Convolutional neural network compression via tensor-train decomposition on permuted weight tensor with automatic rank determination

Mateusz Gabor and Rafał Zdunek

Faculty of Electronics, Photonics, and Microsystems,
Wroclaw University of Science and Technology,
Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
{mateusz.gabor, rafal.zdunek }@pwr.edu.pl

Abstract. Convolutional neural networks (CNNs) are among the most commonly investigated models in computer vision. Deep CNNs yield high computational performance, but their common issue is a large size. For solving this problem, it is necessary to find effective compression methods which can effectively reduce the size of the network, keeping the accuracy on a similar level. This study provides important insights into the field of CNNs compression, introducing a novel low-rank compression method based on tensor-train decomposition on a permuted kernel weight tensor with automatic rank determination. The proposed method is easy to implement, and it allows us to fine-tune neural networks from decomposed factors instead of learning them from scratch. The results of this study examined on various CNN architectures and two datasets demonstrated that the proposed method outperforms other CNNs compression methods with respect to parameter and FLOPS compression at a low drop in the classification accuracy.

Keywords: Neural network compression · Convolutional neural network · Tensor decomposition · Tensor train decomposition.

1 Introduction

The area of convolutional neural networks (CNNs) has attracted growing attention in the field of computer vision for achieving one of the best results in tasks such as image classification [11], segmentation [27] or object detection [26].

However, achieving better results of CNNs is mostly done by designing deeper neural networks, which translates into larger architectures requiring more space and more computing power. Because most of the deep neural networks are overparametrized [5], there exists a possibility of compressing them without reducing the quality of the network significantly. The neural network compression methods can be classified into weight sharing, pruning, knowledge distillation, quantization and low-rank approximations [22,1,16]. The weight sharing method is the simplest form of compressing a neural network size, in which the weights of

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

the neural network are shared between layers. From this approach, clustering-based weight sharing can be distinguished, in which the clustering is performed on weights, and at the end clustered weights are merged into new compressed weights. In the pruning approach, the redundant connections between neurons are removed, which results in a lower number of parameters and FLOPs. In most cases, the fine-tuning is necessary to recover the original accuracy of the network and often pruning/fine-tuning is alternately repeated in loop to gain larger compression. Quantization is another approach to compress neural network weights. In this method, the neural network weights are represented in a lower-precision format, the most popular is INT8, but the most extreme quantization is based on binary weights. On the other hand, knowledge distillation methods learn a small (student) network from a large one (teacher) using supervision. In short, a student network mimics a teacher network and leverages the knowledge of the teacher, achieving a similar or higher accuracy.

Besides the aforementioned methods, it is possible to compress the neural network using dimensionality reduction techniques such as matrix/tensor decompositions [24] in which the neural network weights are represented in a low-rank format. The low-rank compression methods can be divided into direct decomposition and tensorization. Direct decomposition methods use the factors obtained from the decomposition as new approximated weights, perform all operations on them, and are simple in implementation because they use basic convolutional neural network blocks from deep learning frameworks. The most popular two approaches of using the direct tensor decomposition to compress convolutional layers are the Tucker-2 [15] and CP [18] decomposition. The CP decomposition transforms the original weight tensor into a pipeline of two 1×1 convolutions and two depthwise separable convolutions, and the Tucker-2 into two 1×1 convolutions and one standard convolution, which is the same as the Bottleneck block in ResNet networks. Recently, Hameed *et al.* [9] proposed a new direct tensor decomposition method in which the Kronecker product decomposition is generalized to be applied to compress CNN weights. On the other hand, in the tensorization approach, the original weight tensor is tensorized into a higher-order tensor format and new weights are initialized randomly. In this approach, the decomposition algorithm is not used, and therefore the pretrained information from the baseline network is lost. By using tensorization, the achieved compression is relatively high, but the quality of the compressed network is significantly worse than the baseline model. The first tensorization approach to CNN compression was proposed by Garipov *et al.* [8], in which the tensor-train (TT) format was used to matricized weight tensor. The input feature maps tensor was reshaped into a matrix, and the convolution operation was performed as a sequence of tensor contractions. Garipov *et al.* also proposed a *naive* direct TT compression method in which the weight tensor was directly decomposed. All the decomposed cores were kept in memory, but during the convolution operation, the TT cores were reshaped into the original weight tensor, and the initialization was performed randomly. Among other methods of tensorization, one can mention the tensor ring format [21] or hierarchical Tucker format [31].

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

In this study, we propose a novel direct low-rank neural network compression method using direct tensor-train decomposition on the permuted kernel weight tensor with automatic rank determination. This method will be referred to as TTPWT. In our approach, each original convolutional layer is replaced and initialized with a sequence of four layers obtained from the decomposed factors, and the original convolution is approximated with four smaller convolutions, which is profitable both with respect to computational and storage complexity. The proposed compression method was applied to four neural networks: TT-conv-CNN [8], VGGnet [28], ResNet-56 [11] and ResNet-110 [11]. The experiments run on the CIFAR-10 and CIFAR-100 datasets showed that the TTPWT considerably outperforms many state-of-the-art compression methods with respect to parameter and FLOPS compression at a low drop in the classification accuracy.

The remainder of this paper is organized as follows. Section 2 presents the notation and the preliminaries to fundamental mathematical operations on tensors. It also contains a short description of the TT decomposition method. The proposed TT-based compression model is presented in Section 3. Numerical experiments performed using various CNN architectures tested on the CIFAR-10 and CIFAR-100 datasets are presented and discussed in Section 4. The final section provides concluding statements.

2 Preliminary

Notation: Multi-way arrays, matrices, vectors, and scalars are denoted by calligraphic uppercase letters (e.g., $\mathcal{X}$), boldface uppercase letters (e.g., $\boldsymbol{X}$), lowercase boldface letters (e.g., $\boldsymbol{x}$), and unbolded letters (e.g., x), respectively. Multi-way arrays will be equivalently referred to as tensors. We used Kolda's notation [17] for standard mathematical operations on tensors.

Mode-n unfolding: The mode-n unfolding of the N-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ rearranges its entries by placing its mode-n fibers as the columns of matrix $\boldsymbol{X}_{(n)} = [x_{i_n,j}] \in \mathbb{R}^{I_n \times \prod_{p \neq n} I_p}$ for $n \in \{1, \ldots, N\}$, where $j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) j_k$ with $j_k = \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m$, and $i_n = 1, \ldots, I_n$.

Mode-$\{n\}$ canonical matricization: This matricization reshapes tensor $\mathcal{X}$ into matrix $\boldsymbol{X}_{<n>} \in \mathbb{R}^{\prod_{p=1}^{n} I_p \times \prod_{r=n+1}^{N} I_r}$ by mapping tensor element $x_{i_1,\ldots,i_N}$ to matrix element $x_{i,j}$, where $i = 1 + \sum_{p=1}^{n} (i_p - 1) \prod_{m=1}^{p-1} I_m$ and $j = 1 + \sum_{r>n}^{N} (i_r - 1) \prod_{m=n+1}^{r-1} I_m$. The mode-$n$ unfolding is a particular case of the mode-$\{n\}$ canonical matricization.

Mode-n product (also known as the tensor-matrix product): The mode-n product of tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ with matrix $\boldsymbol{U} \in \mathbb{R}^{J \times I_n}$ is defined by

$$\mathcal{Z} = \mathcal{X} \times_n \boldsymbol{U}, \tag{1}$$

where $\mathcal{Z} = [z_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N}] \in \mathbb{R}^{I_1 \times \ldots \times I_{n-1} \times J \times I_{n+1} \times \ldots \times I_N}$, and

$$z_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N} = \sum_{i_n=1}^{I_n} x_{i_1,i_2,\ldots,i_N} u_{j,i_n}.$$

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

Tensor contraction: The tensor contraction of tensor $\mathcal{X} = [x_{i_1,\ldots,i_N}] \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ across its n-th mode with tensor $\mathcal{Y} = [y_{j_1,\ldots,j_M}] \in \mathbb{R}^{J_1 \times \ldots \times J_M}$ across its m-th mode, provided that $I_n = J_m$, gives tensor $\mathcal{Z} = \mathcal{X} \times_n^m \mathcal{Y}$ whose entries are given by:

$$z_{i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N,j_1,\ldots,j_{m-1},j_{m+1},\ldots,j_M} = \\ = \sum_{i_n=1}^{I_n} x_{i_1,\ldots,i_{n-1},i_n,i_{n+1},\ldots,i_N} y_{j_1,\ldots,j_{m-1},i_n,j_{m+1},\ldots,j_M}. \tag{2}$$

For the matrices: $\boldsymbol{A} \times_2^1 \boldsymbol{B} = \boldsymbol{A}\boldsymbol{B}$. The contraction: $\times_N^1$ will be denoted by the symbol $\bullet$. Thus: $\mathcal{X} \bullet \mathcal{Y} = \mathcal{X} \times_N^1 \mathcal{Y}$.

Kruskal convolution: Let $\mathcal{X} = [x_{i_1,i_2,c}] \in \mathbb{R}^{I_1 \times I_2 \times C}$ be any activation tensor in any convolutional layer with C input channels, $\mathcal{W} = [w_{t,c,d_1,d_2}] \in \mathbb{R}^{T \times C \times D_1 \times D_2}$ be the kernel weight tensor, Δ be the stride, and P be the zero-padding size. The Kruskal convolution maps input tensor $\mathcal{X}$ to output tensor $\mathcal{Y} = [y_{\tilde{i}_1,\tilde{i}_2,t}] \in \mathbb{R}^{\tilde{I}_1 \times \tilde{I}_2 \times T}$ by the following linear mapping:

$$y_{\tilde{i}_1,\tilde{i}_2,t} = x_{i_1,i_2,c} \star w_{t,c,d_1,d_2} = \sum_{c=1}^{C} \sum_{d_1=1}^{D} \sum_{d_2=1}^{D} w_{t,c,d_1,d_2} x_{i_1(d_1),i_2(d_2),c}, \tag{3}$$

where $i_1(d_1) = (\tilde{i}_1 - 1)\Delta + i_1 - P$ and $i_2(d_2) = (\tilde{i}_2 - 1)\Delta + i_2 - P$.

1×1 **convolution**: If $D = 1$, $\Delta = 1$, and $P = 0$, then $\mathcal{W} \in \mathbb{R}^{T \times C \times 1 \times 1}$, and the Kruskal convolution comes down to the 1×1 convolution: $y_{i_1,i_2,t} = \sum_{c=1}^{C} w_{t,c} x_{i_1,i_2,c}$. Using the notation of the mode-n product in (1), the 1×1 convolution takes the form:

$$\mathcal{Y} = \mathcal{X} \times_3 \boldsymbol{W}, \tag{4}$$

where $\boldsymbol{W} = [w_{tc}] \in \mathbb{R}^{T \times C}$.

Tensor train (TT) decomposition: The TT model [23] decomposes tensor $\mathcal{X} = [x_{i_1,\ldots,i_N}] \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ to a chain of smaller (3-way) core tensors that are connected by the tensor contraction with operator $\bullet$. It can be formulated as follows:

$$\mathcal{X} = \mathcal{X}^{(1)} \bullet \mathcal{X}^{(2)} \bullet \ldots \bullet \mathcal{X}^{(N)}, \tag{5}$$

where $\mathcal{X}^{(n)}$ is the n-th core tensor of size $R_{n-1} \times I_n \times R_n$ for $n = 1, \ldots, N$. The number $\{R_0, \ldots, R_N\}$ determine the TT ranks. Assuming $R_0 = R_N = 1$, we have $\mathcal{X}^{(1)} = \boldsymbol{X}^{(1)} \in \mathbb{R}^{I_1 \times R_1}$ and $\mathcal{X}^{(N)} = \boldsymbol{X}^{(N)} \in \mathbb{R}^{R_{N-1} \times I_N}$, i.e. the first and the last core tensors become matrices. Model (5) can be expressed equivalently as:

$$x_{i_1,\ldots,i_N} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_{N-1}=1}^{R_{N-1}} x^{(1)}_{i_1,r_1} x^{(2)}_{r_1,i_2,r_2} \cdots x^{(N-1)}_{r_{N-2},i_{N-1},r_{N-1}} x^{(N)}_{r_{N-1},i_N}, \tag{6}$$

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

where $\forall n : \mathcal{X}^{(n)} = [x^{(n)}_{r_{n-1},i_n,r_n}] \in \mathbb{R}^{R_{n-1}\times I_n \times R_n}$.

Assuming $I_1 = \ldots = I_N = I$ and $R_1 = \ldots = R_N = R$, the storage complexities of the CANDECOM/PARAFAC (CP) [2,10], Tucker [29], and TT decomposition models can be approximated by $\mathcal{O}(NIR)$, $\mathcal{O}(NIR + R^N)$, and $\mathcal{O}(NIR^2)$. It is thus obvious that the CP model has the lowest storage complexity, but its flexibility in adapting to the observed data is very low, especially for tensors that have strongly unbalanced modes. Unfortunately, this is the case in the discussed problem because two modes of the decomposed tensor have small dimensions, but the other modes are large. Hence, it is difficult to select the optimal rank. The Tucker decomposition relaxes these problems considerably, but its storage complexity grows up exponentially with the size of the core tensor, which is also not favorable in our case because the ranks for large modes are usually pretty large. The TT model assures the best trade-off between the CP and Tucker decompositions, alleviating the curse of dimensionality and yielding a flexible decomposition with multiple TT ranks. Hence, these advantages of the TT model motivate this study.

3 Proposed Method

We assume that each convolutional layer has C input and T output channels, and the size of the filter is $D \times D$. Hence, it can be represented by the kernel weight tensor $\mathcal{W} = [w_{t,c,d_1,d_2}] \in \mathbb{R}^{T\times C\times D\times D}$. The input data is represented by activation tensor $\mathcal{X} = [x_{i_1,i_2,c}] \in \mathbb{R}^{I_1\times I_2\times C}$ that consists of C activation maps – each has the resolution of $I_1 \times I_2$ pixels. Each layer performs a linear mapping of tensor $\mathcal{X}$ to output activation tensor $\mathcal{Y} = [y_{\tilde{i}_1,\tilde{i}_2,t}] \in \mathbb{R}^{\tilde{I}_1\times\tilde{I}_2\times T}$, where the mapping is determined by the Kruskal convolution in (3). Each output activation map has the resolution of $\tilde{I}_1 \times \tilde{I}_2$ pixels, and there are T output channels.

3.1 Model

To reduce the number of parameters and FLOPS in each convolutional layer, the kernel weight tensor $\mathcal{W}$ is decomposed with the TT model.

Remark 1. Note that if $\mathcal{W} \in \mathbb{R}^{T\times C\times D\times D}$ is decomposed according to (5), ranks R_2 and R_3 cannot be greater than D^2 and D, respectively. This restriction limits the flexibility of compression only to rank R_1. Furthermore, the 3D core tensor capturing the second mode of $\mathcal{W}$ could not be processed with a simple 1×1 convolution. Thus, we propose to apply the circular permutation to $\mathcal{W}$ with one left shift lag. Thus:

$$\tilde{\mathcal{W}} = \texttt{circular\_permutation}\,(\mathcal{W}, -1) \in \mathbb{R}^{C\times D\times D\times T}. \tag{7}$$

Applying the TT decomposition to $\tilde{\mathcal{W}} = [\tilde{w}_{c,d_1,d_2,t}]$, we have:

$$\tilde{w}_{c,d_1,d_2,t} = \sum_{r_1=1}^{R_1}\sum_{r_2=1}^{R_2}\sum_{r_3=1}^{R_3} \tilde{w}^{(1)}_{c,r_1}\tilde{w}^{(2)}_{r_1,d_1,r_2}\tilde{w}^{(3)}_{r_2,d_2,r_3}\tilde{w}^{(4)}_{r_3,t}. \tag{8}$$

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

Inserting model (8) to mapping (3) and rearranging the summands, we get:

$$
\begin{aligned}
y_{\tilde{i}_1,\tilde{i}_2,t} &= \sum_{c=1}^{C}\sum_{d_1=1}^{D}\sum_{d_2=1}^{D}\sum_{r_1=1}^{R_1}\sum_{r_2=1}^{R_2}\sum_{r_3=1}^{R_3} \tilde{w}^{(1)}_{c,r_1}\tilde{w}^{(2)}_{r_1,d_1,r_2}\tilde{w}^{(3)}_{r_2,d_2,r_3}\tilde{w}^{(4)}_{r_3,t}x_{i_1(d_1),i_2(d_2),c} \\
&= \sum_{r_3=1}^{R_3}\tilde{w}^{(4)}_{r_3,t}\left[\sum_{d_2=1}^{D}\sum_{r_2=1}^{R_2}\tilde{w}^{(3)}_{r_2,d_2,r_3}\sum_{d_1=1}^{D}\sum_{r_1=1}^{R_1}\tilde{w}^{(2)}_{r_1,d_1,r_2}\right. \\
&\quad \times \left.\left(\underbrace{\sum_{c=1}^{C}\tilde{w}^{(1)}_{c,r_1}x_{i_1(d_1),i_2(d_2),c}}_{1\times 1 \text{ conv.}}\right)\right] \\
&= \sum_{r_3=1}^{R_3}\tilde{w}^{(4)}_{r_3,t}\left[\sum_{d_2=1}^{D}\sum_{r_2=1}^{R_2}\tilde{w}^{(3)}_{r_2,d_2,r_3}\left(\underbrace{\sum_{d_1=1}^{D}\sum_{r_1=1}^{R_1}\tilde{w}^{(2)}_{r_1,d_1,r_2}z_{i_1(d_1),i_2(d_2),r_1}}_{D_1\times 1 \text{ conv.}}\right)\right] \\
&= \sum_{r_3=1}^{R_3}\tilde{w}^{(4)}_{r_3,t}\left[\underbrace{\sum_{d_2=1}^{D}\sum_{r_2=1}^{R_2}\tilde{w}^{(3)}_{r_2,d_2,r_3}z^{(V)}_{\tilde{i}_1,i_2(d_2),r_2}}_{1\times D_2 \text{ conv.}}\right] = \underbrace{\sum_{r_3=1}^{R_3}\tilde{w}^{(4)}_{r_3,t}z^{(V,H)}_{\tilde{i}_1,\tilde{i}_2,r_3}}_{1\times 1 \text{ conv.}}
\end{aligned} \tag{9}
$$

It can be easy to note that z_{i_1,i_2,r_1} in (9) can be computed with the 1×1 convolution. According to (4), we have:

$$\mathcal{Z} = \mathcal{X}\times_3 \tilde{\boldsymbol{W}}^{(1)T} \in \mathbb{R}^{I_1\times I_2\times R_1}, \tag{10}$$

where $\tilde{\boldsymbol{W}}^{(1)} = [\tilde{w}^{(1)}_{c,r_1}] \in \mathbb{R}^{C\times R_1}$. Physically, to perform operation (10), the first sublayer with the 1×1 convolutions in the analyzed convolutional layer is created. The activation tensor $\mathcal{Z}$ computed in the first sub-layer is then provided to the second convolutional sublayer represented by $\tilde{\mathcal{W}} = [\tilde{w}^{(2)}_{r_1,d_1,r_2}] \in \mathbb{R}^{R_1\times D\times R_2}$, which is much smaller than $\mathcal{W}$, and this sublayer computes the 1D convolutions along the 1-st mode (vertically):

$$z^{(V)}_{\tilde{i}_1,i_2(d_2),r_2} = \sum_{d_1=1}^{D}\sum_{r_1=1}^{R_1}\tilde{w}^{(2)}_{r_1,d_1,r_2}z_{i_1(d_1),i_2(d_2),r_1} \tag{11}$$

As a result, we get the second-sublayer output activation tensor

$$\mathcal{Z}^{(V)} = [z^{(V)}_{\tilde{i}_1,i_2(d_2),r_2}] \in \mathbb{R}^{\tilde{I}_1\times I_2\times R_2}.$$

Next, the third 1D convolutional sublayer is created to compute the 1D convolutions along the horizontal direction. The output activation tensor obtained from

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

this sublayer has the form: $\mathcal{Z}^{(V,H)} = [z^{(V,H)}_{\tilde{i}_1,\tilde{i}_2,r_3}] \in \mathbb{R}^{\tilde{I}_1 \times \tilde{I}_2 \times R_3}$. Finally, the fourth sublayer is created, which performs 1×1 convolutions according to the model:

$$\mathcal{Y} = \mathcal{Z}^{(V,H)} \times_3 \tilde{\boldsymbol{W}}^{(4)T} \in \mathbb{R}^{\tilde{I}_1 \times \tilde{I}_2 \times T}, \tag{12}$$

where $\tilde{\boldsymbol{W}}^{(4)} = [\tilde{w}^{(4)}_{r_3,t}] \in \mathbb{R}^{R_3 \times T}$.

3.2 TT decomposition algorithm

The TT decomposition of $\tilde{\mathcal{W}}$ in (7) can be obtained by using sequential SVD-based projections. In the first step, TSVD with a given precision δ_1 is applied to $\tilde{\mathcal{W}}$ unfolded with respect to its first-mode. Thus:

$$\tilde{\boldsymbol{W}}_{(1)} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T + \boldsymbol{E}_1, \tag{13}$$

under the assumption the truncation error satisfies the condition $||\boldsymbol{E}_1||_F \leq \delta_1$. Matrix $\tilde{\boldsymbol{W}}^{(1)} \in \mathbb{R}^{C \times R_1}$ is created from $\boldsymbol{U}$ that contains the first R_1 left singular vectors (associated with the most significant singular values) of $\tilde{\boldsymbol{W}}_{(1)}$. Note that rank R_1 is determined by a given threshold δ_1 for the truncation error. In the second step, $\tilde{\boldsymbol{W}}^{(2)} \in \mathbb{R}^{R_1 I_2 \times R_2}$ is created from the first R_2 left singular vectors of the matrix obtained by reshaping matrix $\boldsymbol{\Sigma}\boldsymbol{V}^T$ using the mode-2 canonical matricization. In this step, $||\boldsymbol{E}_2||_F \leq \delta_2$ and the core tensor is obtained by reshaping $\tilde{\boldsymbol{W}}^{(2)}$ accordingly. The similar procedure is applied in the third step, where $\tilde{\mathcal{W}}^{(3)} \in \mathbb{R}^{R_2 \times I_3 \times R_3}$ is created from the first R_3 singular vectors, and $\tilde{\boldsymbol{W}}^{(4)} \in \mathbb{R}^{R_3 \times T}$ is created from the scaled right singular vectors. Oseledets [23]

Algorithm 1 TT-SVD

Input : $\mathcal{W} \in \mathbb{R}^{T \times C \times D \times D}$ – input kernel weight tensor, τ - threshold
Output: $\{\mathcal{W}^{(1)}, ..., \mathcal{W}^{(4)}\}$ - estimated core tensors

Compute $\tilde{\mathcal{W}} \in \mathbb{R}^{C \times D \times D \times T}$ with (7) and set $R_0 = 1$ and $N = 4$,
$\boldsymbol{M} = \tilde{\boldsymbol{W}}_{(1)} = \texttt{unfolding}(\tilde{\mathcal{W}}, 1)$; `// Unfolding`
for $n = 1, \ldots, N-1$ **do**
  Compute: $\left[\tilde{\boldsymbol{U}}, \tilde{\boldsymbol{S}}, \tilde{\boldsymbol{V}}, R_n\right] = \texttt{TSVD}_\delta(\boldsymbol{M}, \tau)$; `// TSVD`
  $\tilde{\mathcal{W}}^{(n)} = \texttt{reshape}(\tilde{\boldsymbol{U}}, [R_{n-1}, I_n, R_n])$
  $\boldsymbol{M} = \texttt{reshape}(\tilde{\boldsymbol{S}}\tilde{\boldsymbol{V}}^T, [R_n I_{n+1}, \prod_{p=n+2}^{N} I_p])$; `// Canonical matricization`
end
$\tilde{\mathcal{W}}^{(4)} = \texttt{reshape}(\boldsymbol{M}, [R_{N-1}, I_N, 1])$

showed that $||\tilde{\mathcal{W}} - \tilde{\mathcal{W}}^{(1)} \bullet \ldots \bullet \tilde{\mathcal{W}}^{(N)}||_F \leq \sqrt{\sum_{n=1}^{N-1} \delta_n^2}$. Assuming $\delta = \delta_1 = \ldots =$

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

δ_{N-1}, then the truncation threshold can be set to $\delta = \frac{\epsilon}{\sqrt{N-1}}||\tilde{\mathcal{W}}||_F$, where $\epsilon > 0$ is a prescribed relative error.

In our approach, the optimal rank of TSVD for matrix $\boldsymbol{M} \in \mathbb{R}^{P\times R}$ in each step was computed by using the energy-threshold criterion. Thus:

$$R_* = \arg\min_j \left\{ \frac{\sum_{i=1}^{j} \sigma_i^2}{\sum_{i=1}^{I} \sigma_i^2} > \tau \right\}, \tag{14}$$

where $Q = \min\{P, Q\}$, σ_i is the i-th singular value of $\boldsymbol{M}$, and $\tau = \frac{\epsilon}{\sqrt{N-1}}$ is a given threshold. The energy captured by i components (singular vectors) is expressed in the nominator of (14), the total energy is presented in the denominator.

Due to the low-rank approximation, the TT model always assures the compression [25], i.e.

$$R_n \leq \min\left\{ \prod_{i=1}^{n} I_i, \prod_{j=n+1}^{N} I_j \right\}, \quad \text{for} \quad n = 1, \ldots, N-1. \tag{15}$$

The complete sequential routine is presented in Algorithm 1. Function $\texttt{TSVD}_\delta$ performs the δ-truncated SVD at a given threshold δ, where the optimal rank R_* is computed by the energy-based criterion (14).

3.3 Implementation

The procedure for training/fine-tuning networks was implemented in the deep learning framework *PyTorch* and the *tensor-train* decomposition in *Matlab*. The convolutional kernel is the main component of the convolutional layer, which is represented as the 4-th order tensor (top block, Figure 1). After using permutation, the weight tensor can be decomposed into four factors, including two matrices and two 3-rd order tensors. All the factors are used as new weights in a sequence of four sublayers (Tensor-Train model, Figure 1). Because the basic class of convolutional layer in *PyTorch* accepts only 4-th order tensor as weights, it is necessary to add extra two dimensions to matrices: $\tilde{\boldsymbol{W}}^{(1)} \in \mathbb{R}^{C\times R_1} \rightarrow \tilde{\mathcal{W}}^{(1)} \in \mathbb{R}^{R_1\times C\times 1\times 1}$ (Figure 1, sublayer Conv2D.1), $\tilde{\boldsymbol{W}}^{(4)} \in \mathbb{R}^{R_3\times T} \rightarrow \tilde{\mathcal{W}}^{(4)} \in \mathbb{R}^{T\times R_3\times 1\times 1}$ (Figure 1, sublayer Conv2D.4) and extra one dimension to 3-rd order tensors $\tilde{\mathcal{W}}^{(2)} \in \mathbb{R}^{R_1\times D\times R_2} \rightarrow \tilde{\mathcal{W}}^{(2)} \in \mathbb{R}^{R_2\times R_1\times D\times 1}$ (Figure 1, sublayer Conv2D.2), $\tilde{\mathcal{W}}^{(3)} \in \mathbb{R}^{R_3\times D\times R_2} \rightarrow \tilde{\mathcal{W}}^{(3)} \in \mathbb{R}^{R_3\times R_2\times 1\times D}$ (Figure 1, sublayer Conv2D.3) and permute the modes accordingly.

3.4 Computational complexity

The space and time complexity of the convolution is defined as $\mathcal{O}(CTD^2)$ and $\mathcal{O}(CTD^2 I_1 I_2)$. By applying the tensor-train decomposition, the time and space complexity is bounded by $\mathcal{O}(CR_1 + R_2 D(R_1 + R_3) + R_3 T)$ and $\mathcal{O}(R_1 I_1 I_2 + R_2 D(R_1 I_1 I_2 + R_3 \tilde{I}_1 I_2) + R_3 T \tilde{I}_1 \tilde{I}_2)$ respectively, where I_1 and I_2 define the height and width of the input image, respectively, and $\tilde{I}_1$ and $\tilde{I}_2$ define the reduced height and width after convolution.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

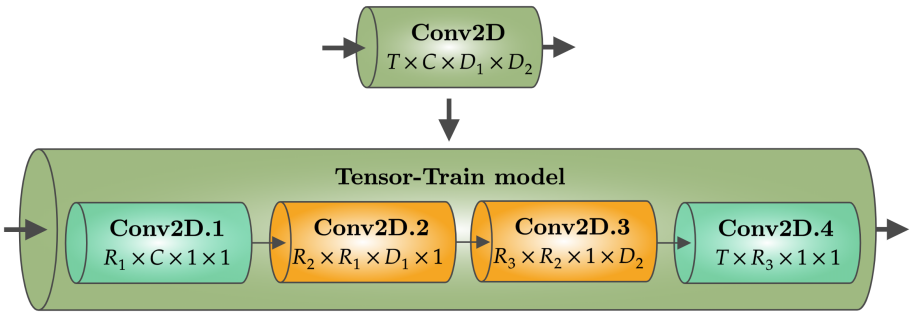

Fig. 1. Visual representation of how the decomposed factors are used as new weights in *PyTorch* framework (output channels × input channels × filter height × filter width) for the compressed convolutional layer in the TT model.

4 Results

We evaluated our method on two datasets (CIFAR-10 and CIFAR-100). Each consists of 60,000 examples, including 50,000 in the training dataset, and 10,000 in the validation dataset with 10 and 100 classes respectively. To evaluate effectiveness of our method on networks of various sizes, we selected the following networks: TT-conv-CNN [8], VGGnet [28], ResNet-56 [11] and ResNet-110 [11]. The networks cover the range of models with a medium to a large number of parameters and FLOPS. The total number of FLOPS and the parameters of the mentioned networks are listed in Table 1. The compression experiments were performed with the following scheme:

$$\text{energy threshold selection} \longrightarrow \text{baseline CNN compression} \longrightarrow \text{fine-tuning.}$$

All the convolutional layers were compressed in each neural network except for the first one whose size is small. All the baseline networks were trained according to the source guidelines. TT-conv-CNN was trained for 100 epochs using stochastic gradient descent (SGD) with a momentum of 0.9, the weight decay was set to 0, the initial learning rate was set to 0.1, and it was decreased by a factor of 0.1 after every 20 epochs. For the fine-tuning process, all the hyperparameters remained the same. VGGnet, ResNet-56, and ResNet-110 were trained for 200 epochs using the SGD with a momentum of 0.9, the weight decay was set to 10^{-4}, the initial learning rate was set to 0.1, and it was decreased by a factor of 0.1 after 80 and 120 epochs for VGGnet, and after 100 and 150 for ResNets. In the fine-tuning step, the learning rate was lowered to 0.01 and the weight decay was increased to 10^{-3} for the VGGnet, and the hyperparameters were unchanged for ResNet-56 and ResNet-110.

To evaluate the network compression and performance, we used two metrics, such as the parameter compression ratio (PCR) that is defined as $\Downarrow Param = \frac{\text{Param(baseline network)}}{\text{Param(compressed network)}}$, and the FLOPS compression ratio (FCR) defined as

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

$\Downarrow FLOPS = \frac{\text{FLOPS(baseline network)}}{\text{FLOPS(compressed network)}}$. The quality of the network was evaluated with the drop in the classification accuracy of the compressed network with respect to the baseline network, i.e. $\Delta Acc = Acc_{compressed} - Acc_{baseline}$. The error rate is often shown alongside with the accuracy in the literature. However, the error rate may be misleading since we fine-tune the neural networks from decomposed factors. Hence, the accuracy is sufficient to be shown for better interpretability of results. The values of PCR or FCR are not provided in all the papers, which we refer to as the reference results. Hence, the unavailable data are marked with the "—" sign in the tables.

Table 1. Total number of FLOPs and parameters for baseline networks.

Network	Params	FLOPS
TT-conv-CNN	558K	105M
ResNet-56	853K	125M
ResNet-110	1.73M	255M
VGGnet	20M	399M

4.1 CIFAR-10

TT-conv-CNN : Table 2 shows the results obtained for the TT-conv-CNN compression. We compared our method with the tensorized tensor-train version of the matricized weight tensor (TT-conv), direct tensorized tensor-train weight tensor (TT-conv (naive)), and the weight sharing method – Deep k-Means [32]. As we can see, our method outperforms both TT-based methods proposed by Garipov *et al.* in terms of PCR, FCR, and the drop in accuracy is at a much lower level. Compared with Deep k-Means, our method achieved better accuracy with higher compression.

VGGnet : The VGGnet network is a modified VGG-19 neural network adopted for CIFAR datasets. It is the largest neural network analyzed in this study, with 20M of parameters and 399M of FLOPS. Compression of VGGnet using our method was compared with the following pruning approaches: DCP [35], Random-DCP [35], WM+ [35], CP [14] and PFEC [19]. The results given in Table 3 demonstrate that our method outperforms all the compared approaches in terms of PCR and FCR. Our compressed network achieved a positive drop (gain) in the accuracy compared to the baseline network, and only DCP obtained a higher gain but with worse parameter compression. Moreover, TTPWT reduces FLOPS nearly 2.35 times more than DCP.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

Table 2. Results of the TT-conv-CNN [8] compression on the CIFAR-10 validation dataset. Different rows of the TT-conv and TT-conv (naive) mean different ranks. The value in parentheses denotes the energy threshold.

Method	ΔAcc	$\Downarrow$Param	$\Downarrow$FLOPS
TT-conv-1 [8]	−0.80	2.02	—
TT-conv-2 [8]	−1.50	2.53	—
TT-conv-3 [8]	−1.40	3.23	—
TT-conv-4 [8]	−2.00	4.02	—
TT-conv-1 (naive) [8]	−2.40	2.02	—
TT-conv-2 (naive) [8]	−3.10	2.90	—
Deep k-Means [32]	+0.05	2.00	—
TTPWT (0.6)	**+0.14**	3.06	2.95
TTPWT (0.5)	−0.25	**5.03**	**4.73**

Table 3. Results of VGGnet compression on the CIFAR-10 validation dataset. The value in parentheses denotes the energy threshold.

Method	ΔAcc	$\Downarrow$Param	$\Downarrow$FLOPS
DCP [35]	+0.31	1.93	2.00
Random-DCP [35]	+0.03	1.93	2.00
WM+ [35]	−0.10	1.93	2.00
CP [14,35]	−0.32	1.93	2.00
PFEC [19,35]	+0.15	2.78	1.52
TTPWT (0.6)	+0.15	**3.03**	**4.71**

4.2 CIFAR-100

ResNet-56 : ResNet-56 was the first network evaluated by us on the CIFAR-100 dataset. We compared the obtained results of our method with pruning approaches. As pruning competitors, we chose the following methods: SFP [12], FPGM [13], DMPP [20], CCPrune [4], FPC [3], and FPDC[36]. As can be seen in Table 4 our method achieved the largest FCR and PCR, and the lowest accuracy drop. It is interesting that TTPWT reduces FLOPS twice as much as SFP, FPGM and CCPrune.

ResNet-110 : As the second network, we selected ResNet-110 that is one of the largest ResNet networks developed for CIFAR datasets. Similar to the previous results, ResNet-110 was compared with different pruning approaches [30,36,33,34,7,13,12,6]. As shown in Table 5, it is clear that TTPWT achieved the lowest drop in accuracy and the largest PCR and FCR over all the compared methods.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

Table 4. Results of ResNet-56 compression on the CIFAR-100 validation dataset. The value in parentheses denotes the energy threshold.

Method	ΔAcc	$\Downarrow$Param	$\Downarrow$FLOPS
SFP [12,20]	−2.61	3.20	2.11
FPGM [13,20]	−1.75	3.30	2.11
CCPrune [4]	−0.63	1.69	2.94
FPDC [36]	−1.43	1.93	1.99
TTPWT (0.55)	**−0.50**	**3.93**	**4.19**

Table 5. Results of ResNet-110 compression on the CIFAR-100 validation dataset. The value in parentheses denotes the energy threshold.

Method	ΔAcc	$\Downarrow$Param	$\Downarrow$FLOPS
OED [30]	−3.83	2.31	3.23
FPDC [36]	−0.61	1.93	3.24
PKPSMIO [33]	−0.14	3.40	3.24
PKP [34]	−0.61	2.42	2.37
TAS [7]	−1.90	—	2.11
FPGM [13,7]	−1.59	—	2.10
SFP [12,7]	−2.86	—	2.10
LCCL [6,7]	−2.01	—	1.46
TTPWT (0.55)	**−0.03**	**3.96**	**4.21**

5 Conclusions

This study proposes a new approach to low-rank compression of CNNs. The proposed method is based the tensor train decomposition of a permuted weight tensor with automatic rank determination. The original convolution is approximated with a pipeline of four smaller convolutions, which allows us to significantly reduce a number of parameters and FLOPS at the cost of a low drop in accuracy. The results obtained on two datasets using four networks of different sizes confirm that our method outperforms the other neural network compression methods presented in this study. Further research is needed to investigate the compression of larger CNNs on the ImageNet dataset and to extend the current approach for higher order convolutional neural networks, including 3D CNNs.

References

1. Alqahtani, A., Xie, X., Jones, M.W.: Literature review of deep network compression. In: Informatics. vol. 8, p. 77. Multidisciplinary Digital Publishing Institute (2021)
2. Carroll, J.D., Chang, J.J.: Analysis of individual differences in multidimensional scaling via an n-way generalization of Eckart–Young decomposition. Psychometrika **35**, 283–319 (1970)

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

3. Chen, Y., Wen, X., Zhang, Y., He, Q.: Fpc: Filter pruning via the contribution of output feature map for deep convolutional neural networks acceleration. Knowledge-Based Systems **238**, 107876 (2022)
4. Chen, Y., Wen, X., Zhang, Y., Shi, W.: Ccprune: Collaborative channel pruning for learning compact convolutional networks. Neurocomputing **451**, 35–45 (2021)
5. Denil, M., Shakibi, B., Dinh, L., Ranzato, M., De Freitas, N.: Predicting parameters in deep learning. In: Advances in Neural Information Processing Systems (NIPS). pp. 2148–2156 (2013)
6. Dong, X., Huang, J., Yang, Y., Yan, S.: More is less: A more complicated network with less inference complexity. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5840–5848 (2017)
7. Dong, X., Yang, Y.: Network pruning via transformable architecture search. Advances in Neural Information Processing Systems **32** (2019)
8. Garipov, T., Podoprikhin, D., Novikov, A., Vetrov, D.: Ultimate tensorization: compressing convolutional and fc layers alike. [Online] arXiv preprint arXiv:1611.03214 (2016)
9. Hameed, M.G.A., Tahaei, M.S., Mosleh, A., Nia, V.P.: Convolutional neural network compression through generalized kronecker product decomposition. arXiv preprint arXiv:2109.14710 (2021)
10. Harshman, R.A.: PARAFAC2: Mathematical and technical notes. UCLA Working Papers in Phonetics **22**, 30–44 (1972)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)
12. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. In: International Joint Conference on Artificial Intelligence (IJCAI). pp. 2234–2240 (2018)
13. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4340–4349 (2019)
14. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision (CVPR). pp. 1389–1397 (2017)
15. Kim, Y.D., Park, E., Yoo, S., Choi, T., Yang, L., Shin, D.: Compression of deep convolutional neural networks for fast and low power mobile applications. In: International Conference on Learning Representations (ICLR) (2015)
16. Kirchhoffer, H., Haase, P., Samek, W., Müller, K., Rezazadegan-Tavakoli, H., Cricri, F., Aksu, E., Hannuksela, M.M., Jiang, W., Wang, W., Liu, S., Jain, S., Hamidi-Rad, S., Racapé, F., Bailer, W.: Overview of the neural network compression and representation (nnr) standard. IEEE Transactions on Circuits and Systems for Video Technology pp. 1–1 (2021). https://doi.org/10.1109/TCSVT.2021.3095970
17. Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM Review **51**(3), 455–500 (2009)
18. Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V.: Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In: International Conference on Learning Representations (ICLR) (2014)
19. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: International Conference on Learning Representations (ICLR) (2016)

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54

20. Li, J., Zhao, B., Liu, D.: DMPP: Differentiable multi-pruner and predictor for neural network pruning. Neural Networks **147**, 103–112 (2022)
21. Li, N., Pan, Y., Chen, Y., Ding, Z., Zhao, D., Xu, Z.: Heuristic rank selection with progressively searching tensor ring network. Complex & Intelligent Systems pp. 1–15 (2021)
22. Neill, J.O.: An overview of neural network compression. arXiv preprint arXiv:2006.03669 (2020)
23. Oseledets, I.V.: Tensor-train decomposition. SIAM Journal on Scientific Computing **33**(5), 2295–2317 (2011)
24. Panagakis, Y., Kossaifi, J., Chrysos, G.G., Oldfield, J., Nicolaou, M.A., Anandkumar, A., Zafeiriou, S.: Tensor methods in computer vision and deep learning. Proceedings of the IEEE **109**(5), 863–890 (2021)
25. Phan, A.H., Cichocki, A., Uschmajew, A., Tichavský, P., Luta, G., Mandic, D.P.: Tensor networks for latent variable analysis: Novel algorithms for tensor train approximation. IEEE Transactions on Neural Networks and Learning Systems **31**(11), 4622–4636 (2020)
26. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 779–788 (2016)
27. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI). pp. 234–241. Springer (2015)
28. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference Learning Representations (ICLR) (2015)
29. Tucker, L.R.: The extension of factor analysis to three-dimensional matrices. In: Gulliksen, H., Frederiksen, N. (eds.) Contributions to mathematical psychology., pp. 110–127. Holt, Rinehart and Winston, New York (1964)
30. Wang, Z., Lin, S., Xie, J., Lin, Y.: Pruning blocks for cnn compression and acceleration via online ensemble distillation. IEEE Access **7**, 175703–175716 (2019)
31. Wu, B., Wang, D., Zhao, G., Deng, L., Li, G.: Hybrid tensor decomposition in neural network compression. Neural Networks **132**, 309–320 (2020)
32. Wu, J., Wang, Y., Wu, Z., Wang, Z., Veeraraghavan, A., Lin, Y.: Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. In: International Conference on Machine Learning (ICML). pp. 5363–5372. PMLR (2018)
33. Zhu, J., Pei, J.: Progressive kernel pruning with saliency mapping of input-output channels. Neurocomputing **467**, 360–378 (2022)
34. Zhu, J., Zhao, Y., Pei, J.: Progressive kernel pruning based on the information mapping sparse index for cnn compression. IEEE Access **9**, 10974–10987 (2021)
35. Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., Zhu, J.: Discrimination-aware channel pruning for deep neural networks. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31 (NeurIPS). pp. 881–892. Curran Associates, Inc. (2018)
36. Zuo, Y., Chen, B., Shi, T., Sun, M.: Filter pruning without damaging networks capacity. IEEE Access **8**, 90924–90930 (2020)

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08757-8_54