Knowledge-Based Systems 255 (2022) 109723

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A novel compact design of convolutional layers with spatial transformation towards lower-rank representation for image classification

Baichen Liu [a,b,c], Zhi Han [a,b,*], Xi'ai Chen [a,b], Wenming Shao [d], Huidi Jia [a,b,c], Yanmei Wang [a,b,c], Yandong Tang [a,b]

[a] State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang, 110016, Liaoning, China
[b] Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang, 110169, China
[c] University of Chinese Academy of Sciences, Beijing, 100049, China
[d] College of Computer Science and Technology, China University of Petroleum, Qingdao, 266580, China

ARTICLE INFO

Article history:
Received 1 May 2022
Received in revised form 13 August 2022
Accepted 15 August 2022
Available online 20 August 2022

Keywords:
Neural network compression
Tucker decomposition
Spatial transformation
Image classification

ABSTRACT

Convolutional neural networks (CNNs) usually come with numerous parameters and thus are not convenient for some situations, such as when the storage space is limited. Low-rank decomposition is one effective way for network compression or compaction. However, the current methods are far from theoretical optimal compression performance because the low-rankness of the commonly trained convolution filter sets is limited because of the versatility of convolution filters. We propose a novel compact design for convolutional layers with spatial transformation for achieving a much lower-rank form. The convolution filters in our design are generated using a predefined Tucker product form, followed by learnable individual spatial transformations on each filter. The low-rank (Tucker) part lowers the parameter capacity while the transformation part enhances the feature representation capacity. We validate our proposed approach on an image classification task. Our approach focuses on compressing parameters while also improving accuracy. We perform experiments on the MNIST, CIFAR10, CIFAR100, and ImageNet datasets. On the ImageNet dataset, our approach outperforms low-rank based state-of-the-arts by 2% to 6% in top-1 validation accuracy. Furthermore, our approach outperforms a series of low-rank-based state-of-the-arts on various datasets. The experiments validate the efficacy of our proposed method. Our code is available at https://github.com/liubc17/low_rank_compact_transformed.

1. Introduction

Convolutional neural networks (CNNs) have recently demonstrated success in image classification [1–5], object detection [6–8] and remote sensing [9,10] tasks. The core operator, convolution, is partially inspired by the animal visual cortex, where different neurons respond to stimuli in a restricted and partially overlapped region known as the receptive field [11,12]. The window size of convolution filters becomes smaller from 11×11 [1] to 3×3 [13] along with the deeper CNNs. Stacks of convolutional layers with smaller window sizes have the same receptive field as convolutional layers with larger window sizes. Deeper models with a smaller window size of convolution filters (such as 3×3) have fewer parameters and benefit from more non-linear rectification layers. However, adding more layers to a suitable deep model increases training error and reduces accuracy [14–16]. Some frameworks use special connections between layers, such as shortcut connections [15] and dense connectivity [17], to solve the degradation problem. However, these networks still have millions of parameters and are not suitable for some situations where storage space is limited.

Some neural network compression methods [18–22] are proposed to address this issue and to obtain a light-weighted network with significantly fewer parameters. Two types of effective methods are low-rank decomposition based methods and compact design methods. Convolution filters are represented as matrices or tensors in low-rank decomposition methods [19,23–28], and the low-rank decomposition of these matrices or tensors can reduce the number of network parameters. However, the decomposition operation brings high computing costs, and the compressed network requires extensive model retraining to achieve convergence. The compact design methods [20,29–31]

* Corresponding author at: State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang, 110016, Liaoning, China.
E-mail address: hanzhi@sia.cn (Z. Han).

https://doi.org/10.1016/j.knosys.2022.109723

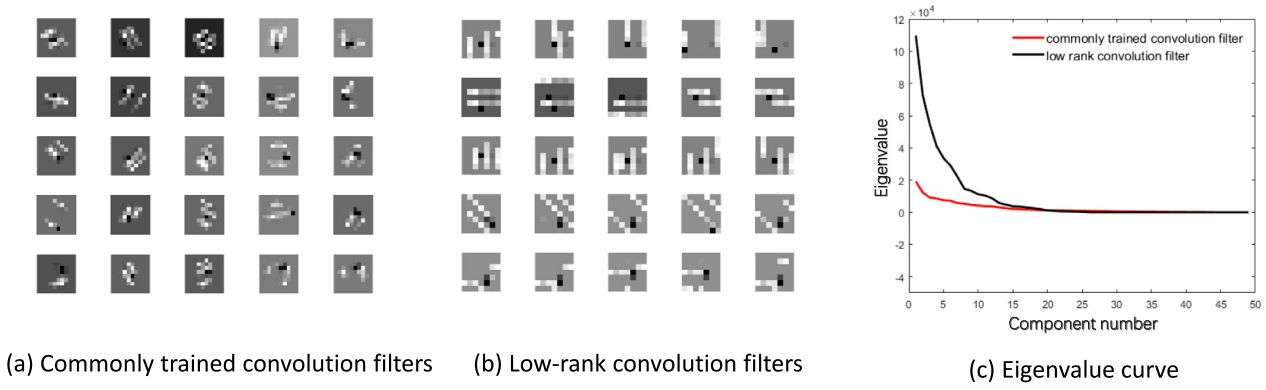

Fig. 1. (a) The commonly trained convolution filters of a convolutional layer. (b) The low-rank convolution filters after spatial transformation. (c) Eigenvalue curve of the commonly trained convolution filters and the low-rank convolution filters.

use transferred convolutional filters to compress CNN models, but the transfer assumptions are usually too strong to guide the algorithm and cause unstable results on some datasets [32].

Some studies have shown that the information of CNNs is redundant among different feature channels and filters [33], and there are some effective low-rank decomposition methods to address this redundancy. We further investigate the low-rankness of commonly trained convolution filters in Fig. 1. A series of commonly trained convolution filters of a well-trained convolutional layer is shown in Fig. 1a. We discover that commonly trained convolution filters have little in common and are extremely versatile. This allows CNN to extract more detailed features. However, the lack of similarity means a lack of low-rankness, which limits the compression rate of low-rank decomposition approaches for compressing CNNs. We discover that common trained convolution filters can have better low-rankness after appropriate spatial transformation. For example, in Fig. 1b, low-rank convolution filters are transformed via affine transformation from commonly trained convolution filters in Fig. 1a. These low-rank convolution filters have good similarity, which leads to better low-rankness. Therefore, the compression rate of low-rank decomposition methods to the low-rank convolution filters can be much higher than commonly trained convolution filters. The eigenvalue curve quantitatively validates the difference in similarity and low-rankness between commonly trained convolution filters and low-rank convolution filters, as shown in Fig. 1c. As a result, by applying an appropriate spatial transformation to commonly trained convolution filters, we can improve low-rankness and low-rank representation performance. However, finding the best spatial transformation for thousands of convolution filters is a difficult dynamic problem that is computationally expensive.

To address these concerns, we propose a compact CNN design (Fig. 2) to optimize this compression procedure. We define unfolded convolution filters in Tucker factor form to generate low-rank convolution filters. We apply a group of spatial transformations to the low-rank convolution filters and obtain versatile convolution filters used for the convolutional layer of compact design to retain and even promote the representation capacity of the low-rank convolution filters. Spatial transformations can be learned and come in various forms, such as rotation and affine transformation. It can be used in different hierarchies, such as 2D and 3D convolution filters. The network is thus end-to-end and can be trained from the ground up, avoiding the complex computation of low-rank decomposition. On an image classification task, we validate our low-rank compact transform (LCT) CNNs. The experiment results show that our LCT-designed CNNs can compress parameters while improving accuracy.

To summarize, the main contributions of this paper are as follows:

(1) We find that the redundancy of commonly trained convolution filters can be represented in a lower-rank tensor form using appropriate spatial transformation. We verify that the low-rankness of convolution filters can be improved significantly after the spatial transformation.

(2) We propose a novel compact design of convolutional layers with a spatial transformation that is simple to implement in popular CNNs. Our end-to-end LCT CNN can be trained from the ground up. Its number of parameters is greatly reduced, and its performance is improved when compared to baseline CNN.

2. Related work

We review related works of convolutional neural networks compression methods, including low-rank decomposition methods and compact design methods. In [24], Canonical Polyadic (CP) decomposition is proposed to compress the number of parameters and speed up the network. In [19], Tucker decomposition is proposed to compress convolutional layers for fast and low power mobile applications. Tensor-Train (TT) decomposition [26] is proposed and effective for solving dense connection problems to avoid the curse of dimensionality. However, these low-rank decomposition methods involve computationally expensive decomposition operations and have performance degradation. Jaderberg et al. [34] propose a linear combination of a smaller basis set of 2D separable filters to approximate the 2D filter set to speed up the evaluation of convolutional neural networks. However, this approach does not consider spatial transformation to convolution filters. Its representation capacity degrades compared with common convolution filters. In addition to low-rank decomposition approaches, knowledge distillation approaches [35,36], network pruning approaches [37–39], and neural network architecture searching approaches [40,41] are effective for compact structures.

Existing low-rank approaches mainly focus on compressing parameters directly by low-rank decomposition and do not consider the influence of spatial transformation on low-rankness. These approaches apply low-rank decomposition directly to well-trained weights and then fine-tune the compressed network for some epochs to recovery accuracy. However, even after fine-tuning, the classification accuracy of the network compressed by these approaches is lower than that of the original network. Different from existing low-rank approaches, we design an LCT layer to derive versatile convolution filters from unfolded convolution filters in low-rank Tucker factors form. Our approach focuses on compressing parameters while also improving network performance. The unfolded convolution filters can be used to form low-rank convolution filters, as shown in Fig. 1. In addition, we design spatial transformation to enrich the versatility of the low-rank convolution filters. The spatial transformation transforms

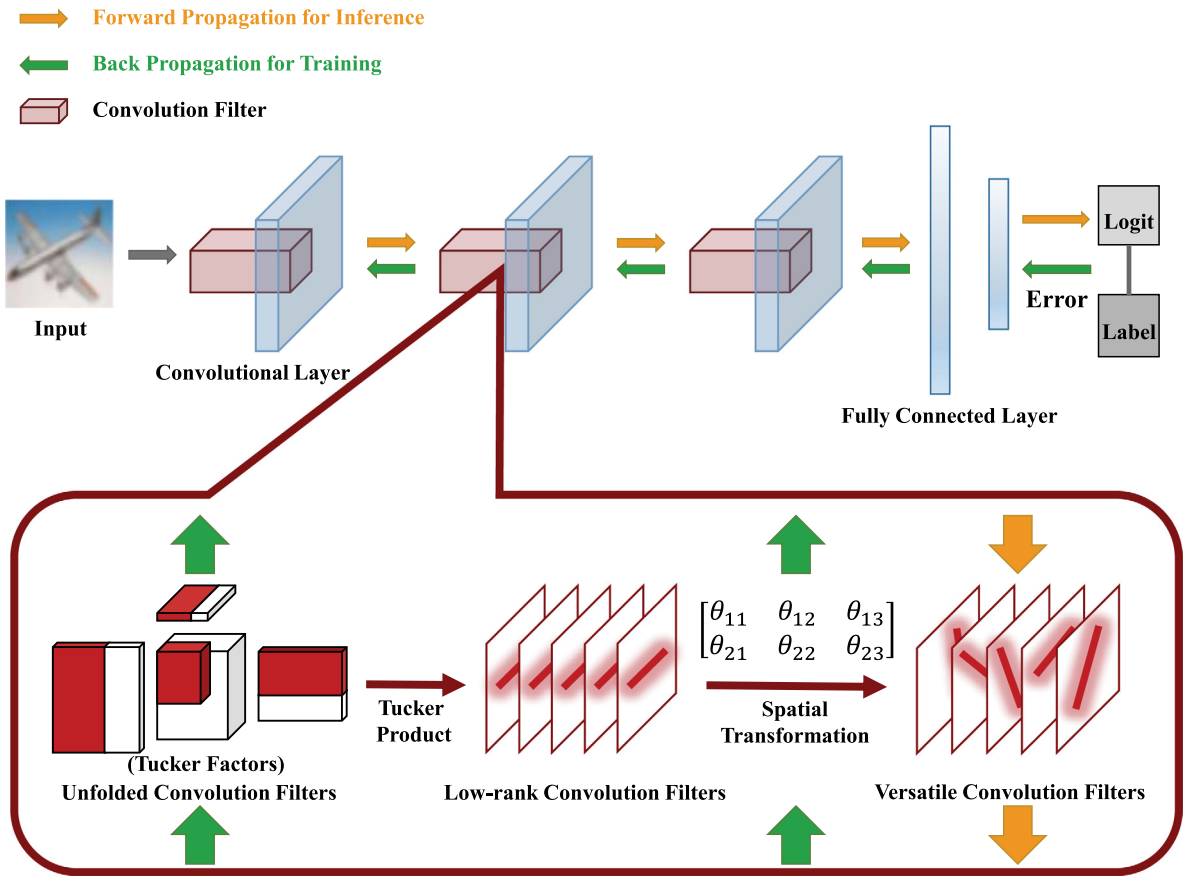

Fig. 2. The architecture of our compact design CNN. Convolution filters of the compact design convolutional layer are generated in two steps. Firstly, low-rank convolution filters are generated from unfolded convolution filters of a predefined Tucker factors form. Then, we apply appropriate spatial transformations to the low-rank convolution filters to obtain versatile convolution filters.

low-rank convolution filters into versatile convolution filters. Our LCT CNN is end-to-end and can be trained from the ground up. Our LCT approach can approach commonly trained convolution filters in terms of versatility and representation capacity.

[42] presents a spatial transformer formulation and the corresponding spatial transformer network (STN). The spatial transformer is applied to feature maps of the network and can learn invariance to translation, scale, rotation, and more generic warping. It can be inserted into existing convolutional architectures as an additional layer. In STN, for multi-channel inputs, the same spatial transformation is applied to each channel. We focus on the representation capacity of convolution filters, as opposed to the spatial transformer in STNs, and apply learnable spatial transformations to low-rank convolution filters. And the transformations differ depending on the convolution filter. As a result, we achieve a better trade-off between representation capacity and compression rate.

3. Low-rank compact transformed design for convolutional layers

The primary goal of this work is to reconstruct convolutional layers in a light-weighted manner. In this section, we introduce the low-rank compact representation of convolution filters and spatial transformation of different forms and hierarchies and analyze the hyper-parameter setting of LCT convolutional layers.

3.1. Preliminary notations for low-rank tensor decomposition

A tensor is a multi-dimensional array. A d-order tensor represents a d dimensional multi-way array. Scalars, vectors and matrices are 0-order, 1-order and 2-order tensors, respectively. In this paper, we use lowercase letters ($x, y, z, \ldots$), bold lowercase letters ($\mathbf{x}, \mathbf{y}, \mathbf{z}, \ldots$) and uppercase letters ($X, Y, Z, \ldots$) to denote scalar, vector and matrix, respectively. An N-order tensor (N $\geq$ 3) is denoted as $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and each element is denoted as $x_{i_1, i_2, \ldots, i_N}$. The mode-n matrix form of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the operation of reshaping the tensor into a matrix $X_{(n)} \in \mathbb{R}^{I_n \times (I_1 \cdots I_{n-1} I_{n+1} \cdots I_N)}$. For a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, its Tucker decomposition is defined as:

$$\begin{aligned} \mathcal{X} &= \sum_{s_1=1}^{S_1} \times \sum_{s_N}^{S_N} g_{s_1 s_2 \ldots s_N} (u_{s_1}^{(1)} \circ u_{s_2}^{(2)} \circ \cdots \circ u_{s_N}^{(N)}) \\ &= \mathcal{G} \times_1 U^{(1)} \times_2 U^{(2)} \cdots \times_N U^{(N)}. \end{aligned} \tag{1}$$

where $\mathcal{G} \in \mathbb{R}^{S_1 \times S_2 \times \cdots \times S_N}$ denotes the core tensor and $U^{(n)} = [u_1^{(n)}, u_2^{(n)}, \ldots, u_N^{(n)}] \in \mathbb{R}^{I_n \times S_n}$ denotes a factor matrix.

3.2. Low-rank compact representation

The convolution filters of a convolutional layer can be regarded as a 4-order tensor $\mathcal{X} \in \mathbb{R}^{W \times H \times C \times N}$, where W and H represent the filter width and height, C is the number of input channels, and N is the number of output channels. To realize the low-rank compact representation of convolution filters, we define a 4-order core tensor and four corresponding factor matrices to perform an inverse operation of Tucker decomposition and generate the 4-order low-rank convolution filter. The generation of the 4-order convolution filter $\mathcal{X}^L$ is formulated as:

$$\mathcal{X}^L = \mathcal{G} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \times_4 U^{(4)}, \tag{2}$$

where $\mathcal{G} \in \mathbb{R}^{w\times h\times c\times n}$ denotes the core tensor, and $U^{(1)} \in \mathbb{R}^{W\times w}$, $U^{(2)} \in \mathbb{R}^{H\times h}$, $U^{(3)} \in \mathbb{R}^{C\times c}$, $U^{(4)} \in \mathbb{R}^{N\times n}$ denote four factor matrices, respectively. w, h denote the compressed filter width and height of the convolution filter, respectively. c, n denote the compressed number of input and output channels, respectively.

The dimension of the core tensor is critical to the representation capacity of convolution filters. The 4-order low-rank convolution filter $\mathcal{X}^L$ can be calculated from any mode-n($n \in [1, 4]$) matrix form and then is reshaped to the tensor form $\mathcal{X}^L \in \mathbb{R}^{W\times H\times C\times N}$. For example, the mode-4 matrix form $\mathcal{X}^L_{(4)}$ is calculated as:

$$\mathcal{X}^L_{(4)} = U^{(4)} \cdot \mathcal{G}_{(4)} \cdot \left(U^{(3)} \otimes U^{(2)} \otimes U^{(1)}\right)^T, \tag{3}$$

where $\mathcal{X}^L_{(4)} \in \mathbb{R}^{N\times CWH}$, $\mathcal{G}_{(4)} \in \mathbb{R}^{n\times cwh}$ and $\otimes$ is the Kronecker product operator.

However, the computational complexity of Eq. (3) is very high. A matrix $A \in \mathbb{R}^{M\times N}$ multiplied by another matrix $B \in \mathbb{R}^{N\times P}$ has a computational complexity of $\mathcal{O}(MNP)$. A matrix $A \in \mathbb{R}^{M\times N}$ that takes the Kronecker product with another matrix $B \in \mathbb{R}^{P\times Q}$ has a computational complexity of $\mathcal{O}(MNPQ)$. The computational complexity of Eq. (3) is then $\mathcal{O}(WHwh\,(1 + Cc) + WHCn\,(N + whc))$.

We optimize Eq. (3) to achieve the lowest computational complexity. We transpose $\mathcal{X}^L$ to $\mathbb{R}^{C\times W\times H\times N}$. The calculation of Eq. (3) can then be rewritten as:

$$\mathcal{X}^L_{(43)} = \left(U^{(4)} \otimes U^{(3)}\right) \cdot \mathcal{G}_{(43)} \cdot \left(U^{(2)} \otimes U^{(1)}\right)^T, \tag{4}$$

where $\mathcal{X}^L_{(43)} \in \mathbb{R}^{NH\times CW}$, $\mathcal{G}_{(43)} \in \mathbb{R}^{nh\times cw}$, $U^{(4)} \in \mathbb{R}^{N\times n}$, $U^{(3)} \in \mathbb{R}^{H\times h}$, $U^{(2)} \in \mathbb{R}^{W\times w}$ and $U^{(1)} \in \mathbb{R}^{C\times c}$, respectively. In this way, the computational complexity of Eq. (4) is $\mathcal{O}(NHnh\,(1 + CW) + CWcw\,(1 + nh))$, which is much smaller than that of Eq. (3). After the calculation, $\mathcal{X}^L_{(43)}$ is reshaped into a tensor with dimensions $\mathbb{R}^{C\times W\times H\times N}$ and then transposed to $\mathbb{R}^{W\times H\times C\times N}$ to get $\mathcal{X}^L$. Thus the computational complexity of calculating $\mathcal{X}^L$ is greatly reduced.

3.3. Spatial transformation

In the low-rank convolution filter $\mathcal{X}^L \in \mathbb{R}^{W\times H\times C\times N}$ generated from the inverse operation of Tucker decomposition, there are a lot of similarity among the 2-dimensional filters $F^L_k \in \mathbb{R}^{W\times H}$ ($k \in [1, CN]$), i.e., the redundant information. It seriously affects the representation capacity of the convolution filter $\mathcal{X}^L$. To obtain a versatile convolution filter $\mathcal{X}$ from $\mathcal{X}^L$, we want to obtain appropriate spatial transformation to derive each $F_k \in \mathbb{R}^{W\times H}$ ($k \in [1, CN]$) of the versatile convolution filter $\mathcal{X}$ from the corresponding $F^L_k \in \mathbb{R}^{W\times H}$ ($k \in [1, CN]$) of the low-rank convolution filter $\mathcal{X}^L$. The spatial transformation is formulated as: $F^L_k = \mathcal{T}(F_k)$, where $\mathcal{T}$ is the spatial transformation. We apply the method proposed in [42] to obtain the spatial transformation $\mathcal{T}$ and F_k. The spatial transformation $\mathcal{T}$ can take different forms. We choose rotation transformation and affine transformation in this paper. To obtain each element value of $F_k\left(x_i, y_j\right) \in \mathbb{R}^{W\times H}$ ($x_i = 1, 2, \ldots, W; y_j = 1, 2, \ldots, H$), we calculate $\left(x^T_i, y^T_j\right)$ transformed from $\left(x_i, y_j\right)$ firstly. For rotation transformation $\mathcal{T}_R$, $\left(x^T_i, y^T_j\right)$ is calculated as:

$$\begin{pmatrix} x^T_i \\ y^T_j \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_i \\ y_j \end{pmatrix}, \tag{5}$$

where θ is the learnable rotation angle. $\left(x^T_i, y^T_j\right)$ for affine transformation $\mathcal{T}_A$ is calculated as:

$$\begin{pmatrix} x^T_i \\ y^T_j \end{pmatrix} = \begin{pmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{pmatrix} \begin{pmatrix} x_i \\ y_j \\ 1 \end{pmatrix}, \tag{6}$$

where $(\theta_{11}, \ldots, \theta_{23})$ are the learnable affine parameters.

We then use a bilinear sampling kernel [42] to get each element value of $F_k\left(x_i, y_j\right) \in \mathbb{R}^{W\times H}$, which is calculated as:

$$F_k\left(x_i, y_j\right) = \sum_{n=1}^{H}\sum_{m=1}^{W} F^L_k(m, n) \max\left(0, 1 - \left|x^T_i - m\right|\right) \max\left(0, 1 - \left|y^T_j - n\right|\right). \tag{7}$$

In this way, we can eliminate the redundant information among the 2-dimensional filters $F^L_k \in \mathbb{R}^{W\times H}$ and obtain each $F_k \in \mathbb{R}^{W\times H}$ ($k \in [1, CN]$) of the versatile convolution filter $\mathcal{X}$. Furthermore, we propose slice spatial transformation for 2D convolution filters and group spatial transformation for 3D convolution filters for different transformation hierarchies.

Slice Spatial Transformation For each filter $F^L_k \in \mathbb{R}^{W\times H}$ ($k \in [1, CN]$) of $\mathcal{X}^L \in \mathbb{R}^{W\times H\times C\times N}$, we can obtain a transformation $\mathcal{T}_k$ and corresponding F_k, as stated above. Each filter $F^L_k \in \mathbb{R}^{W\times H}$ ($k \in [1, CN]$) has different spatial transformation, which we refer to as slice spatial transformation.

Group Spatial Transformation Slice spatial transformation produces richer patterns for convolution filters while also introducing more parameters. We can let related slices share the same spatial transformation to reduce the number of spatial transformations. We implement it by merging $F^L_k \in \mathbb{R}^{W\times H}$ of the convolution filter $\mathcal{X}^L$ used for the same input or output channel into a group. For each group of convolution filters corresponding to input channels $\mathcal{X}^L_i \in \mathbb{R}^{W\times H\times C}$ ($i \in [1, N]$) of $\mathcal{X}^L \in \mathbb{R}^{W\times H\times C\times N}$, we can obtain a transformation $\mathcal{T}_i$ and corresponding $\mathcal{X}_i$. Similarly, for each group of convolution filters corresponding to output channels $\mathcal{X}^L_j \in \mathbb{R}^{W\times H\times N}$ ($j \in [1, C]$) of $\mathcal{X}^L \in \mathbb{R}^{W\times H\times C\times N}$, we can obtain a transformation $\mathcal{T}_j$ and corresponding $\mathcal{X}_j$. The transformation shared by the slices of convolution filters in the same group is known as group spatial transformation. Group spatial transformation significantly reduces the number of spatial transformations and network parameters by sharing transformations. The experiment in Section 5.1 demonstrates different spatial transformations performances.

3.4. Hyper-parameter analysis

In a convolutional layer, the number of parameters denoted as P_0 of a convolution filter $\mathcal{X} \in \mathbb{R}^{W\times H\times C\times N}$ is calculated by

$$P_0 = W \times H \times C \times N. \tag{8}$$

The number of parameters denoted as P_C for generating $\mathcal{X} \in \mathbb{R}^{W\times H\times C\times N}$ by our compact design method includes two parts, the number of parameters of Tucker product factors and the number of learnable parameters of spatial transformations. The number of parameters of Tucker product factors P_L is the sum of the parameters of the core tensor $\mathcal{G} \in \mathbb{R}^{w\times h\times c\times n}$ and four factor matrices $U^{(1)} \in \mathbb{R}^{W\times w}$, $U^{(2)} \in \mathbb{R}^{H\times h}$, $U^{(3)} \in \mathbb{R}^{C\times c}$, $U^{(4)} \in \mathbb{R}^{N\times n}$:

$$P_L = w \times h \times c \times n + W \times w + H \times h + C \times c + N \times n. \tag{9}$$

The number of parameters of the spatial transformations P_T depends on their form and hierarchy. Let N_T and p_T denote the number of transformations and the number of parameters of each transformation, respectively. Then P_T is calculated by

$$P_T = N_T \times p_T. \tag{10}$$

For slice spatial transformation, $N_T = C \times N$, where C and N are the number of input and output channels, respectively. Similarly, $N_T = N$ or $N_T = C$ for group spatial transformation, depending on the group dimension of input or output channel. $p_T = 1$ for rotation transformation and $p_T = 6$ for affine transformation. The number of parameters P_C for a low-rank compact design convolution layer is then determined by

$$P_C = P_L + P_T. \tag{11}$$

The compression rate of the convolutional layer denoted as r is then calculated by:

$$r = \frac{P_C}{P_O} = \frac{P_L + P_T}{P_O}. \tag{12}$$

Here, we discuss the number of parameters and compression rate in a common case where a convolution layer has the same number of input and output channels and employs a 2-dimensional square filter, i.e., $C = N$ and $W = H$. Furthermore, we set $w = h$ and the same channel compression rate $\lambda = \frac{c}{C}$ for input and output channels. In this case, P_L, P_T and r becomes:

$$P_L = w^2\lambda^2C^2 + 2Ww + 2\lambda C^2 \approx (w^2\lambda^2 + 2\lambda)C^2, \tag{13}$$

$$P_T = \begin{cases} 6C^2 & slice - affine \\ 6C & group - affine \\ C^2 & slice - rotation \\ C & group - rotation \end{cases}. \tag{14}$$

We can calculate the compression rate of different spatial transformation forms by combining Eqs. (12)–(14) as:

$$r = \frac{P_L + P_T}{P_O} = \begin{cases} \frac{w^2\lambda^2+2\lambda+6}{W^2} & slice - affine \\ \frac{(w^2\lambda^2+2\lambda)C+6}{W^2C} & group - affine \\ \frac{w^2\lambda^2+2\lambda+1}{W^2} & slice - rotation \\ \frac{(w^2\lambda^2+2\lambda)C+1}{W^2C} & group - rotation \end{cases}. \tag{15}$$

We can see that the slice-affine form for spatial transformation introduces many more parameters than the group-affine form, and the group-affine form for spatial transformation can significantly reduce the number of parameters when compared to the slice-affine form. In practice, we recommend a group-affine form for spatial transformation to achieve a better trade-off between accuracy and parameter count, as detailed in Section 5.1.

3.5. Gradient back-propagation

We demonstrate the gradient back-propagation for each part of the low-rank compact design convolution layer. Each element of $\mathcal{X}^L$ is calculated as:

$$\mathcal{X}^L_{ijkl} = \sum_{p=1}^{w}\sum_{q=1}^{h}\sum_{r=1}^{c}\sum_{s=1}^{n} g_{pqrs}u^{(1)}_{ip}u^{(2)}_{jq}u^{(3)}_{kr}u^{(4)}_{ls}, \tag{16}$$

and each corresponding element of the versatile convolution filter $\mathcal{X}$ is calculated as:

$$\mathcal{X}_{ijkl} = \sum_{n=1}^{H}\sum_{m=1}^{W}\mathcal{X}^L_{mnkl}\max\left(0, 1 - \left|x^T_i - m\right|\right)\max\left(0, 1 - \left|y^T_j - n\right|\right), \tag{17}$$

Let L denote the loss of the convolutional layer. The gradient of the core tensor $\mathcal{G}$ is then calculated as:

$$\frac{\partial L}{\partial \mathcal{G}_{pqrs}} = \frac{\partial L}{\partial \mathcal{X}}\cdot\frac{\partial \mathcal{X}}{\partial \mathcal{X}^L}\cdot\frac{\partial \mathcal{X}^L}{\partial \mathcal{G}_{pqrs}} = \frac{\partial L}{\partial \mathcal{X}}\cdot\sum_{n=1}^{H}\sum_{m=1}^{W}\max\left(0, 1 - \left|x^T_i - m\right|\right)\max\left(0, 1 - \left|y^T_j - n\right|\right) \cdot\sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{k=1}^{C}\sum_{l=1}^{N} u^{(1)}_{ip}u^{(2)}_{jq}u^{(3)}_{kr}u^{(4)}_{ls}, \tag{18}$$

and the gradient of the factor matrix $U^{(1)}$ (similarly for $U^{(2)}$, $U^{(3)}$ and $U^{(4)}$) is calculated as:

$$\frac{\partial L}{\partial U^{(1)}_{ip}} = \frac{\partial L}{\partial \mathcal{X}}\cdot\frac{\partial \mathcal{X}}{\partial \mathcal{X}^L}\cdot\frac{\partial \mathcal{X}^L}{\partial U^{(1)}_{ip}} = \frac{\partial L}{\partial \mathcal{X}}\cdot\sum_{n=1}^{H}\sum_{m=1}^{W}\max\left(0, 1 - \left|x^T_i - m\right|\right)\max\left(0, 1 - \left|y^T_j - n\right|\right) \cdot\sum_{j=1}^{H}\sum_{k=1}^{C}\sum_{l=1}^{N}\left(\sum_{q=1}^{h}\sum_{r=1}^{c}\sum_{s=1}^{n} g_{pqrs}u^{(2)}_{jq}u^{(3)}_{kr}u^{(4)}_{ls}\right). \tag{19}$$

The gradient for the transformation $\mathcal{T}$ is calculated as:

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \mathcal{X}}\cdot\frac{\partial \mathcal{X}}{\partial \theta} = \frac{\partial L}{\partial \mathcal{X}}\cdot\left(\sum_{n=1}^{H}\sum_{m=1}^{W}\mathcal{X}^L_{nm}\max\left(0, 1 - \left|y^T_j - n\right|\right)\frac{\partial \mathcal{X}}{\partial x^T_i}\frac{\partial x^T_i}{\partial \theta} + \sum_{n=1}^{H}\sum_{m=1}^{W}\mathcal{X}^L_{nm}\max\left(0, 1 - \left|x^T_i - n\right|\right)\frac{\partial \mathcal{X}}{\partial y^T_j}\frac{\partial y^T_j}{\partial \theta}\right) \tag{20}$$

where

$$\frac{\partial \mathcal{X}}{\partial x^T_i} = \begin{cases} 0 & if\ \left|x^T_i - m\right| \geq 1 \\ 1 & if\ m \geq x^T_i \\ -1 & if\ m < x^T_i \end{cases}, \tag{21}$$

$$\frac{\partial \mathcal{X}}{\partial y^T_j} = \begin{cases} 0 & if\ \left|y^T_j - n\right| \geq 1 \\ 1 & if\ n \geq y^T_j \\ -1 & if\ n < y^T_j \end{cases}, \tag{22}$$

and $\frac{\partial x^T_i}{\partial \theta}$, $\frac{\partial y^T_j}{\partial \theta}$ can be derived from the spatial transformation $\mathcal{T}$.

4. Experiment

In this section, we replace the convolutional layer in CNN with our LCT convolutional layer, and our approach is denoted as "LCT" throughout the experiment. Our LCT convolutional layer is implemented by Pytorch. We use the group-affine form for spatial transformation to achieve a better trade-off between accuracy and parameter count (see Section 5.1 for the reason for such choice derived from the ablation study). Fig. 3 shows the 4-d Tucker product of convolution filters to better illustrate the hyper-parameter setting. On the right side, the size of the 4-d low-rank convolution filter is $W \times H \times C \times N$, where $W \times H$ is the 2-d convolution filter size, C is the number of input channels and N is the number of output channels. The corresponding 4-d unfolded convolution filter is shown on the left side, with compressed sizes of corresponding dimensions w, h, c, and n, respectively. In the experiment of this paper, we only compress square convolution filters where $W = H$ to the same compressed size w = h. We use the same channel compression rate $\lambda = \frac{c}{C} = \frac{n}{N}$ for input and output channels. In the tables of experiments, our approach is denoted by "LCT(λ-W-w)", where λ is the channel compression rate and the filter size of the convolution filter is compressed from $W \times W$ to w × w. The influence of hyper-parameter setting is discussed in detail in Section 5.

We validate our approach on different datasets, including MNIST [43], CIFAR10, CIFAR100 [44] and ImageNet [45]. To validate the effectiveness of our approach, we compare it to several recent low-rank approaches, including Tucker [19], SSS [46], ADMM-TT [47], RSTR [48], LRER [49], Hinge [50], TR [51], TRP [52], SVD [53] and LRE [34], and vision transformer [54]. All of the following experiments are run on a single Nvidia GeForce RTX 3090 GPU.

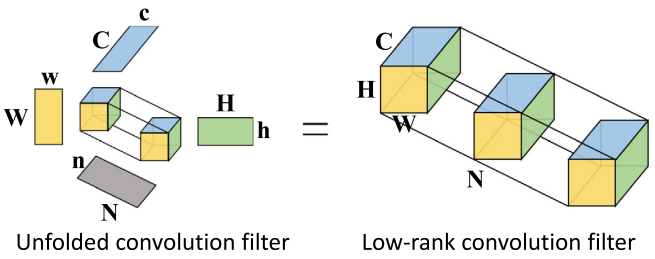

Fig. 3. 4-d Tucker product of convolution filters.

Table 1
Comparison of compression results on MNIST dataset. The parameter remaining percentage of the original baseline LeNet5 is denoted by "Params". Parentheses indicate the channel compression rate and convolution filter size compression settings. "0.5-5-5" denotes that the channel compression rate $\lambda = 0.5$ and the filter size of the convolution filter is compressed from 5×5 to 5×5. The following tables follow the same notation.

Model	Method	Accuracy (%)	Params (%)
LeNet5	Original	99.45	100
	ADMM-TT [47]	**99.51**	12.1
	LREL [49]	98.67	4.5
	LCT(ours,0.5-5-5)	99.23	4.5
	LCT(ours,0.75-3-3)	99.20	**4.4**

4.1. MNIST

MNIST is a well-known deep learning dataset that contains 60k training and 10k testing hand-written digits in 10 classes (0–9). This dataset is too simple, so we choose a simple CNN LeNet5 [43] as the backbone network. We replace the second convolutional layer in LeNet5 with our LCT layer. We train compressed LeNet5 with an Adam optimizer for 50 epochs. It begins with a learning rate of 0.004 and is divided by 2 at 20 and 40 epochs. The batch size is set to 128. Table 1 compares our approach to [47,49], and the original LeNet5 result is also from [49]. Our approach achieves the highest compression rate among these compression approaches. Under the same compression rate, the accuracy of our LCT approach is 0.56% higher than the LREL approach. Though the accuracy of our LCT approach is 0.28% lower than the ADMM-TT approach, our parameter compression rate is much higher.

4.2. CIFAR10 and CIFAR100

CIFAR10 and CIFAR100 datasets consist of colored natural images with 32×32 pixels in 10 and 100 classes, respectively. Each dataset contains 50k training images and 10k testing images. All of the CIFAR10 and CIFAR100 experiments that follow use the data augmentation provided in [55] for training: 4 pixels of value 0 are padded on each side for the 32×32 image, and a 32×32 crop is randomly sampled from the padded image or its horizontal flip. We only evaluate the original 32×32 images for testing.

We choose popular CNNs as backbone networks, including ResNet20, ResNet32, VGG16 and DenseNet40. We replace all of the convolutional layers in these CNNs with our LCT convolutional layers. The baseline networks and the compressed networks are trained with an SGD optimizer for 300 epochs. The momentum is 0.9, and the weight decay factor is 10^{-4}. It begins with a learning rate of 0.05 and is divided by 10 at 100 and 200 epochs. The batch size is set to 128. Furthermore, we select the CMT [54] approach as the backbone network, which combines CNN and vision transformer. CMT begins the network with a stem architecture that includes three convolutional layers to better extract local information. We use our LCT convolutional layers to replace the three convolutional layers. The baseline networks and the compressed networks are trained with an AdamW optimizer with a weight decay factor 10^{-5} for 150 epochs. The initial learning rate is 6×10^{-5} and the learning rate scheduler follows a cosine annealing schedule. The batch size is set to 64.

Table 2
Comparison of compression results on CIFAR10 dataset. We report top-1 accuracy of each approach together with its baseline counterpart (denoted as "BL"). "Pro(%)" denotes the promotion compared with baseline counterpart, calculated by top-1 accuracy minus baseline accuracy.

Model	Method	Top-1/BL (%)	Pro (%)	Params (%)
ResNet20	SSS [46]	90.85/92.53	−1.68	83.4
	ADMM-TT [47]	91.03/91.25	−0.22	**14.7**
	LREL [49]	90.81/91.82	−1.01	58.0
	Hinge [50]	**91.84**/92.54	−0.70	44.6
	RSTR [48]	88.30/90.40	−2.10	16.7
	LCT(ours,0.5-5-3)	90.77/91.25	−0.48	38.1
	LCT(ours,0.75-5-3)	91.19/91.25	**−0.06**	74.4
ResNet32	ADMM-TT [47]	91.96/92.49	−0.53	17.2
	LREL [49]	91.95/92.92	−0.97	57.2
	RSTR [48]	88.10/92.50	−4.40	**6.67**
	Tucker [19]	87.70/92.50	−4.80	20.0
	LCT(ours,0.75-3-3)	92.16/92.02	+0.14	74.4
	LCT(ours,0.75-5-3)	**92.28**/92.02	**+0.26**	75.6
VGG16	SSS [46]	91.22/93.91	−2.69	32.8
	LREL [49]	92.95/93.68	−0.73	27.9
	Hinge [50]	**93.59**/94.02	−0.43	19.95
	LCT(ours,0.5-5-3)	92.65/92.72	−0.07	38.7
	LCT(ours,0.75-5-3)	92.83/92.72	**+0.11**	80.4
DenseNet40	Hinge [50]	**94.67**/94.74	−0.07	72.5
	LCT(ours,0.25-3-3)	94.14/94.26	−0.12	**55.1**
	LCT(ours,0.5-3-3)	94.26/94.26	**+0.00**	71.9
CMT	CMT-S [54]	91.43/91.43	+0.00	100
	LCT(ours,0.5-3-3)	91.46/91.43	+0.03	**42.7**
	LCT(ours,0.75-3-3)	**91.54**/91.43	**+0.11**	80.3

As shown in Table 2, we compare our approach with various latest effective approaches on the CIFAR10 dataset. We report top-1 accuracy, baseline accuracy, and promotion compared with the baseline counterpart of each approach. Our approach can simultaneously improve accuracy and compress parameters, which is highlighted in bold at "Pro(%)". The ordinary Tucker decomposition approach [19] has 4.8% accuracy drop with 20% remaining parameters, which validates the effectiveness of our spatial transformation. Under similar remaining percentage of parameters in the ResNet20 results comparison, our LCT has 0.48% accuracy drop with 38.1% remaining parameters and Hinge [50] has 0.70% accuracy drop with 44.6% remaining parameters, which validates a better trade-off of our LCT between accuracy and compression rate. For CMT comparison, the reported parameter remaining percentage is of the stem architecture. As understood from the results, although the compression rate of our LCT approach is not the best, our approach can improve accuracy and compress parameters simultaneously.

As shown in Table 3, our LCT approach outperforms all other approaches in terms of top-1 accuracy on ResNet20, ResNet32, and CMT, which is highlighted in bold. Under similar remaining percentage of parameters in the ResNet20 results comparison, our LCT has 0.29% accuracy drop with 39.5% remaining parameters, while Hinge [50] has up to 2.49% accuracy drop with 33.6% remaining parameters. RSTR [48] has the least parameters of 12.5%, but the accuracy drops significantly up to 4.1%. ADMM-TT [47] achieves a good trade-off between accuracy and compression rate. It has larger compression rate than our LCT. Our LCT achieves 1.69% and 1.34% better accuracy than ADMM-TT on ResNet20 and ResNet32, respectively. It is worth mentioning that we can

Table 3
Comparison of compression results on CIFAR100 dataset. We report the top-1 accuracy of each approach together with its baseline counterpart. "Pro(%)" denotes the promotion compared with the baseline counterpart, calculated by top-1 accuracy minus baseline accuracy.

Model	Method	Top-1/BL (%)	Pro (%)	Params (%)
ResNet20	SSS [46]	65.58/69.09	−3.51	54.4
	ADMM-TT [47]	64.92/65.40	−0.48	17.9
	Hinge [50]	66.34/68.83	−2.49	33.6
	RSTR [48]	61.30/65.40	−4.10	**12.5**
	LCT(ours,0.5-5-3)	66.17/66.46	−0.29	39.5
	LCT(ours,0.75-5-3)	**66.61**/66.46	+**0.15**	75.0
ResNet32	ADMM-TT [47]	67.17/68.10	−0.93	19.2
	RSTR [48]	64.50/68.10	−3.60	**8.33**
	LCT(ours,0.5-5-3)	67.76/68.23	−0.47	38.5
	LCT(ours,0.75-3-3)	**68.51**/68.23	+**0.28**	74.7
CMT	CMT-S [54]	70.11/70.11	+0.00	100
	LCT(ours,0.5-3-3)	71.88/70.11	+1.77	**42.7**
	LCT(ours,0.75-3-3)	**72.06**/70.11	+**1.95**	80.3

Table 4
Comparison of compression results on ImageNet dataset. We report the top-1 and top-5 accuracy of each approach.

Model	Method	Top-1 (%)	Top-5 (%)	Params (%)
ResNet18	TR [51]	–	86.29	23.4
	ADMM-TT [47]	–	87.47	**21.7**
	TRP [52]	65.51	86.74	38.5
	SVD [53]	63.10	84.44	70.9
	LRE [34]	62.80	83.72	50.0
	LCT(ours,0.5-3-3)	**67.87**	**88.10**	38.9
	LCT(ours,0.75-3-3)	**69.33**	**88.84**	73.7
ResNet50	Hinge [50]	74.7	–	48.6
	SSS [46]	72.98	91.08	83.1
	TRP [52]	72.69	91.41	**43.5**
	SVD [53]	71.80	90.20	66.7
	LCT(ours,0.25-3-3)	**75.06**	**91.59**	48.4
	LCT(ours,0.5-3-3)	**76.12**	**92.95**	60.5
	LCT(ours,0.75-3-3)	**76.71**	**93.19**	87.1

achieve 1.95% accuracy improvement on CMT-S by only replacing convolutional layers in the stem architecture with our LCT convolutional layers.

4.3. ImageNet

ImageNet contains 1.28 million training images and 50 thousand validation images from 1000 different classes. The data augmentation for training includes a random resized cropping and a random horizontal flip. The image is center cropped to match the input size for validation.

We choose ResNet18 and ResNet50 as backbone networks. We replace all the convolutional layers in ResNet18 with our LCT convolutional layers. The bottleneck structure in ResNet50 consists 1×1 and 3×3 convolutional layers. As spatial transformation does not apply to 1×1 convolutional filters, we only replace 3×3 convolutional layers with our LCT convolutional layers. In Pytorch, we define an LCT convolutional layer with a (3, 3) filter size pair to take advantage of pre-trained weights from model-zoo. By applying Tucker decomposition to the pre-trained weights of the corresponding layer, we initialize the core tensor and factor matrices of the LCT convolutional layer. The filter width and height rank are set to 3. The rank of input and output channels can be easily calculated given a channel compression rate. An identity affine transformation is used to initialize the spatial transformation.

We fine-tune the compressed networks with an SGD optimizer for 30 epochs. The momentum is 0.9, and the weight decay factor is 10^{-4}. The initial learning rate is 10^{-5} and divided by 10 at 20 epoch. The used batch size is 128 for the compressed ResNet18, and the used batch size is 64 for the compressed ResNet50.

As shown in Table 4, our LCT approach outperforms all other approaches in terms of accuracy. Aside from improved accuracy, our LCT has a higher compression rate than Hinge [50], SSS [46], SVD [53] and LRE [34]. Under a similar remaining percentage of parameters in ResNet18 results comparison, Our LCT (38.9% parameter percentage) achieves 2.36% higher top-1 accuracy and 1.36% higher top-5 accuracy compared with TRP (38.4% parameter percentage) [52]. TR [51] and ADMM-TT [47] achieve higher compression rates than our LCT because of the properties of Tensor Train and Tensor Ring decomposition. Our LCT achieves 2.55% and 1.37% higher top-5 accuracy than TR and ADMM-TT.

5. Ablation study

There is a trade-off between accuracy and compression rate. In Sections 5.1, 5.2, and 5.3, we give the ablation study for the hyper-parameter setting of our LCT convolution layer, including different spatial transformations, convolution filter size compression, and channel compression rates.

All the ablation study experiments are conducted on the CIFAR-10 dataset. We use a common CNN architecture as the baseline to demonstrate the performance of our LCT convolution layer. The Conv-BN-ReLU module is simply referred to as a conv-module because the convolutional layer is followed by a Batch Normalization layer (BN) [56] and rectified linear units (ReLU) [57]. The CNN utilizes 4 stacks of conv-module, an average pooling layer after the first 2 stacks, and ends with a global average pooling [58], flatten, and a 10-way fully-connected layer.

For convolution filters, we use a Xavier initializer [59] and an $l2$ regularizer with a weight decay of 0.0001. These models are trained on a single Nvidia Titan XP GPU using a minibatch size of 128. We train the networks for 100 epochs with an Adam optimizer, starting with a learning rate of 0.001 and dividing by 10 at 35 and 70 epochs.

5.1. Different spatial transformations

Following the experimental setup described above, we show how different spatial transformations affect network performance. As shown in Table 5, all the convolutional layers of the baseline 7×7-CNN utilize 7×7 convolution filters and have 64 output channels (the same setting for baseline CNNs in Sections 5.2 and 5.3). The following 5 networks are implemented by replacing all the convolutional layers with our LCT convolution layers. The filter size of the convolution filter is compressed from 7×7 to 5×5. The channel compression rate is set as 0.5 for input and output channels. In the first LCT convolutional layer, the number of input channels is 3, which is too small to compress. As a result, in all experiments, we do not compress first-layer input channels. L-CNN is a network with a low-rank compact representation for a 4-order convolution filter that does not undergo any spatial transformation. GR-CNN, SR-CNN, GA-CNN, and SA-CNN are networks with group-rotation, slice-rotation, group-affine, and slice-affine spatial transformations, respectively.

L-CNN has the highest top-1 validation error when compared to GR-CNN, SR-CNN, GA-CNN, and SA-CNN. It shows that spatial transformations are important for improving the representation capacity of low-rank convolution filters and achieving better performance. SA-CNN and GA-CNN outperform GR-CNN and SR-CNN in terms of performance. It shows that affine form spatial transformations improve the representation capacity of low-rank convolution filters more than rotation form spatial transformations. The test accuracy of SA-CNN is 0.5% higher than that of GA-CNN, but the number of parameters of SA-CNN is 1.7 times

Table 5
Test accuracy of our low-rank compact CNNs without spatial transformation and with different spatial transformations on CIFAR-10. 7 × 7-CNN is the baseline network.

Model	Accuracy (%)	Num of params (k)	Params (%)
7 × 7-CNN	85.35	615.6	100
L-CNN	81.69	97.9	15.9
GR-CNN	83.60	98.1	15.9
SR-CNN	85.15	110.4	17.9
GA-CNN	86.71	99.1	16.0
SA-CNN	**87.21**	172.8	28.1

Table 6
Test accuracy of our LCT CNNs with different convolution filter size compression on CIFAR-10. 3 × 3-CNN, 5 × 5-CNN and 7 × 7-CNN are baseline networks.

Model	Accuracy (%)	Num of params (k)	Params (%)
7 × 7-CNN	85.35	615.6	100
5 × 5-CNN	84.62	316.1	51.3
3 × 3-CNN	82.67	116.4	18.9
LCT(0.5-3-3)	83.65	48.2	7.8
LCT(0.5-5-3)	85.06	48.2	7.8
LCT(0.5-7-3)	85.01	48.3	7.8
LCT(0.5-7-5)	**86.71**	99.1	16.0
LCT(0.5-9-5)	85.98	99.2	16.1

that of GA-CNN. GA-CNN has a parameter remaining percentage of 16.0% while 1.36% accuracy improvement compared with baseline 7 × 7-CNN. This demonstrates that group-affine spatial transformations achieve a better trade-off between parameter number and performance.

5.2. Different convolution filter size compression

In this section, we discuss how different convolution filter sizes affect network performance. For both input and output channels, the channel compression rate is set to 0.5. We apply group-affine spatial transformations to the low-rank convolution filter. In Table 6, we compare our LCT CNNs with three baseline CNNs, utilizing convolution filters of size 3 × 3, 5 × 5 and 7 × 7, denoted as 3 × 3-CNN, 5 × 5-CNN and 7 × 7-CNN, respectively. For our LCT CNNs, the filter size of convolution filter is compressed from (3 × 3 to 3 × 3), (5 × 5 to 3 × 3), (7 × 7 to 3 × 3), (7 × 7 to 5 × 5) and (9 × 9 to 5 × 5).

As shown in Table 6, compared with baseline CNNs, our LCT CNNs achieve a high compression rate and better performance. Comparing LCT(0.5-3-3), LCT(0.5-5-3), and LCT(0.5-7-3), we find that when the filter size of unfolded convolution filters is fixed, the filter size of versatile convolution filters has little effect on the number of parameters. LCT(0.5-5-3) achieves the best accuracy among the three models. LCT(0.5-7-5) and LCT(0.5-9-5) have similar number of parameters and LCT(0.5-7-5) achieves better accuracy. As a result, choosing an appropriate filter size for unfolded convolution filters and versatile convolution filters is critical. LCT(0.5-7-5) has the best performance of all cases.

5.3. Different channel compression rates

In this section, we discuss how different channel compression rates influence network performance. The baseline networks are 3 × 3-CNN, 5 × 5-CNN, and 7 × 7-CNN in Table 6. The filter size of the convolution filter is compressed from 7 × 7 to 5 × 5. We apply group-affine spatial transformations to the low-rank convolution filter. Let λ denote the compression rate of input and output channels, selected from $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8})$. Fig. 4 shows the relation between test error and the number of parameters of three baseline CNNs and our LCT(λ-7-5) with different channel compression rates λ. The curve of our LCT(λ-7-5) is at the lower left corner

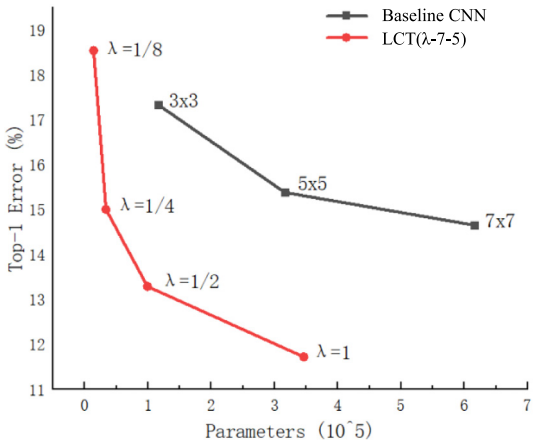

Fig. 4. The trade-off between the number of parameters and test error for our LCT(λ-7-5) with different channel compression rates λ, compared with baseline CNNs.

of the curve of baseline CNNs. It demonstrates that our LCT(λ-7-5) achieves a better trade-off between the number of parameters and test error. Within the same number of parameters, our LCT(λ-7-5) achieves a rather smaller error than the baseline CNN. Within the same error, our LCT(λ-7-5) achieves a high compression rate compared with the baseline CNN. The performance degrades seriously while using a very small $\lambda = \frac{1}{8}$, as the number of compressed channels is too small to retain the necessary information of convolutional filters. LCT(λ-7-5) with $\lambda = \frac{1}{2}$ achieves the best trade-off between the number of parameters and test error.

6. Discussion on experiment

From Table 5, our LCT CNNs can achieve 15.9% to 28.1% remaining percentage of parameters while 1.36% to 1.86% accuracy improvement compared with baseline 7 × 7-CNN. While compressing CNNs of filter size 3 × 3 such as ResNet32 and VGG16, our approach can achieve only about 38% remaining percentage of parameters and close accuracy compared with the baseline network. We summarize the reasons as follows. 7 × 7 convolution filters have much more versatility than common 3 × 3 convolution filters. Therefore, applying spatial transformations to low-rank 7 × 7 convolution filters can enhance their representation capacity more significantly than applying spatial transformations to low-rank 3 × 3 convolution filters. In addition, the filter size of 3 × 3 convolution filters is too small to compress. We will investigate a new way to compress CNNs with 3 × 3 convolution filters in our future work.

7. Conclusion

In this paper, we propose a novel compact design for convolutional layers with spatial transformation. The convolution filters of convolutional layers in our design are generated using a predefined Tucker factor form, which is then followed by learnable individual spatial transformations on each filter. Compact Tucker factor convolution filters have far fewer parameters than standard convolution filters. Furthermore, spatial transformations significantly improve the representation capacity of convolution filters. We discuss various hyper-parameter settings for our LCT convolution layer, such as different spatial transformations, filter size pairs of versatile and unfolded convolution filters, and channel compression rates. The results of the experiments validate our proposed LCT design for convolutional layers.

CRediT authorship contribution statement

Baichen Liu: Software, Validation, Formal analysis, Investigation, Writing - original draft. **Zhi Han:** Conceptualization, Methodology, Writing – review & editing. **Xi'ai Chen:** Resources, Writing – review & editing, Funding acquisition. **Wenming Shao:** Project administration, Writing – review & editing. **Huidi Jia:** Data curation, Visualization. **Yanmei Wang:** Software, Visualization. **Yandong Tang:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFB1313400, the National Natural Science Foundation of China under Grant 61903358, 61773367, 61821005, and the Youth Innovation Promotion Association of the Chinese Academy of Sciences under Grant 2022196, Y202051.

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: Communications of the ACM, Vol. 60, 2017, pp. 84–90.
[2] S. Chen, H. Shen, R. Wang, X. Wang, Towards improving fast adversarial training in multi-exit network, Neural Netw. 150 (2022) 1–11.
[3] S. Song, C. Ma, W. Sun, J. Xu, J. Dang, Q. Yu, Efficient learning with augmented spikes: A case study with image classification, Neural Netw. 142 (2021) 205–212.
[4] N. Guo, K. Gu, J. Qiao, J. Bi, Improved deep CNNs based on Nonlinear Hybrid Attention Module for image classification, Neural Netw. 140 (2021) 158–166.
[5] N. Rodríguez-Barroso, E. Martínez-Cámara, M.V. Luzón, F. Herrera, Backdoor attacks-resilient aggregation based on Robust Filtering of Outliers in federated learning for image classification, Knowl.-Based Syst. 245 (2022) 108588.
[6] Y. Wu, W. Ma, M. Gong, Z. Bai, W. Zhao, Q. Guo, X. Chen, Q. Miao, A coarse-to-fine network for ship detection in optical remote sensing images, Remote Sens. 12 (2020) 246.
[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (2015) 211–252.
[8] Y. Wu, J. Li, Y. Yuan, A.K. Qin, Q.G. Miao, M.G. Gong, Commonality autoencoder: Learning common features for change detection from heterogeneous images, IEEE Trans. Neural Netw. Learn. Syst. 99 (2021) 1–14.
[9] Y. Wu, Z. Xiao, S. Liu, Q. Miao, W. Ma, M. Gong, F. Xie, Y. Zhang, A two-step method for remote sensing images registration based on local and global constraints, IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 14 (2021) 5194–5206.
[10] H. Dong, W. Ma, Y. Wu, J. Zhang, L. Jiao, Self-supervised representation learning for remote sensing image change detection based on temporal prediction, Remote Sens. 12 (2020) 1868.
[11] D.H. Hubel, T.N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, J. Physiol. 160 (1962) 106–154.
[12] D.H. Hubel, T.N. Wiesel, Receptive fields and functional architecture of monkey striate cortex, J. Physiol. 195 (1968) 215–243.
[13] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representations, ICLR, Vol. May-2015, 2015.
[14] K. He, J. Sun, Convolutional neural networks at constrained time cost, in: Computer Vision and Pattern Recognition, CVPR, Vol. June-2015, 2015, pp. 5353–5360.
[15] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Computer Vision and Pattern Recognition, CVPR, Vol. December-2016, 2016, pp. 770–778.
[16] R.K. Srivastava, K. Greff, J. Schmidhuber, Highway networks, 2015, arXiv preprint arXiv:1505.00387. URL http://arxiv.org/abs/1505.00387.
[17] W. Li, W. Xie, Z. Wang, Complex-valued densely connected convolutional networks, in: Communications in Computer and Information Science, Vol. 1257, 2020, pp. 299–309.
[18] X. Jiang, N. Wang, J. Xin, X. Xia, X. Yang, X. Gao, Learning lightweight super-resolution networks with weight pruning, Neural Netw. 144 (2021) 21–32.
[19] Y.D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, D. Shin, Compression of deep convolutional neural networks for fast and low power mobile applications, in: International Conference on Learning Representations, ICLR, Vol. May-2016, 2016.
[20] T.S. Cohen, M. Welling, Group equivariant convolutional networks, in: International Conference on Machine Learning, ICML, Vol. 6, 2016, pp. 4375–4386.
[21] D. Wang, G. Zhao, H. Chen, Z. Liu, L. Deng, G. Li, Nonlinear tensor train format for deep neural network compression, Neural Netw. 144 (2021) 320–333.
[22] B. Wu, D. Wang, G. Zhao, L. Deng, G. Li, Hybrid tensor decomposition in neural network compression, Neural Netw. 132 (2020) 309–320.
[23] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in: Advances in Neural Information Processing Systems, Vol. 2, 2014, pp. 1269–1277.
[24] V. Lebedev, V. Lempitsky, Speeding-up convolutional neural networks: A survey, Bull. Polish Acad. Sci.: Tech. Sci. 66 (2018) 799–810.
[25] J. Ye, L. Wang, G. Li, D. Chen, S. Zhe, X. Chu, Z. Xu, Learning compact recurrent neural networks with block-term tensor decomposition, in: Computer Vision and Pattern Recognition, CVPR, Vol. June-2018, 2018, pp. 9378–9387.
[26] I.V. Oseledets, Tensor-train decomposition, SIAM J. Sci. Comput. 33 (2011) 2295–2317.
[27] Y. Liu, M.K. Ng, Deep neural network compression by Tucker decomposition with nonlinear response, Knowl.-Based Syst. 241 (2022) 108171.
[28] G. Lee, K. Lee, DNN compression by ADMM-based joint pruning, Knowl.-Based Syst. 239 (2022) 107988.
[29] S. Zhai, Y. Cheng, W. Lu, Z. Zhang, Doubly convolutional neural networks, Adv. Neural Inf. Process. Syst. 29 (2016) 1090–1098.
[30] W. Shang, K. Sohn, D. Almeida, H. Lee, Understanding and improving convolutional neural networks via concatenated rectified linear units, in: International Conference on Machine Learning, ICML, Vol. 5, 2016, pp. 3276–3284.
[31] H. Li, W. Ouyang, X. Wang, Multi-bias non-linear activation in deep neural networks, in: International Conference on Machine Learning, ICML, Vol. 1, 2016, pp. 365–373.
[32] J.Y. Li, Y.K. Zhao, Z.E. Xue, Z. Cai, Q. Li, A survey of model compression for deep neural networks, Gongcheng Kexue Xuebao 41 (2019) 1229–1239.
[33] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, N. De Freitas, Predicting parameters in deep learning, in: Advances in Neural Information Processing Systems, Vol. 26, 2013, pp. 2148–2156.
[34] M. Jaderberg, A. Vedaldi, A. Zisserman, Speeding up convolutional neural networks with low rank expansions, in: British Machine Vision Conference, BMVC, Vol. September-2014, 2014.
[35] G. Fang, Y. Bao, J. Song, X. Wang, D. Xie, C. Shen, M. Song, Mosaicking to distill: Knowledge distillation from out-of-domain data, Adv. Neural Inf. Process. Syst. 34 (2021) 11920–11932.
[36] G. Fang, K. Mo, X. Wang, J. Song, S. Bei, H. Zhang, M. Song, Up to 100x faster data-free knowledge distillation, in: AAAI Conference on Artificial Intelligence, Vol. 36, (6) 2022, pp. 6597–6604.
[37] Y. He, P. Liu, Z. Wang, Z. Hu, Y. Yang, Filter pruning via geometric median for deep convolutional neural networks acceleration, in: Computer Vision and Pattern Recognition, CVPR, Vol. June-2019, 2019, pp. 4340–4349.
[38] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, Y. Yang, Asymptotic soft filter pruning for deep convolutional neural networks, IEEE Trans. Cybern. 50 (8) (2019) 3594–3604.
[39] Y. He, P. Liu, L. Zhu, Y. Yang, Filter pruning by switching to neighboring CNNs with good attributes, IEEE Trans. Neural Netw. Learn. Syst. 33 (2022) 1–13.
[40] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, in: Computer Vision and Pattern Recognition, CVPR, Vol. June-2018, 2018, pp. 8697–8710.
[41] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, 2016, arXiv preprint arXiv:1611.01578.
[42] M. Jaderberg, K. Simonyan, A. Zisserman, K. Kavukcuoglu, Spatial transformer networks, in: Advances in Neural Information Processing Systems, Vol. 28, 2015, pp. 2017–2025.
[43] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (1998) 2278–2323.
[44] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images.(2009), Vol. 1, Cs.Toronto.Edu, 2009, pp. 1–58.

[45] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: A large-scale hierarchical image database, in: Computer Vision and Pattern Recognition, CVPR, Vol. June-2009, 2009, pp. 248–255.
[46] Z. Huang, N. Wang, Data-driven sparse structure selection for deep neural networks, in: Lecture Notes in Computer Science, Vol. 11220, 2018, pp. 317–334.
[47] M. Yin, Y. Sui, S. Liao, B. Yuan, Towards efficient tensor decomposition-based DNN model compression with optimization framework, in: Computer Vision and Pattern Recognition, CVPR, Vol. June-2021, 2021, pp. 10669–10678.
[48] Z. Cheng, B. Li, Y. Fan, Y. Bao, A novel rank selection scheme in tensor ring decomposition based on reinforcement learning for deep neural networks, in: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, Vol. May-2020, 2020, pp. 3292–3296.
[49] Y. Idelbayev, M. Carreira-Perpiñán, Low-rank compression of neural nets: Learning the rank of each layer, in: Computer Vision and Pattern Recognition, CVPR, Vol. June-2020, 2020, pp. 8046–8056.
[50] Y. Li, S. Gu, C. Mayer, L. Van Gool, R. Timofte, Group sparsity: The hinge between filter pruning and decomposition for network compression, in: Computer Vision and Pattern Recognition, CVPR, Vol. June-2020, 2020, pp. 8015–8024.
[51] Q. Zhao, G. Zhou, S. Xie, L. Zhang, A. Cichocki, Tensor ring decomposition, 2016, arXiv preprint arXiv:1606.05535. URL http://arxiv.org/abs/1606.05535.
[52] Y. Xu, Y. Li, S. Zhang, W. Wen, B. Wang, Y. Qi, Y. Chen, W. Lin, H. Xiong, TRP: Trained rank pruning for efficient deep neural networks, in: International Joint Conference on Artificial Intelligence, IJCAI, Vol. January-2021, 2020, pp. 977–983.
[53] X. Zhang, J. Zou, K. He, J. Sun, Accelerating very deep convolutional networks for classification and detection, IEEE Trans. Pattern Anal. Mach. Intell. 38 (2016) 1943–1955.
[54] J. Guo, K. Han, H. Wu, Y. Tang, X. Chen, Y. Wang, C. Xu, CMT: Convolutional neural networks meet vision transformers, in: Computer Vision and Pattern Recognition, CVPR, Vol. June-2021, 2021, pp. 12175–12185, URL http://arxiv.org/abs/2107.06263.
[55] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, Z. Tu, Deeply-supervised nets, in: Artificial Intelligence and Statistics, AISTATS, Vol. May-2015, 2015, pp. 562–570.
[56] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International Conference on Machine Learning, ICML, Vol. 1, 2015, pp. 448–456.
[57] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, J. Mach. Learn. Res. 15 (2011) 315–323.
[58] M. Lin, Q. Chen, S. Yan, Network in network, in: International Conference on Learning Representations, ICLR, vol. April-2014, 2014.
[59] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, J. Mach. Learn. Res. 9 (2010) 249–256.