. According to the root formula of the quadratic equation in one variable, there are two solutions to this problem, which are $a = ((2 - c \pm (c^2 + 4)^{1/2})/2)$. One of the solutions $a = ((2 - c - (c^2 + 4)^{1/2})/2)$ can be ignored for it is negative. The other solution is $a = ((2 - c + (c^2 + 4)^{1/2})/2)$, the value range of $(c^2 + 4)^{1/2}$ is $\sqrt{c^2} < (c^2 + 4)^{1/2} < ((c + 2)^2)^{1/2}$, so the value range of $((2 - c + (c^2 + 4)^{1/2})/2)$ is $1 < ((2 - c + (c^2 + 4)^{1/2})/2) < 2$. It can be seen that when $a > 2$, the formula $a^2 + (c - 2)a - c > 0$ always holds. In other words, when $a > 2$, $a \times b - (a + b) > 0$ always holds, which completes the proof.

B. Proof of Theorem 2

Theorem 2: For a given integer N that satisfies $N = \prod_{i=1}^{n} N_i$, where N_i is the factor and $i = 1, 2, \ldots, n$, the sum of these factors $\sum_{i=1}^{n} N_i$ is the smallest when all the factors are prime factors.

Proof: Suppose there exists a factor N_j that is not a prime factor, then N_j can be expressed by the product of two factors (except 1 and itself), which is $N_j = a \times b$. Therefore, the sum of factors can be rewritten as $\sum_{i=1}^{j-1} N_i + \sum_{i=j+1}^{n} N_i + N_j = \sum_{i=1}^{j-1} N_i + \sum_{i=j+1}^{n} N_i + a + b$.

When $a = b = 2$, we have that $a + b = a \times b$, and $\sum_{i=1}^{j-1} N_i + \sum_{i=j+1}^{n} N_i + a + b = \sum_{i=1}^{j-1} N_i + \sum_{i=j+1}^{n} N_i + a \times b = \sum_{i=1}^{n} N_i$.

When $a = 2$ and $b > 2$, we have that $a \times b - (a + b) = 2b - a - b = b - a > 0$, and $\sum_{i=1}^{i-1} N_i + \sum_{i=i+1}^{n} N_i + a + b < \sum_{i=1}^{j-1} N_i + \sum_{i=j+1}^{n} N_i + a \times b = \sum_{i=1}^{n} N_i$. The same is true when $b = 2$ and $a > 2$.

When $a > 2$ and $b > 2$, based on Theorem 1, we have that a and b satisfy $a + b < a \times b$, and $\sum_{i=1}^{i-1} N_i + \sum_{i=i+1}^{n} N_i + a + b < \sum_{i=1}^{j-1} N_i + \sum_{i=j+1}^{n} N_i + a \times b = \sum_{i=1}^{n} N_i$.

It can be seen that as long as there exists a nonprime factor, it can be further represented by other prime factors, thereby reducing the sum of factors. Then, when the factors are all prime factors, the sum of the factors is the smallest.

C. Proof of Theorem 3

Theorem 3: Suppose that there are two positive integers a and b greater than 2. When the product of a and b is a fixed

constant, the smaller the difference $|a-b|$ between the two integers, the smaller the sum $a+b$ of the two numbers.

Proof: Suppose $a \times b = c$ is a fixed constant. Let $d = |a-b|$. We have $d^2 = (a-b)^2 = a^2+b^2-2ab = a^2+b^2-2c$. We further have $a^2+b^2 = d^2-2c$. As $2c$ is constant, obviously, we have that the smaller the difference $|a-b|$ between the two integers, the smaller the a^2+b^2. Moreover, as $(a+b)^2 = a^2+b^2+2ab = a^2+b^2+2c$, the proof is completed.

D. Proof of Theorem 4

Theorem 4: For m TR core tensors $\mathcal{W}^i \in \mathbb{R}^{R\times I_i\times R}, i = 1,2,\ldots,m$ that satisfy $\prod_{i=1}^{m} I_i = I$, to recover the tensor $\mathcal{W}^I \in \mathbb{R}^{R\times I_1\times I_2\times\cdots\times I_m\times R}$ following the contraction order obtained by our tree splitting algorithm, the computational complexity in terms of the number of FLOPs is between $2R^3I$ and $4R^3I$.

Proof: Suppose $I_1 = I_2 = \cdots = I_m = \bar{I} \geq 2$ and the number of leaf nodes $m \geq 2$. The FLOPs of the root node are always $2R^3I = 2R^3\bar{I}^m$. If $m = 2$, there are only two leaf nodes with one contraction operation in the whole tree. Obviously, the computational cost of the whole tree is $2R^3I$. If $m > 2$ and only one of the two child nodes is a leaf node, there is a recursion $2R^3I = 2R^3\bar{I}^m = 2R^3\bar{I}\bar{I}^{m-1} \geq 4R^3\bar{I}^{m-1}$. If $m > 2$ and none of the two child nodes is a leaf node, the recursion is $2R^3I = 2R^3\bar{I}^m = 2R^3\bar{I}^{m1}\bar{I}^{m2} \geq 4R^3(\bar{I}^{m_1}+\bar{I}^{m_2}), m1, m2 \geq 2$, and $m_1+m_2 = m$. Since the number of FLOPs at each parent node is at least two times that at the child node, the total number of FLOPs must always be between $2R^3I$ and $4R^3I$. The proof is completed.

REFERENCES

[1] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 160–167.

[2] A. Graves, A.-R. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, May 2013, pp. 6645–6649.

[3] W. Wang et al., "Topic compositional neural language model," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2018, pp. 356–365.

[4] L. C. Chen, G. Papandreou, and I. Kokkinos, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, Jun. 2017.

[5] Y. Yang, D. Krompass, and V. Tresp, "Tensor-train recurrent neural networks for video classification," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3891–3900.

[6] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, "Predicting parameters in deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 26, 2013, pp. 2148–2156.

[7] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6655–6659.

[8] H. Zhang and V. M. Patel, "Convolutional sparse and low-rank coding-based image decomposition," *IEEE Trans. Image Process.*, vol. 27, no. 5, pp. 2121–2133, May 2018.

[9] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," in *Proc. Interspeech*, 2013, pp. 2365–2369.

[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. 2015.

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

[12] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 3, pp. 568–579, Mar. 2020.

[13] J. Qiu, C. Chen, S. Liu, H.-Y. Zhang, and B. Zeng, "SlimConv: Reducing channel redundancy in convolutional neural networks by features recombining," *IEEE Trans. Image Process.*, vol. 30, pp. 6434–6445, 2021.

[14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[15] C. Blakeney, X. Li, Y. Yan, and Z. Zong, "Parallel blockwise knowledge distillation for deep neural network compression," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1765–1776, Jul. 2021.

[16] Y. Idelbayev and M. Á. Carreira-Perpiñán, "Low-rank compression of neural nets: Learning the rank of each layer," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 8046–8056.

[17] S. E. Sofuoglu and S. Aviyente, "Multi-branch tensor network structure for tensor-train discriminant analysis," *IEEE Trans. Image Process.*, vol. 30, pp. 8926–8938, 2021.

[18] 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*.

[19] K. Xie et al., "Accurate recovery of Internet traffic data under variable rate measurements," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1137–1150, Jun. 2018.

[20] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[21] X. Li et al., "A light-weight and robust tensor convolutional autoencoder for anomaly detection," *IEEE Trans. Knowl. Data Eng.*, early access, pp. 1–14, 2023.

[22] K. Xie et al., "Fast tensor factorization for accurate Internet anomaly detection," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3794–3807, Dec. 2017.

[23] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki, "Tensor ring decomposition," 2016, *arXiv:1606.05535*.

[24] 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.

[25] S. Chen, J. Zhou, W. Sun, and L. Huang, "Joint matrix decomposition for deep convolutional neural networks compression," *Neurocomputing*, vol. 516, pp. 11–26, Jan. 2023.

[26] C. Faloutsos, T. G. Kolda, and J. Sun, "Mining large graphs and streams using matrix and tensor tools," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2007, p. 1174.

[27] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, Jan. 2011.

[28] J. Sokolic, R. Giryes, G. Sapiro, and M. R. D. Rodrigues, "Generalization error of deep neural networks: Role of classification margin and data structure," in *Proc. Int. Conf. Sampling Theory Appl. (SampTA)*, Jul. 2017, pp. 147–151.

[29] 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*.

[30] Y. Liu and M. K. Ng, "Deep neural network compression by tucker decomposition with nonlinear response," *Knowl.-Based Syst.*, vol. 241, Apr. 2022, Art. no. 108171, doi: 10.1016/j.knosys.2022.108171.

[31] M. Yin, Y. Sui, S. Liao, and B. Yuan, "Towards efficient tensor decomposition-based DNN model compression with optimization framework," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 10669–10678.

[32] B. Wu, D. Wang, G. Zhao, L. Deng, and G. Li, "Hybrid tensor decomposition in neural network compression," *Neural Netw.*, vol. 132, pp. 309–320, Dec. 2020, doi: 10.1016/j.neunet.2020.09.006.

[33] D. Wang et al., "Kronecker CP decomposition with fast multiplication for compressing RNNs," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 5, pp. 2205–2219, May 2023.

[34] H. Li, Z. Lin, T. Ma, X. Zhao, A. Plaza, and W. J. Emery, "Hybrid fully connected tensorized compression network for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 61, pp. 1–16, 2023.

[35] S. Tan, Q. Li, L. Li, B. Li, and J. Huang, "STD-NET: Search of image steganalytic deep-learning architecture via hierarchical tensor decomposition," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 3, pp. 2657–2673, May 2023.

[36] Y. Song, Z. Gong, Y. Chen, and C. Li, "Tensor-based sparse Bayesian learning with intra-dimension correlation," *IEEE Trans. Signal Process.*, vol. 71, pp. 31–46, 2023.

[37] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, "Tensorizing neural networks," 2015, *arXiv:1509.06569*.

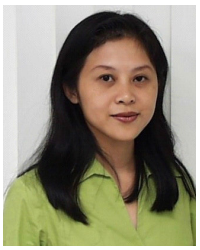
[38] A. Cichocki et al., "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Process. Mag.*, vol. 32, no. 2, pp. 145–163, Mar. 2015.
[39] Q. Zhao, M. Sugiyama, L. Yuan, and A. Cichocki, "Learning efficient tensor representations with ring-structured networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2019, pp. 8608–8612.
[40] Z. Long, C. Zhu, J. Liu, and Y. Liu, "Bayesian low rank tensor ring for image recovery," *IEEE Trans. Image Process.*, vol. 30, pp. 3568–3580, 2021.
[41] W. Wang, V. Aggarwal, and S. Aeron, "Efficient low rank tensor ring completion," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 5697–5705.
[42] V. Aggarwal, W. Wang, B. Eriksson, Y. Sun, and W. Wang, "Wide compression: Tensor ring nets," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9329–9338.
[43] J. Yu, G. Zhou, W. Sun, and S. Xie, "Robust to rank selection: Low-rank sparse tensor-ring completion," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 5, pp. 2451–2465, May 2023.
[44] L. Yuan, C. Li, D. Mandic, J. Cao, and Q. Zhao, "Tensor ring decomposition with rank minimization on latent space: An efficient approach for tensor completion," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 9151–9158.
[45] Y. Chen, W. He, N. Yokoya, T.-Z. Huang, and X.-L. Zhao, "Nonlocal tensor-ring decomposition for hyperspectral image denoising," *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 2, pp. 1348–1362, Oct. 2019.
[46] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[47] Y. LeCun. (2015). *LeNet-5, Convolutional Neural Networks*. [Online]. Available: http://yann.lecun.com/exdb/lenet
[48] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[49] Y. LeCun, C. Cortes, and C. J. Burges. (2010). *MNIST Handwritten Digit Database*. [Online]. Available: http://yann.lecun.com/exdb/mnist
[50] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
[51] A. Krizhevsky. (2009). *Learning Multiple Layers of Features From Tiny Images*. [Online]. Available: https://api.semanticscholar.org/CorpusID
[52] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, "Ultimate tensorization: Compressing convolutional and FC layers alike," 2016, *arXiv:1611.03214*.
[53] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*.
[54] A. Li and S. Su, "Accelerating binarized neural networks via bit-tensorcores in Turing GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1878–1891, Jul. 2021.
[55] N.-M. Ho and W.-F. Wong, "Tensorox: Accelerating GPU applications via neural approximation on unused tensor cores," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 2, pp. 429–443, Feb. 2022.
[56] Z. Cai et al., "TensorOpt: Exploring the tradeoffs in distributed DNN training with auto-parallelism," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 8, pp. 1967–1981, Aug. 2022.
[57] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015, *arXiv:1512.00567*.
[58] N. Ma, X. Zhang, H. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," 2018, *arXiv:1807.11164*.
[59] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy, "Sparse convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 806–814.

Kun Xie (Member, IEEE) received the Ph.D. degree in computer application from Hunan University, Changsha, China, in 2007.

She is currently a Professor with Hunan University. Her research interests include network monitoring, network security, matrix/tensor decomposition, and AI.

Can Liu is currently pursuing the Ph.D. degree with the College of Computer Science and Electronics Engineering, Hunan University, Changsha, China.

Her research interests include neural network compression.

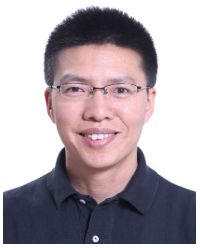


Xin Wang (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Columbia University, New York, NY, USA, in 2001.

She is currently an Associate Professor with the Department of Electrical and Computer Engineering, The State University of New York at Stony Brook, Stony Brook, NY, USA. Her current research interests include mobile and distributed computing, and networked sensing and detection.

Xiaocan Li received the Ph.D. degree from the College of Computer Science and Electronics Engineering, Hunan University, Changsha, China, in 2020.

He is currently an Associate Professor with the College of Computer Science and Electronics Engineering, Hunan University. His research interests include network measurement, network security, and matrix/tensor decomposition.

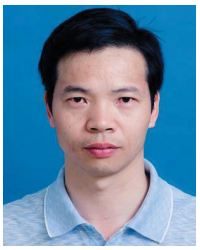


Gaogang Xie (Member, IEEE) received the B.S. degree in physics and the M.S. and Ph.D. degrees in computer science from Hunan University, Changsha, China, in 1996, 1999, and 2002, respectively.

He is currently a Professor with the Computer Network Information Center (CNIC), Chinese Academy of Sciences, Beijing, China; the University of Chinese Academy of Sciences, Beijing. His research interests include Internet architecture and forwarding.

Jigang Wen received the Ph.D. degree in computer application from Hunan University, Changsha, China, in 2011.

He is currently an Associate Professor with the School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, China. His research interests include wireless network and mobile computing, high-speed network measurement, and management.



Kenli Li (Senior Member, IEEE) received the Ph.D. degree in computer science from Huazhong University of Science and Technology, Wuhan, China, in 2003.

He is currently the Dean and a Full Professor of computer science and technology with Hunan University, Changsha, China, where he is also the Director of the National Supercomputing Center. His major research areas include parallel and distributed computing, and high-performance computing.