IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 13 January 2023, accepted 26 January 2023, date of publication 30 January 2023, date of current version 2 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3240784

RESEARCH ARTICLE

TT-MLP: Tensor Train Decomposition on Deep MLPs

JIALE YAN[1], KOTA ANDO[2], (Member, IEEE), JAEHOON YU[1], (Member, IEEE), AND MASATO MOTOMURA[1], (Fellow, IEEE)

[1]Tokyo Institute of Technology, Yokohama, Kanagawa 226-8502, Japan
[2]Faculty of Information Science and Technology, Hokkaido University, Sapporo, Hokkaido 060-0814, Japan

Corresponding author: Masato Motomura (motomura@artic.iir.titech.ac.jp)

This work was partially supported by JST CREST Grant Number JPMJCR18K3, Japan.

ABSTRACT Deep multilayer perceptrons (MLPs) have achieved promising performance on computer vision tasks. Deep MLPs consist solely of fully-connected layers as the conventional MLPs do but adopt more sophisticated network architectures based on mixer layers composed of token-mixing and channel-mixing components. These architectures enable deep MLPs to have global receptive fields, but the significant increase of parameters becomes a massive burden on practical applications. To tackle this problem, we focus on using tensor-train decomposition (TTD) for compressing deep MLPs. At first, this paper analyzes deep MLPs under conventional TTD methods, especially using various designs of a macro framework and micro blocks: The former is how to concatenate mixer layers, and the latter is how to design a mixer layer. Based on the analysis, we propose a novel TTD method named *Train-TTD-Train*. The proposed method exerts the learning capability of channel-mixing components and improves the trade-off between accuracy and size. In the evaluation, the proposed method showed a better trade-off than conventional TTD methods on ImageNet-1K and achieved a 0.56% higher inference accuracy with a 15.44% memory reduction on Cifar-10.

INDEX TERMS Tensor-train decomposition, low-rank approximation, deep neural networks, deep multilayer perceptron, network parameter compression.

I. INTRODUCTION

Deep learning has achieved state-of-the-art performance in computer vision. With the development of high-performance and larger-scale datasets, we have witnessed the trend shift from multilayer perceptron (MLP) to convolutional neural networks (CNNs) [1], [2], [3], [4], and further to Vision Transformer (ViT) [5], [6], [7].

MLPs are the most classical neural networks composed of multiple layers that are pairs of linear and nonlinear transformations. Although their structure was the key to solving linearly inseparable problems, it was not possible to exploit their potential in the 1970s due to insufficient computational capability and the absence of proper training algorithms. Consequently, MLPs lost the interest of researchers and were practically forgotten for decades.

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

In recent years, MLP-based deep neural networks have captured our attention once again, which is triggered by the advent of MLP-Mixer [8]. Compared with the conventional MLP [9], [10], [11], MLP-Mixer adopted deep layers, showed higher inference accuracy, and boosted it to a comparable level with ViT. MLP-Mixer achieved this accuracy boost by applying more hidden layers to a sequence of non-overlapping patches cropped from an input image, which is a much simpler strategy than that of ViT. After MLP-Mixer, a series of subsequent studies succeeded it, such as ResMLP [12], gMLP [13], MetaFormer [14], and they have successfully improved accuracy by using deeper MLP structures.

Deep MLPs benefit from the global receptive fields created by FC layers, but on the other hand, they also suffer from the memory requirements of the FC layers. In contrast to convolution layers sharing weights within a kernel, an FC layer has an unshared weight for each pair of pre- and post-synaptic neurons. So, the number of its weights equals the

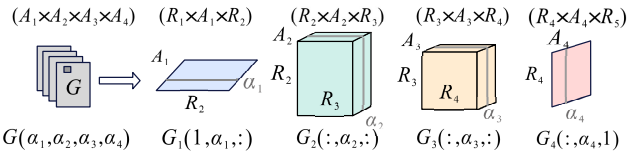


FIGURE 1. Example of TTD: The gray tensor is decomposed into four TT-cores. The first and last TT-cores have a 2D shape, and the second and the third TT-cores have a 3D shape, where A_i is the i-th dimensionality of the gray tensor, and R_j is the j-th TT-rank, in which R_1 and R_5 always equal to 1.

product of both numbers of pre- and post-synaptic neurons, becoming a large burden on memory resources. To tackle this issue, it is necessary to find a method to reduce memory requirements for FC layers without an accuracy drop.

Tensor-Train Decomposition (TTD) [15] could be a good solution for this issue. TTD is a low-rank approximation for tensors, and as in FIGURE 1, the key idea is decomposing a higher-rank tensor $\mathbf{G}$ with elements $G(\alpha_1, \alpha_2, \ldots, \alpha_d)$ into tensor-train (TT) cores G_i, a series of lower-rank tensors, with elements $G_i(r_i, \alpha_i, r_{i+1})$:

$$\begin{aligned} &G(\alpha_1, \alpha_2, \ldots, \alpha_d) \\ &\quad \approx G_1(1, \alpha_1, :)\, G_2(:, \alpha_2, :) \ldots G_d(:, \alpha_d, 1) \\ &\quad = \sum_{r_2, \ldots, r_d}^{R_2, \ldots, R_d} G_1(1, \alpha_1, r_2) \ldots G_d(r_d, \alpha_d, 1), \end{aligned} \tag{1}$$

where α_i is the i-th dimension index of tensor $\mathbf{G}$, and R_j is the j-th TT-rank that decides the size of TT-core G_i, in which R_1 and R_{d+1} are always equal to 1. In [16], Novikov et al. successfully convert the dense weight matrices of CNN's FC layers to the TT-format based on TTD, where TTD enables a higher compression ratio than pruning for FC layers: TTD achieves 5 000 – 700 000× compression, while a pruning method [17] achieves only 4 – 25×.

There are many other studies that use TTD in connection with parameter reduction [18], [19], [20], [21], [22], [23], [24]. Similar to [16], studies [18], [19] realized parameter reduction by random initialization and learning of the TT matrix. Among them, Qi et al. [18] decomposed several FC layers in spoken command recognition tasks. It showed that TTD made a good effect when decomposing a small part of a CNN framework. Yang et al. [19] enlarged the decomposition scope by applying TTD to an RNN's main structure and achieved more than 1 000× parameter reduction. It showed the TTD's potential ability if applied to the whole network. Rather than decomposing neural networks, Bigoni et al. [20] combined the discrete TT format with spectral theory to solve high-dimensional mathematical problems. It adopted the TT-DMRG-cross algorithm [21] rather than TT-SVD algorithm [15]. Based on TTD's wide usage, some studies [22], [23], [24] developed good libraries about TTD. T3F [22] sped up tensor calculus by adopting Riemannian optimization under TensorFlow. Tntorch [24] supported the PyTorch framework without Riemannian optimization. Mpnum [23] supported the TT-DMRG-cross algorithm.

However, an open problem is how to apply these methods to the deep MLPs without an accuracy drop. Although [20] showed that TTD has potential capabilities for high-dimensional problems in mathematics, its applicability to neural networks is unknown. For TTD studies [16], [18] on CNNs, the decomposing scopes are limited to several FC layers. Although Yang et al. [19] already showed that TTD enables compressing the entire RNNs, the accuracy drop of applying TTD to other networks' main structures is unknown. Moreover, deep MLPs' heterogeneous structure, the combination of token-mixing and channel-mixing components, makes it complicated because both components show different sensitivity against the low-rank approximation. Consequently, a larger TTD space has to be explored for deep MLPs than CNNs. To solve this issue, we can utilize the key ideas from these TTD libraries [22], [23], [24]. Our proposed idea is taking advantage of TT-SVD and hybridizing different TTD methods into deep MLPs. The libraries are not directly suited to deep MLPs, so we developed the initial idea and implemented them tightly with deep MLPs.

To exploit the capability of TTD for deep MLPs, we implement various TTD methods on deep MLPs and analyze their effects. Furthermore, we propose a novel TTD method, *Train-TTD-Train*, based on the analysis. The contributions of this paper are summarized as follows:

1) To the best of our knowledge, this is the first study of TTD to decompose deep MLPs,
2) this paper compares existing TTD methods on deep MLPs for the first time,
3) this paper analyzes the characteristics of deep MLPs under TTD to maximize compression ratio without accuracy drop, and
4) we propose a novel TTD method for deep MLPs, called Train-TTD-Train.

The rest of this paper is organized as follows. Section II explains deep MLPs and TTD to help understand this study. Section III describes the proposed TTD methodology for deep MLPs, and Section IV provides experimental results and analysis. Finally, Section V concludes this paper.

II. RELATED WORK

For further discussion, it is essential to understand the details of deep MLPs and TTD. This section explains the basic architecture of deep MLPs and their structural differences, introduces the previous studies on TTD for deep neural networks (DNNs), and describes their limitations.

A. DEEP MLPs

As in FIGURE 2, deep MLPs share a basic architecture that consists of four parts: a per-patch FC layer, N mixer layers, a pooling layer, and a classifier head. A deep MLP divides an input image into patches, the per-patch FC layer embeds each patch into a token, and the mixer layers transform the

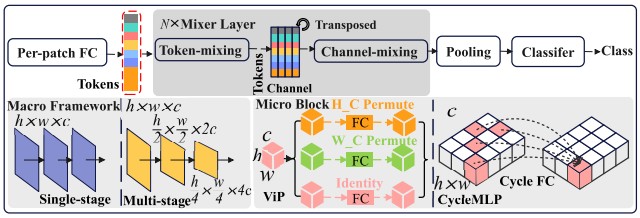


FIGURE 2. Overview of deep MLPs: A deep MLP consists of a per-patch FC layer, mixer layers, a pooling layer, and a classifier head. The distinctive feature of each deep MLP comes from how to design the macro framework concatenating mixer layers and the micro blocks composing a mixer layer.

tokens to generate a feature map. After that, the classifier head following the pooling layer obtains the final classification.

The structural differences between deep MLPs mainly come from the design of mixer layers. Deep MLPs have mixer layers consisting of FC layers with additional operations in common but adopt different designs for the macro framework concatenating mixer layers and the micro blocks composing a mixer layer. Consequently, these structural differences affect the results when we apply TTD to deep MLPs.

Single-stage and multi-stage designs are the current main streams for the macro framework. The single-stage design adopts mixer layers with the input and the output tensors of an identical size [8], [12], [13]. In contrast, the multi-stage design uses a hierarchical pyramid of mixer layers, where a deeper mixer layer has an input tensor with a spatially lower resolution with more channels [25], [26], [27] or vice versa. Recent work tends to prefer the multi-stage design over the single-stage design because the hierarchical pyramid performs better accurately in some cases.

On the other hand, the micro block has more variations than the macro framework — a mixer layer consists of token-mixing and channel-mixing blocks in common, but its details are different. For instance, ViP [28] has a token-mixing block containing three mixing branches, element-wise addition to combine their outputs, and an FC layer for feature fusion. This multi-branch mixing enables ViP to extract more feature information and boost its accuracy. Also, CycleMLP [27] utilizes a Cycle FC layer for mixing, which conducts MLP operation over points sampled in a cyclical style along the channel dimension. By using Cycle FC, CycleMLP successfully reduces both parameters and FLOPs simultaneously.

B. TENSOR-TRAIN FOR DEEP NEURAL NETWORKS

TTD is a tensor decomposition method proposed by [15]. It is based on a low-rank approximation of unfolding matrices and decomposes a tensor into a train of low-rank tensors called TT-cores, as its name "tensor-train" suggests.

Recent studies have focused on the low-rank approximation and proposed TTD methods for compressing DNNs [16], [18], [19], [22], [23], [24]. The TTD methods proposed in the previous studies can be categorized into two types: TTD-after-train and TTD-before-train. For the convenience of explanation, we call the two types Train-TTD and TTD-Train, respectively. Some previous studies also contain both of them in experiments. As summarized in TABLE 1, Train-TTD is the type applying TTD to a pre-trained model; fine-tuning can follow TTD as well [18], [22], [23], [24]. On the other hand, TTD-train is the type that trains a model in TT-format from scratch [16], [18], [19], [22], [23], [24]. As in TABLE 1, the proposed method is a combination of both types, but its details will be explained in Section III.

TABLE 1. Comparison between two types of the previous TTD methods (Train-TTD / TTD-train) and the proposed method (Train-TTD-Train).

TTD Type	Description
Train-TTD [18], [22]–[24]	Applying TTD to pre-trained models
TTD-Train [16], [18], [19], [22]–[24]	Training TT-format neural networks from scratch
Train-TTD-Train (proposed)	Training TT-format neural networks obtained by Train-TTD from scratch

The conventional methods successfully reduced the parameters of recurrent neural networks (RNNs) and convolutional neural networks (CNNs). Reference [19] achieved more than 1 000× parameter reduction of the input-to-hidden layers on RNNs. References [16], [18] also achieved up to 7× and 4× parameter reduction on CNNs, where these numbers are the compression ratio of the whole networks, and TTD decomposes only FC layers.

Despite their high compression performance, these methods suffer from accuracy degradation. TTD-Train is difficult to achieve higher accuracy because TTD-Train limits TT-ranks before training. To be more specific, the models' capacity is limited and closely related to TT-rank values. Among previous studies, most of TTD-Train studies [16], [18], [19] usually adopt balanced TT-rank values, while using TT-SVD results in unbalanced TT-rank values. This balanced setting is not mathematically consistent with the decomposition attribute and makes training much more susceptible to initialization. Also, the approximation error is an inevitable issue for Train-TTD. Although Train-TTD provides a model with a good initialization by directly decomposing a full-rank model, the decomposed model is still difficult to be recovered even after fine-tuning. The previous study [18] uses two TTD methods in the experiments, but the results are very close without any in-depth analysis. Even tensor-ring, a Train-TTD-like SOTA method, reports that it suffers from a 0.88% accuracy drop with a 2.2× compression ratio for ResNet32 on Cifar-10 [29]. Moreover, accuracy degradation can be worse on deep MLPs because their structures and blocks essentially differ from RNNs and CNNs.

III. PROPOSED DEEP TENSOR-TRAIN MLPs

Conveniently, deep MLPs consist solely of FC layers that are the exact target of the previous TTD studies, but the unique structure of their mixer layers is an unexplored area for TTD. This section introduces the proposed method named *Train-TTD-Train* and explains how Train-TTD-Train builds the TT-format from each FC layer and trains deep TT-MLPs.

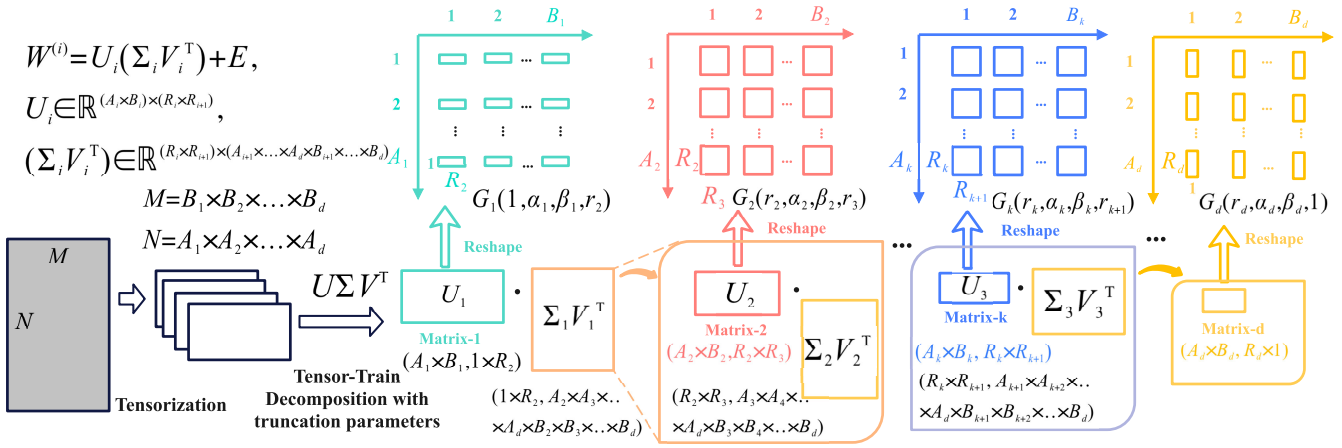


FIGURE 3. FC layer in TT-format: TT-cores are decomposed iteratively. The gray matrix is the original weight matrix. Firstly, its width and length values are factorized. Then it is decomposed by the TT-SVD algorithm iteratively. During each iteration, the U matrix is reshaped into a 4D TT-core, and the remaining matrix continues to be decomposed. For example, at the first iteration, the light green Matrix-1 U_1 is reshaped into a 4D TT-core, and the orange matrix $\Sigma_1 V_1^{\top}$ continues to be decomposed. Subsequent operations follow this way.

A. FULLY-CONNECTED LAYER IN TRAIN-TTD-TRAIN

FC Layer is the basic building block of deep MLPs, projecting an input vector $x \in \mathbb{R}^N$ into an output vector $y \in \mathbb{R}^M$ linearly as

$$y = Wx + b, \tag{2}$$

where $W \in \mathbb{R}^{M\times N}$ is a weight matrix and $b \in \mathbb{R}^M$ is a bias vector. We call it the low-to-high layer when $M > N$ and the high-to-low layer when $M < N$ in this paper.

As shown in FIGURE 3, Train-TTD-Train follows tensor-train singular value decomposition (TT-SVD) to convert FC layers in TT-format. Its processes are as follows. (i) First, it tensorizes the weight matrix $W \in \mathbb{R}^{M\times N}$ into a weight tensor

$$\mathbf{W} \in \mathbb{R}^{A_1\times A_2\times\ldots A_d\times B_1\times B_2\times\cdots\times B_d}, \tag{3}$$

where $N = \prod_{i=1}^{d} A_i$ and $M = \prod_{i=1}^{d} B_i$. Since W and $\mathbf{W}$ have identical elements, they are interconvertible. (ii) Second, it initializes $i = 1$ and obtains the initial decomposition target matrix $W^{(1)}$ from the weight tensor $\mathbf{W}$:

$$W^{(1)} \in \mathbb{R}^{(A_1\times B_1)\times(A_2\times\cdots\times A_d\times B_2\times\ldots B_d)}, \tag{4}$$

where the input and the output dimensions of the weight matrix W are rearranged and will be decomposed simultaneously. (iii) Third, $W^{(i)}$ is decomposed in the SVD manner. The upper bound of approximation error is $\frac{\epsilon}{\sqrt{d-1}}\|W\|_F$ [15], where ϵ is a prescribed accuracy and $\|\cdot\|_F$ is the Frobenius norm. In addition, each rank value R_i is constrained to be below the maximum value R_{max}. R_{max} is defined based on compression requirements:

$$\begin{aligned} W^{(i)} &= U_i\Sigma_i V_i^{\top} + E,\ \|E\|_F \le \frac{\epsilon}{\sqrt{d-1}}\|W\|_F \\ &\text{s.t.}\quad U_i \in \mathbb{R}^{(A_i\times B_i)\times(R_i\times R_{i+1})}, \\ \Sigma_i V_i^{\top} &\in \mathbb{R}^{(R_i\times R_{i+1})\times(A_{i+1}\times\cdots\times A_d\times B_{i+1}\times\cdots\times B_d)}, \\ R_i &\le R_{max}. \end{aligned} \tag{5}$$

(iv) The matrix U_i is reshaped into a TT-core G_i:

$$G_i \in \mathbb{R}^{R_i\times A_i\times B_i\times R_{i+1}}. \tag{6}$$

(v) It defines $W^{(i+1)} = \Sigma_i V_i^{\top}$ and updates $i = i+1$. (vi) The processes (iii) – (v) are repeated d times. (vii) As a result, the sequence of d TT-cores $(G_1, \ldots, G_d)$ is obtained.

In the final TT-format of an FC layer, the output $y \in \mathbb{R}^{(B_1\times\cdots\times B_d)}$ with elements $y(\beta_1, \ldots, \beta_d)$ can be written as

$$\begin{aligned} &y(\beta_1, \ldots, \beta_d) \\ &= \sum_{\alpha_1,\ldots,\alpha_d}^{A_1,\ldots,A_d} G_1(:,\alpha_1,\beta_1,:)\ldots G_d(:,\alpha_d,\beta_d,:)x(\alpha_1,\ldots,\alpha_d) \\ &\quad + b(\beta_1,\ldots,\beta_d). \end{aligned} \tag{7}$$

where A_i and B_i are hyperparameters, and TT-SVD decides the TT-rank factors R_i that adjust the trade-off between compression ratio and approximation capability.

B. TRAINING DEEP TENSOR-TRAIN MLPs

Train-TTD-Train adopts a combined training flow of Train-TTD and TTD-Train. As shown in FIGURE 4, two significant differences exist between both processing flows. One is the order of training and TTD: Train-TTD performs TTD after training and TTD-train vice versa. The other is how to define TT-ranks: TT-SVD defines TT-ranks in Train-TTD, and TT-ranks are hyperparameters in TTD-train. As mentioned in Section II-B, TTD-after-Training causes the approximation error, and user-defined TT-ranks cause the dependency on initialization.

To complement the shortcomings of both approaches, Train-TTD-Train builds TT-format models in the same manner as Train-TTD. After getting a TT model whose compression ratio is bigger than 1, it initializes the whole network's weights and trains them from scratch. In addition, our proposed method adopts a convergence-first scheme during the decomposition process. Initially, it prepares

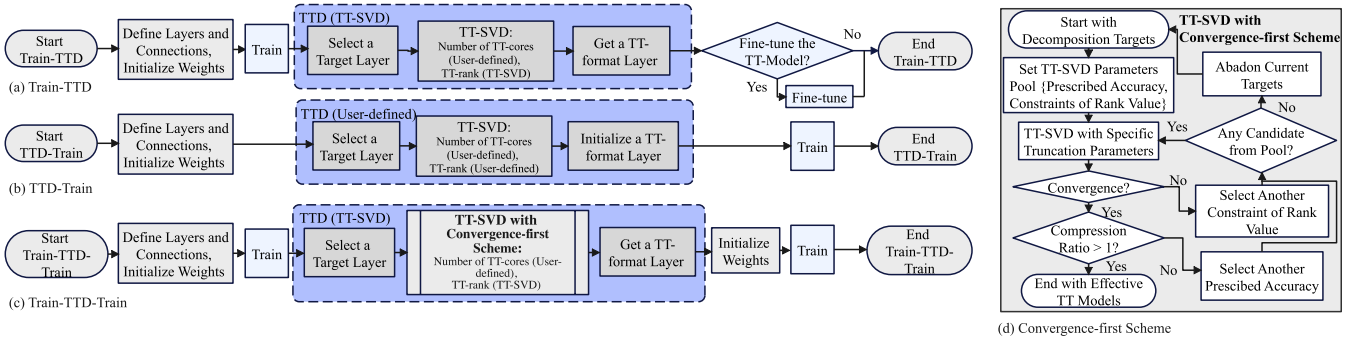


FIGURE 4. Processing flows of Train-TTD, TTD-Train, Train-TTD-Train, and convergence-first scheme: (a) Train-TTD includes fine-tuning and without fine-tuning options. Both of them start by defining networks and initializing weights. Then they train a model and implement the TTD (SVD). Finally, the network will be fine-tuned or not be fine-tuned. TT hyperparameters like TT-ranks come from the TT-SVD algorithm. (b) TTD-Train starts the same as Train-TTD but adopts TTD without training before. Thus, TT hyperparameters are defined by users. Finally, it trains the model from scratch. (c) Train-TTD-Train follows the same as the Train-TTD method at the beginning. The difference is that it only inherits TT hyperparameters: TT-ranks and the number of TT-cores, but initializes the network's weights. Finally, it trains the TT-model from scratch. (d) The convergence-first scheme illustrates details about converge process in the Train-TTD-Train method.

various truncation parameters, including prescribed accuracy and constraints of rank value. It checks convergence first and then the compression ratio during the decomposing process. If it fails to converge, it will select other constraints of rank values (bigger ones). Or if it fails to compress the model size, it will choose another prescribed accuracy until there are no candidates from the parameter pool. By doing so, it is possible to avoid using user-defined TT-ranks and satisfy the upper bound of approximation errors. The proposed Train-TTD-Train resembles Train-TTD with fine-tuning except for the additional initialization, but it shows better performance than Train-TTD with fine-tuning in our experimental results.

Also, to keep inference accuracy, Train-TTD-Train constrains the TT-ranks, R_i, and the hyperparameters, A_i and B_i, in a descending style:

$$A_i \geq A_{i+1}, \quad B_i \geq B_{i+1}, \quad R_i \geq R_{i+1} \tag{8}$$

except $R_1 = R_{d+1} = 1$. Because TTD is an iterative decomposition method, approximation error accumulates along the iterations, which means a TT-core G_{i+1} has more risk of error than a TT-core G_i. The constraints above guarantee the extraction of larger TT-cores, at least the same size as adjacent TT-cores. In earlier decomposition, being more advantageous to keep the original feature information.

IV. EXPERIMENTS

To evaluate and reveal deep MLPs' characteristics under TTD methods, we implemented different TTD methods for deep MLPs and analyzed their performance. This section shows the settings for experiments and analysis of deep MLP's characteristics under the TTD application. Moreover, it describes deep MLP's performance under our proposed Train-TTD-Train method and reveals general rules about TTD.

A. EXPERIMENTAL SETTINGS

This subsection describes baseline models, datasets, and decomposition settings.

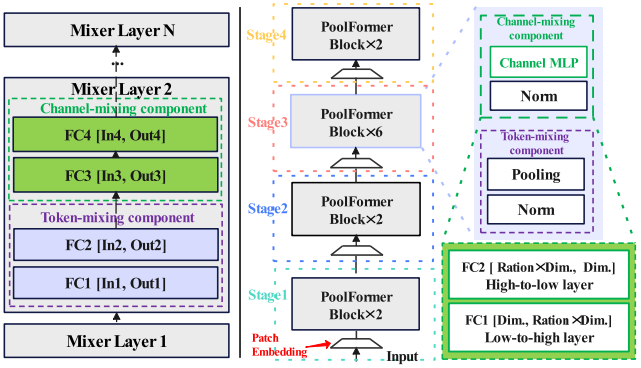


FIGURE 5. Baseline model structures including MLP-Mixer (left) and MetaFormer (right): MLP-Mixer has two configurations. Each configuration is represented as [N, In1, Out1, In2, Out2, In3, Out3, In4, Out4]. The configuration about MLP-Mixer-6 is: [6, 64, 64, 64, 64, 128, 128, 128, 128]. For MLP-Mixer-8, it is: [8, 49, 256, 256, 49, 512, 2048, 2048, 512]. Each mixer layer has a channel-mixing component and a token-mixing component. MetaFormer has four stages. Each stage's configuration is represented as [Token, MLP Ratio, Embed., Dim.]: stage1 [$H/4 \times W/4$, 4, 64], stage2 [$H/8 \times W/8$, 4, 128], stage3 [$H/16 \times W/16$, 4, 320], and stage4 [$H/32 \times W/32$, 4, 512].

1) BASELINE MODELS

MLP-Mixer [8] and MetaFormer [14] are used as benchmarks. Their structures are shown in FIGURE 5. MLP-Mixer is the pioneering deep MLP with a single stage and has classical characteristics. In our settings, MLP-Mixer has two configurations. One (MLP-Mixer-6) adopts six identical mixer layers for Cifar-10, and another (MLP-Mixer-8) adopts eight identical mixer layers for ImageNet-1K. Each mixer layer includes token-mixing and channel-mixing components. In the following experiments, channel-mixing layers represent FC layers belonging to the channel-mixing component, and token-mixing layers represent FC layers belonging to the token-mixing component. MetaFormer uses a pyramidal structure with smaller patch size and higher feature fitness, representing newly developed deep MLPs. PoolFormer-S12 [14] is used as a typical MetaFormer model

with four stages. Each stage has several PoofFormer blocks. In a PoolFormer block, there is one pooling layer as a token-mixing component and two channel-mixing FC layers as a channel-mixing component. Specifically, one is a low-to-high FC layer, and another is a high-to-low FC layer. We use the PoolFormer-S12 for experiments on Cifar-10 and ImageNet-1K datasets.

2) DATASET

The training datasets are Cifar-10 [30] and ImageNet-1K [31]. Cifar-10 contains 50 000 training images and 10 000 test images. ImageNet-1K contains 1.3M training examples and 1 000 classes. For experiments on CIFAR-10, MetaFormer uses a pyramidal structure, requiring big input feature maps. Therefore, we implemented preprocessing for the dataset by resizing the original input shape from 32×32 to 256×256. MLP-Mixer still receives the original input size. For experiments on ImageNet-1K, we use a data loading mechanism from FFCV [32] to speed up training.

3) DECOMPOSITION SETTINGS

The experiments set different prescribed accuracy ϵ [15] when implementing TT-SVD toward a target A. The ϵ influences the truncation threshold δ by $\delta = \epsilon \|A\|_F$. Therefore, TT-MLP models will have different compression ratios (CRs) if implemented with different ϵ. Our experiments set $\epsilon \in \{1.50, 1.25, 1.00, 0.75, 0.50, 0.25, 0.10\}$ for MLP-Mixer, and $\epsilon \in \{1.00, 0.85, 0.75, 0.50, 0.25, 0.10\}$ for MetaFormer. We set 65 and 130 as the constraints to limit the TT-rank values for MLP-Mixer on Cifar-10 and ImageNet-1K, respectively. We also set 130 for MetaFormer models on both Cifar-10 and ImageNet-1K. These settings guarantee each decomposition convergent into an upper bound but cause some ineffective compression models, whose compression ratios are smaller than 1.0 shown in TABLE 2 and TABLE 3. MLP-Mixer-6 is trained from scratch within 900 epochs on Cifar-10 and MLP-Mixer-8 is trained within 120 epochs on ImageNet-1K. MetaFormer is trained from scratch within 1600 and 300 epochs on Cifar-10 and ImageNet-1K, respectively. These models are used as pre-trained models in the Train-TTD and Train-TTD-Train methods. For speeding up the TTD operations, we use an optimized einsum operation [33].

4) HARDWARE SETTING

The GPU device is the TESLA V100, and its memory capacity is 32G. The CPU device is the Intel(R) Xeon(R) CPU E5-2698 v4 at 2.20GHz. The experiments are implemented on these devices.

B. PREPARATION FOR ANALYSIS

Before analyzing experimental results, this subsection describes the baseline accuracy for comparison and effective compression with different prescribed accuracy ϵ.

TABLE 2. Compression ratio and ϵ about layers of MLP-Mixer.

Training Dataset	Prescribed Accuracy	Compression Ratio	
		Token Layers	Channel Layers
Cifar-10	$\epsilon = 1.50$	48.8	170.7
	$\epsilon = 1.25$	12.6–25.0	25.0–33.0
	$\epsilon = 1.00$	4.6–16.7	10.1–11.9
	$\epsilon = 0.75$	1.9–8.4	2.3–2.7
	$\epsilon = 0.50$	1.1–3.8	1.2–1.4
	$\epsilon = 0.25$	0.8–1.7	0.8–0.9
	$\epsilon = 0.10$	0.6–0.9	0.8
ImageNet-1K	$\epsilon = 1.50$	108.1	3276.8
	$\epsilon = 1.25$	18.6–55.0	168.9–224.4
	$\epsilon = 1.00$	7.0–22.2	71.5–84.9
	$\epsilon = 0.75$	2.1–5.3	2.5–3.2
	$\epsilon = 0.50$	1.1–2.0	1.4–1.6
	$\epsilon = 0.25$	0.9–1.0	1.0–1.1
	$\epsilon = 0.10$	0.8–0.9	1.0

TABLE 3. Compression ratio and ϵ about the stages of MetaFormer.

Training Dataset	Prescribed Accuracy	Compression Ratio			
		Stage-1	Stage-2	Stage-3	Stage-4
Cifar-10	$\epsilon = 1.00$	19.3	26.0–28.0	36.3–40.9	71.5–83.2
	$\epsilon = 0.85$	4.1	4.5–4.7	4.2–6.2	4.3–5.2
	$\epsilon = 0.75$	2.6	2.6–3.0	2.6–3.0	2.5–3.0
	$\epsilon = 0.50$	1.29–1.34	1.31–1.38	1.35–1.43	1.37–1.48
	$\epsilon = 0.25$	0.96	1.0	1.03	1.05–1.08
	$\epsilon = 0.10$	0.92	0.92	0.97	0.98
ImageNet-1K	$\epsilon = 1.00$	19.3–20.9	25.9–26.9	36.3–39.2	70.3–74.1
	$\epsilon = 0.85$	4.2	4.1–4.2	3.8–4.7	4.0–4.7
	$\epsilon = 0.75$	2.5	2.5–2.7	2.4–2.7	2.4–2.6
	$\epsilon = 0.50$	1.29	1.33	1.36	1.37
	$\epsilon = 0.25$	0.96	1.00	1.02–1.03	1.05
	$\epsilon = 0.10$	0.93	0.93	0.97	0.98

1) ACCURACY OF PRE-TRAINED MODELS

MetaFormer and MLP-Mixer are trained from scratch. On Cifar-10, MLP-Mixer achieves 80.69% top-1 accuracy with 0.259M parameters, and MetaFormer achieves 96.91% top-1 accuracy with 11.9M parameters. On ImageNet-1K, MLP-Mixer achieves 58.35% top-1 accuracy with 19.1M parameters, and MetaFormer achieves 73.81% top-1 accuracy with 11.9M parameters. The previous studies [8], [14] adopt many data augmentation techniques such as RandAugment [34], MixUp [35], CutMix [36] and more. Compared with them, the FFCV data loader uses half-precision (FP16) and only adopts random horizontal flips without other data augmentation operations. Therefore, our baselines' accuracy on ImageNet-1K is limited.

Generally, when we have limited data and classes, the k-fold cross-validation can be applied to find better hyper-parameters in practical applications [37]. However, it is not suitable for our study in terms of training time. For example, training a PoolFormer-TT model on ImageNet-1K (with a Train-TTD-Train method) takes 28394 s/epoch on a GPU. To explore the TT-MLP attributes with different TTD settings, we set a large amount of TTD experiments. There are various decomposition targets, different decomposition manners, multiple TTD settings, two datasets, and four baseline models. They are time-consuming TTD experiments with our limited computing resources. Instead, we did our best to generalize the models for practical applications, including extending the dataset from Cifar-10 to ImageNet-1K and using data augmentation.

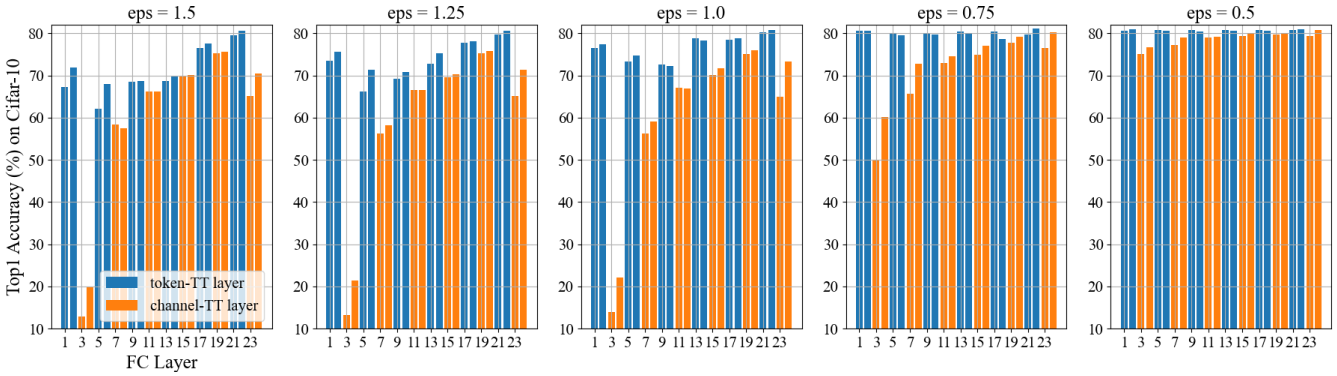


FIGURE 6. Train-TTD without Fine-tuning for MLP-Mixer-6 (FC-layer-by-FC-layer) on Cifar-10: The Top-1 accuracy of Cifar-10 dataset with different prescribed accuracy ϵ. The Train-TTD (without fine-tuning) method is implemented toward each FC layer of the MLP-Mixer. Each graph shows the inference accuracy (y-axis)-TT layer (x-axis). The blue histograms show the inference accuracy of decomposing a token-mixing layer, and the orange histograms are about decomposing a channel-mixing layer.

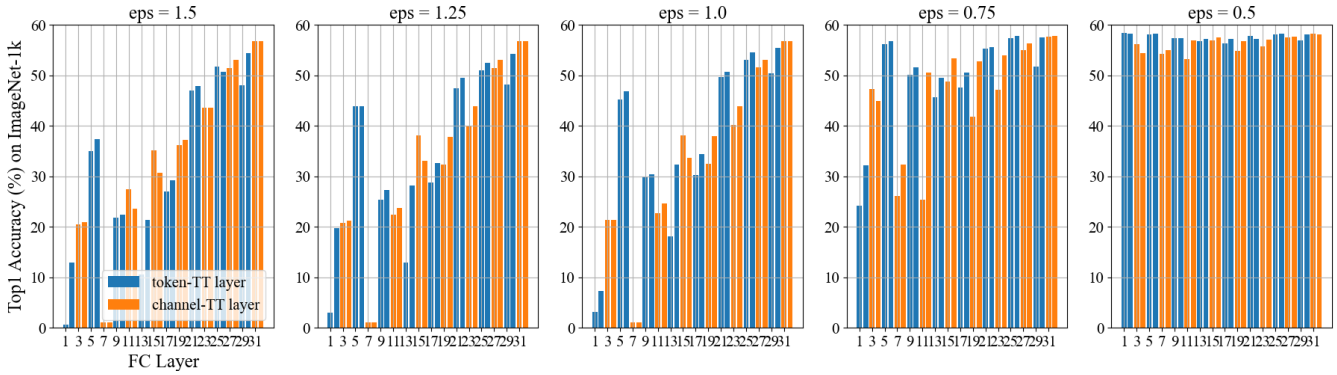


FIGURE 7. Train-TTD without Fine-tuning for MLP-Mixer-8 (FC-layer-by-FC-layer) on ImageNet-1K: The Top-1 accuracy of ImageNet-1K dataset with different prescribed accuracy ϵ. The Train-TTD (without fine-tuning) method is implemented toward each FC layer of the MLP-Mixer. Each graph shows the inference accuracy (y-axis)-TT layer (x-axis). The blue histograms show the inference accuracy of decomposing a token-mixing layer, and the orange histograms are about decomposing a channel-mixing layer.

2) COMPRESSION RATIO FOR BASELINE MODELS

Based on the proposed method, different prescribed accuracy results in various compression ratios for MLP-Mixer and MetaFormer. The results are shown in TABLE 2 and TABLE 3. For MLP-Mixer, we calculated the compression ratio of FC layers belonging to token-mixing and channel-mixing components, respectively. For MetaFormer, we calculated the compression ratio stage-by-stage. The results show that TTD takes no compression effect when $\epsilon \leq 0.25$, so the following discussion mainly focuses on the effective part.

C. DEEP MLPs' DIFFERENT CHARACTERISTICS AFFECT RESULTS OF TTD APPLICATIONS

This subsection describes the exploration and analysis of MLP-Mixer and MetaFormer. It mainly focuses on channel-mixing and token-mixing components for MLP-Mixer, and high-to-low and low-to-high layers for MetaFormer. In addition, the decomposition scope is enlarged from FC-layer-by-FC-layer to mixer-by-mixer and further to stage-by-stage.

1) CHANNEL-MIXING AND TOKEN-MIXING LAYERS

Channel-mixing layers are more sensitive, with higher approximation error under the TT decomposition. Train-TTD (without fine-tuning) is applied toward two pre-trained MLP-Mixer models: MLP-Mixer-6 and MLP-Mixer-8. Specifically, each MLP-Mixer model is decomposed in an FC-layer-by-FC-layer manner. It means converting an FC layer into a TT-format layer and maintaining other FC layers as original ones. Since every mixer layer has four FC layers, the accuracy changes of four FC layers in one mixer layer need attention. For convenience, we call a model a token-TT model or a channel-TT model if its FC layer in a token-mixing or channel-mixing component is decomposed. FIGURE 6 reflects gaps between the accuracy of the token-TT models (blue) and the channel-TT models (orange) for MLP-Mixer-6. The average accuracy gaps are 12.65%, 15.08%, 17.13%, 8.27%, 1.92% when $\epsilon = 1.5, 1.25, 1.0, 0.75, 0.5$, respectively. These accuracy gaps also exist in MLP-Mixer-8, as FIGURE 7 shows. Except for the case where the accuracy gap between the token-TT model and channel-TT model is -1.30% with $\epsilon = 1.5$, most cases show that token-TT models have better accuracy results. The

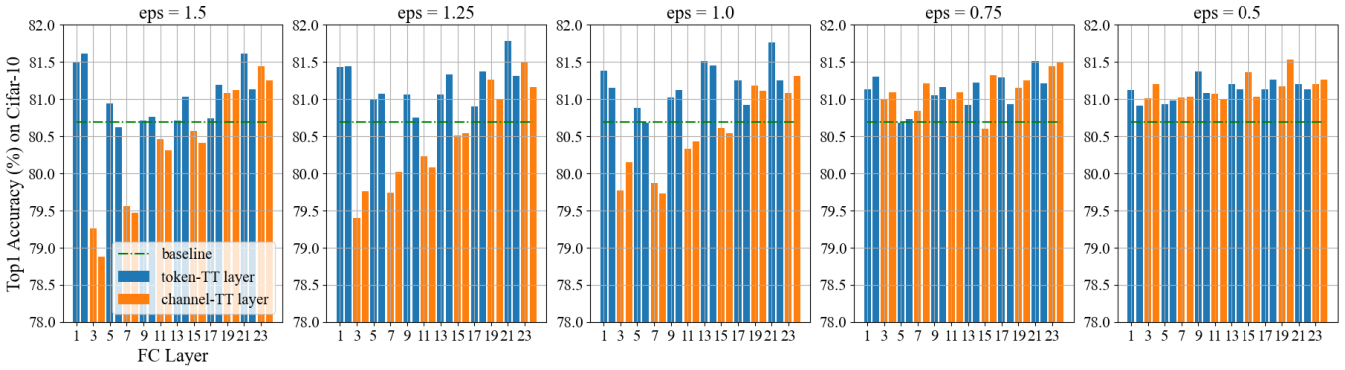


FIGURE 8. Train-TTD with Fine-tuning for MLP-Mixer-6 (FC-layer-by-FC-layer) on Cifar-10: The Top-1 accuracy of Cifar-10 dataset with different prescribed accuracy ϵ. Train-TTD (with fine-tuning) is implemented in each FC layer of MLP-Mixer. Each graph shows the inference accuracy (y-axis)-TT layer (x-axis). The blue histogram is for decomposing a token-mixing layer, the orange histogram is for decomposing a channel-mixing layer, and the green curve is for the baseline accuracy.

accuracy gaps are 2.17%, 3.43%, 3.05%, and 1.35% when $\epsilon = 1.25$, 1.0, 0.75, and 0.5, respectively. These results show that channel-mixing layers are more sensitive if the Train-TTD (without fine-tuning) method is directly implemented.

Channel-mixing layers have better learning capability to gain new features under TT decomposition. Train-TTD (with fine-tuning, 100 epochs) is applied toward a pre-trained MLP-Mixer model. As FIGURE 8 shows, the accuracy gaps between TT-token models and TT-channel models still exist when $\epsilon = 1.5$, 1.25, 1.0. However, when $\epsilon = 0.75$, 0.5, the previous accuracy gaps are not obvious anymore. The accuracy of channel-TT models can be better than the accuracy of token-TT models. This can be observed from the 2^{nd} mixer layer, the 5^{th} mixer layer and the 6^{th} mixer layer when $\epsilon = 0.75$, and the 1^{st} mixer layer, the 2^{nd} mixer layer, the 4^{th} mixer layer, the 5^{th} mixer layer and the 6^{th} mixer layer when $\epsilon = 0.5$. In detail, the average accuracy of channel-TT models is about 1.4% and 1.8% higher than the average accuracy of token-TT models, when $\epsilon = 0.75$, 0.5, respectively. This is because when ϵ becomes smaller, FC layers belonging to channel-mixing components can maintain more original information. In the meantime, they have better learning capability to acquire new features with fine-tuning.

2) HIGH-TO-LOW AND LOW-TO-HIGH LAYERS

High-to-low layers have strong robustness with smaller approximation errors, and low-to-high layers have better learning capability. We implemented Train-TTD (with/without fine-tuning) toward MetaFormer on both Cifar-10 (within 30 epochs) and ImageNet-1K (within 15 epochs). We calculated the average accuracy of models whose TT-FC layers belong to the same stage and divided them into high-to-low and low-to-high categories. As FIGURE 9 shows, the order of accuracy under different TTD methods follows as low-to-high (Train-TTD with fine-tuning, 1^{st}), high-to-lower (Train-TTD with fine-tuning, 2^{nd}), high-to-lower (Train-TTD without fine-tuning, 3^{rd}), low-to-high (Train-TTD without fine-tuning 4^{th}). Low-to-high layers

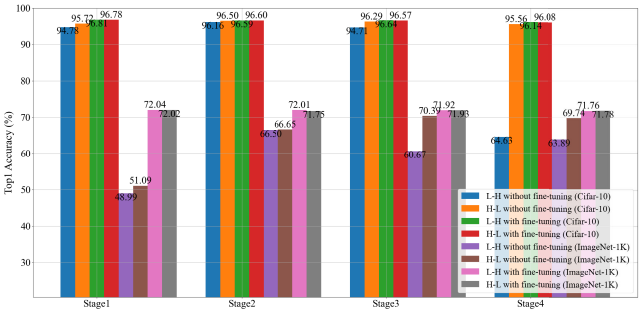


FIGURE 9. Low-to-high Layers and High-to-low Layers in MetaFormer with the Train-TTD Method (FC-layer-by-FC-layer, $\epsilon = 1.00$): Both Train-TTD (with fine-tuning) and Train-TTD (without fine-tuning) are implemented toward MetaFormer in an FC-layer-by-FC-layer manner. The average accuracy is about models whose TT-FC layer belongs to the same stage. The figure shows the top-1 inference accuracy (y-axis)-TT stage (x-axis) for decomposing different kinds of layers in different stages.

need to provide sufficient "scattered" information so that the next high-to-low layer can synthesize the information by "gathering" it. Under Train-TTD, decomposing low-to-high layers directly is much more sensitive. Therefore, the low-to-high (Train-TTD without fine-tuning) is the last one in the order. But a low-to-high layer has better learning capability with fine-tuning, which helps it become the first. For a high-to-low layer, it has a smaller approximation error and strong robustness. Therefore, TT models related to it are in the 2^{nd} and 3^{rd} order.

3) MIXER-BY-MIXER AND STAGE-BY-STAGE

The front mixer layers have worse robustness, and the final mixer layer has better learning capability. We implemented the Train-TTD and the Train-TTD-Train methods toward MLP-Mixer in a mixer-by-mixer manner both on Cifar-10 and ImageNet-1K. This manner means decomposing all FC layers that belong to one mixer layer and maintaining other FC layers. For Cifar-10, FIGURE 10 shows that the Train-TTD-Train and the Train-TTD (with fine-tuning, 100 epochs) methods both improve the accuracy compared with the Train-TTD (without fine-tuning) method. The

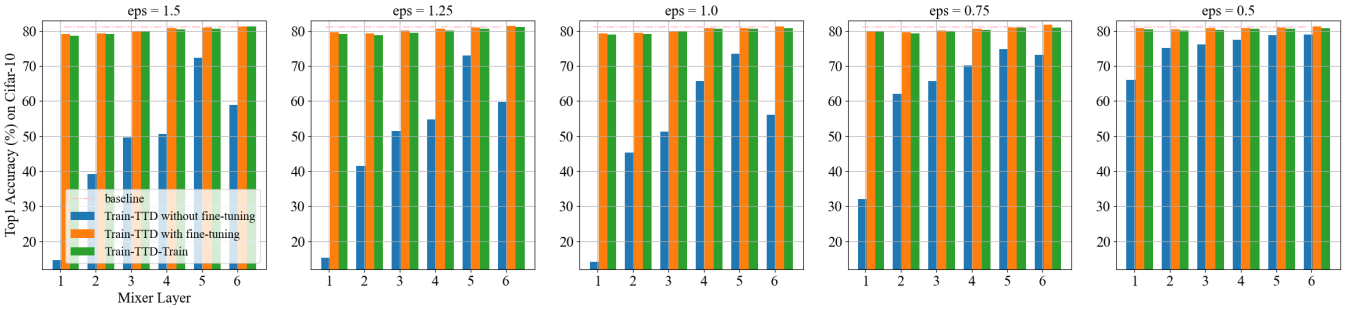


FIGURE 10. MLP-Mixer-6 with Different TTD methods in a Mixer-by-mixer Manner on Cifar-10: Train-TTD-Train, Train-TTD (without fine-tuning), and Train-TTD (with fine-tuning) are implemented to MLP-Mixer. The x-axis is about the mixer layer, which means using TT to decompose one mixer layer's whole FC layers. The y-axis is about the top-1 accuracy.

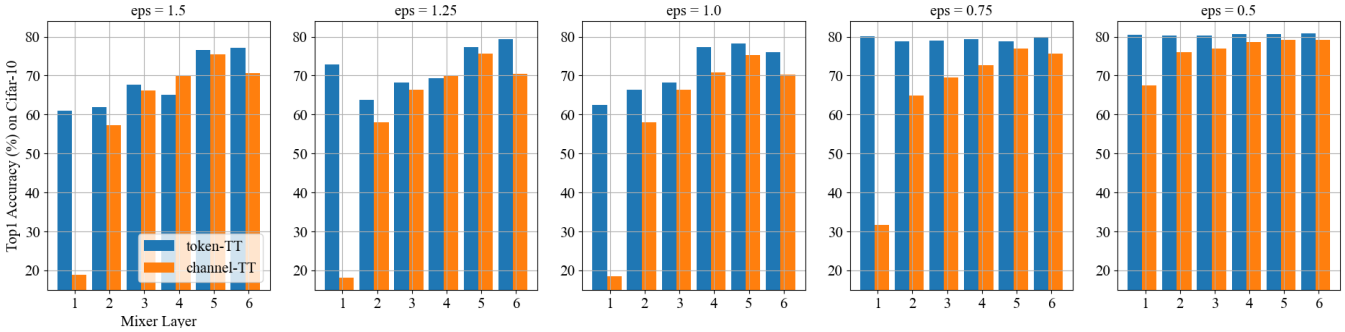


FIGURE 11. MLP-Mixer-6 with the Train-TTD (without fine-tuning) Method on Cifar-10: Channel-mixing layers and token-mixing layers are decomposed separately in a mixer-by-mixer manner. The blue histograms are about decomposing token-mixing layers, and the orange histograms are about decomposing channel-mixing layers.

common characteristic for all methods is that decomposing the first mixer layer causes the lowest accuracy. For Train-TTD-Train and Train-TTD (with fine-tuning), decomposing the final mixer layer leads to the highest accuracy compared with decomposing others. Some cases even achieve better accuracy than the baseline. Such as the TT models whose 6^{th} mixer layer is decomposed achieve 81.37%, 81.47%, 81.27%, 81.75%, and 81.23% top-1 accuracy when $\epsilon = 1.5$, 1.25, 1.0, 0.75, and 0.5, respectively in the Train-TTD (with fine-tuning) method.

If decomposed directly, the channel-mixing layers and token-mixing layers in front mixer parts are more sensitive and have worse approximation errors than others, especially channel-mixing layers. Considering that each mixer part has token-mixing layers and channel-mixing layers, we decomposed them separately with the Train-TTD (without fine-tuning) method and explored them in a mixer-by-mixer manner. FIGURE 11 reflects the MLP-Mixer characteristics when decomposed directly. The blue histograms are about decomposing token-mixing layers, and the orange histograms are about decomposing channel-mixing layers. The overall trend of the orange histograms is increasing from the first mixer layer to the last one. Overall, token-TT models perform with better accuracy than channel-TT models. In addition, whether for token-TT models or channel-TT models, the model whose first mixer layer is decomposed has smaller accuracy than others, especially for the channel-TT models. This phenomenon shows that if decomposed directly, both channel-mixing and token-mixing layers in front mixer parts have worse robustness than layers in subsequent parts.

Experiments on ImageNet-1K also have a similar phenomenon as FIGURE 12 shows. If both token-mixing and channel-mixing layers are decomposed in a mixer-by-mixer manner, the mixer layers that have subsequent positions have stronger robustness than the front layers. Without fine-tuning, the top1 accuracy for TT-Mixer layers 5, 6, 7, and 8 is higher than TT-Mixer layers 1, 2, 3, and 4. The average accuracy gaps between two groups are 32.70%, 32.16%, 32.01%, 24.56%, and 3.76% when $\epsilon = 1.5$, 1.25, 1.0, 0.75, and 0.5, respectively. The final mixer layer has better learning capability if implemented with fine-tuning (within 15 epochs). Specifically, decomposing the final mixer layer leads to the highest accuracy compared with decomposing other mixer layers for all ϵ values.

A stage with more PoolFormer blocks has worse robustness than other stages. Train-TTD (without fine-tuning) is implemented toward MetaFormer by decomposing one stage's whole FC layers. We call this kind of model as TT-stage model. As FIGURE 13 shows, the most telling feature of the figure is the improvement of the TT-stage3 models' accuracy with the reduction of ϵ. This is because the third stage has more PoolFormer blocks (six blocks) than other stages (two blocks). It is much more sensitive to be decomposed directly. When the ϵ reduces from 1.0 to 0.75 and further to 0.5, the TT-stage3 model can keep more original information and have better robustness. FIGURE 14

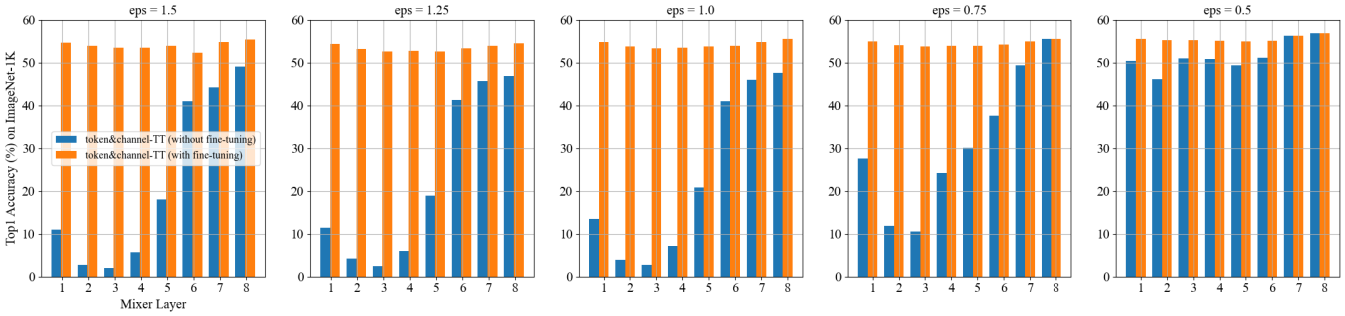


FIGURE 12. MLP-Mixer-8 with Different TTD Methods in a Mixer-by-mixer Manner on ImageNet-1K: Channel-mixing layers and token-mixing layers are decomposed in a mixer-by-mixer manner. The blue histograms are about decomposing them without fine-tuning, and the orange histograms are about decomposing them with fine-tuning (15 epochs).

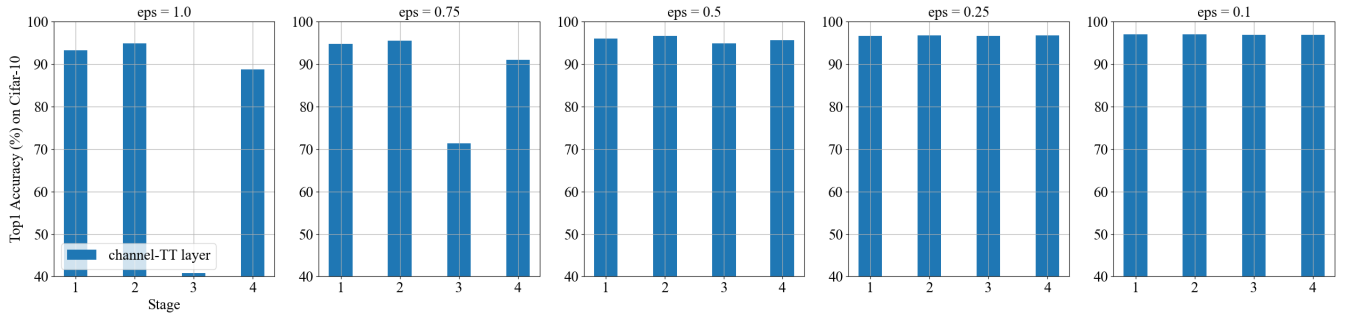


FIGURE 13. MetaFormer with the Train-TTD (without fine-tuning) Method on Cifar-10: We implemented the Train-TTD method in a stage-by-stage manner. It decomposes one stage's whole FC layers. The y-axis is about Top-1 inference accuracy, and the decomposed stage is shown in the x-axis.

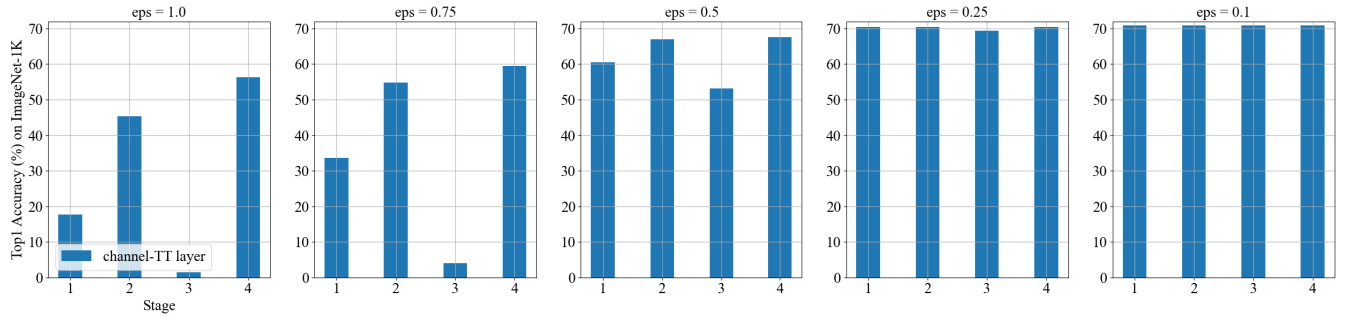


FIGURE 14. MetaFormer with the Train-TTD (without fine-tuning) Method on ImageNet-1K: We implemented the Train-TTD method in a stage-by-stage manner. It decomposes one stage's whole FC layers. The y-axis is about Top-1 inference accuracy, and the decomposed stage is shown in the x-axis.

shows the same tendency on ImageNet-1K. Other than that, FIGURE 13 and FIGURE 14 do not show specific regular patterns because other stages have the same number of MetaFormer blocks. Although each stage has different values of token number and channel dimension, they make up for each other in a dynamically balanced way — the smaller the token number is, the bigger the channel dimension is.

D. TRAIN-TTD-TRAIN

The Train-TTD-Train method exerts the learning potential of channel-mixing components and performs with higher accuracy than the Train-TTD method. On Cifar-10, our method can achieve higher accuracy with parameter reduction compared with the baseline. On ImageNet-1K, it is also a better trade-off than Train-TTD between size and accuracy. In detail, this method is implemented in MLP-Mixer and MetaFormer, and the results are compared with the Train-TTD (with fine-tuning) method. We enlarged the TTD scope by decomposing the whole channel-mixing layers, token-mixing layers, or both channel-mixing and token-mixing layers. For the convenience of description, we still call the token-TT model or channel-TT model to indicate models whose token-mixing layers or channel-mixing layers are decomposed. In addition, if decomposing a model's both channel-mixing and token-mixing layers, we call it a token&channel-TT model.

For MLP-Mixer-6 on Cifar-10, we implemented Train-TTD (with fine-tuning, 100 epochs) method for comparison. All the token-TT models perform better than the channel-TT models and token&channel-TT models. The first sub-figure in FIGURE 15 shows that the accuracy gap becomes smaller with the increment of ϵ. More specifically, the accuracy gaps between the token-TT models and channel-TT are 12.73%, 12.3%, 12.25%, 5.79% and 2.55% when $\epsilon = 1.5, 1, 25, 1.00,$

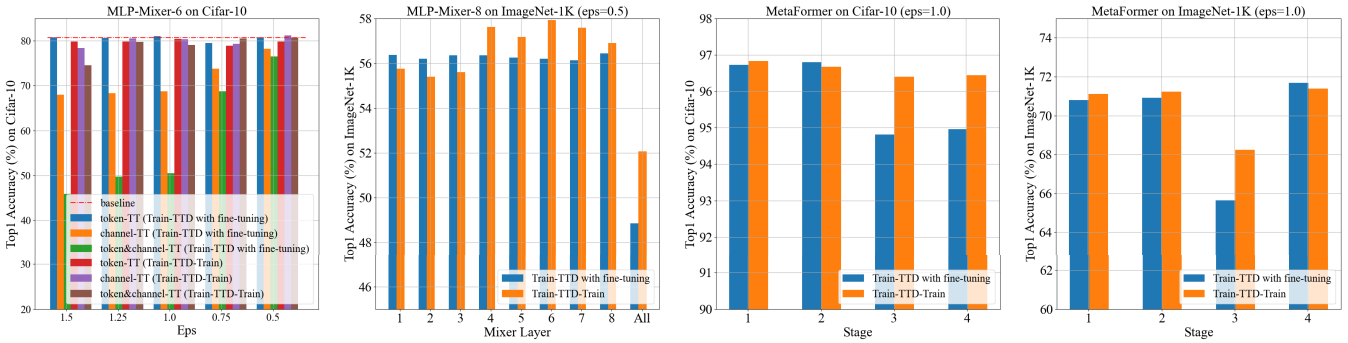


FIGURE 15. MLP-Mixer and MetaFormer Models on Cifar-10 and ImageNet-1K: The first sub-figure is about implementing TTD methods with different ϵ on Cifar-10. The second sub-figure is about implementing TTD methods with $\epsilon = 0.5$ on ImageNet-1K. The channel-mixing layers are decomposed in a mixer-by-mixer manner, and "All" means that all channel-mixing layers are decomposed. The third and fourth sub-figures are about decomposing MetaFormer in a stage-by-stage manner on Cifar-10 and ImageNet-1K, respectively.

0.75 and 0.50, respectively. As for the accuracy gaps between the token-TT models and token&channel-TT models, they are 34.83%, 30.87%, 30.53%, 10.82% and 4.27% when $\epsilon = 1.5$, 1, 25, 1.00, 0.75 and 0.50, respectively.

The phenomenon is different when the MLP-Mixer is implemented by the Train-TTD-Train (900 epochs) method. On the one hand, the accuracy of both channel-TT models and token&channel-TT models is improved much compared with the Train-TTD-Train method; on the other hand, some cases show that the channel-TT models perform with better accuracy than token-TT models. In detail, when $\epsilon = 1.5$, the accuracy gaps still exist. However, the gaps are much smaller than the Train-TTD (with fine-tuning) when $\epsilon = 1.25$ and 1.00. If reducing the ϵ, a new trend is emerging: both the channel-TT models and the token&channel-TT models perform with better accuracy than the token-TT models. In detail, the channel-TT models achieve higher accuracy than the token-TT models with 1.36% and 0.45% when $\epsilon = 0.5$ and 0.75, respectively. The token&channel-TT models have higher accuracy than the token-TT models with 0.79% and 1.6% when $\epsilon = 0.5$ and 0.75, respectively. In addition, there are TT models whose accuracy is higher than the baseline with smaller model sizes. These TT models include token-TT models, channel-TT models, and token&channel-TT models. Among them, a channel-TT model achieves the highest top-1 accuracy of 81.25% with $0.219M$ parameters. These results show that on a small dataset like Cifar-10, channel-mixing components have better learning capability to achieve higher accuracy than the baseline model, especially in the Train-TTD-Train method.

Since decomposing channel-mixing components play a better effect than decomposing other components on Cifar-10, we continued to decompose and test them on ImageNet-1K. For MLP-Mixer-8 on ImageNet-1K, we implemented Train-TTD-Train and Train-TTD (with fine-tuning, 15 epochs) methods toward channel-mixing layers in a mixer-by-mixer manner. The second sub-figure in FIGURE 15 shows that the Train-TTD-Train method achieves higher accuracy than the Train-TTD method in most cases. The accuracy gaps between the Train-TTD-Train method and the Train-TTD method are −0.61%, −0.81%, −0.76%, 1.26%, 0.91%, 1.73%, 1.45%, 0.47%, and 3.21% when decomposition targets are in mixer layer1, mixer layer2, mixer layer3, mixer layer4, mixer layer5, mixer layer6, mixer layer7, mixer layer8 and all mixer layers, respectively. Among TTD models, TT-Mixer-Layer-6 with the Train-TTD-Train method achieves 57.93% top-1 accuracy as the highest. Although there is a 0.42% accuracy drop compared with the baseline model on ImageNet-1K, Train-TTD-Train still performs better accuracy than Train-TTD.

MetaFormer achieves better accuracy in the Train-TTD-Train method than the Train-TTD (with fine-tuning) method both on Cifar-10 and ImageNet-1K. For experiments on Cifar-10, we implemented two decomposition methods in a stage-by-stage manner, including the Train-TTD-Train (1600 epochs) method and the Train-TTD (with fine-tuning, 100 epochs) method. The third sub-figure in FIGURE 15 shows decomposition results when $\epsilon = 1.0$. The Train-TTD-Train method performs better when decomposing the 1^{st}, 3^{rd} and 4^{th} stages, with 0.10%, 1.59%, and 1.48% higher accuracy, respectively. As for the Train-TTD (with fine-tuning) method, it only achieves 0.13% higher than the proposed method when decomposing the 2^{nd} stage. The change of accuracy with the Train-TTD (with fine-tuning) method is a little sharp, and the stage-3 model drops much accuracy. The experiments on ImageNet-1K have a similar tendency. From the fourth sub-figure in FIGURE 15, we can see the Train-TTD-Train method performs better than the Train-TTD (with fine-tuning, 15 epochs) method when decomposing the 1^{st}, 2^{nd} and 3^{rd} stages, with 0.31%, 0.31%, and 2.61% higher accuracy, respectively. As for the Train-TTD (with fine-tuning) method, it only achieves 0.3% higher than the proposed method when decomposing the 4^{th} stage. When comparing TTD models with the baseline model, they have very close accuracy on Cifar-10 but can not achieve higher accuracy on ImageNet-1K. Among TTD methods, Train-TTD-Train shows a better trade-off between accuracy and size than Train-TTD.

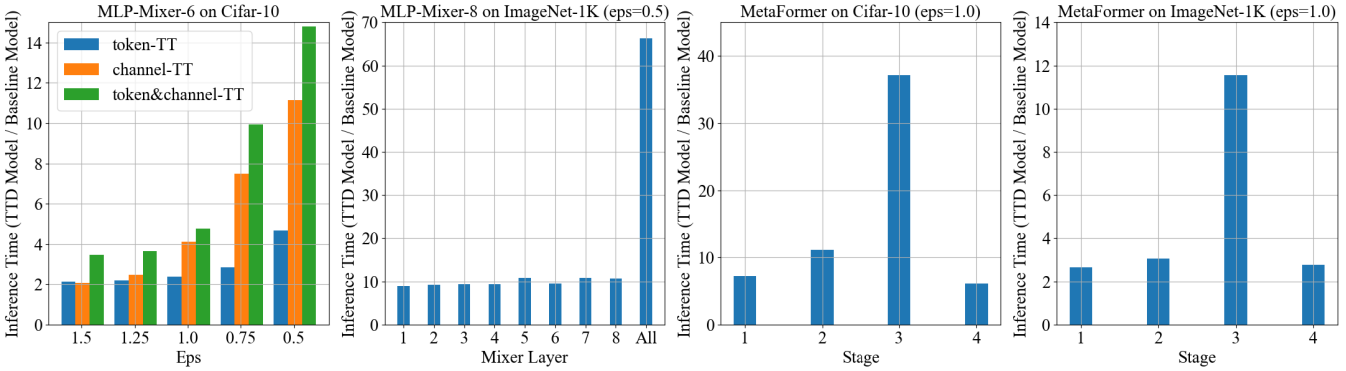


FIGURE 16. Inference Time Evaluation of Train-TTD-Train Method towards MLP-Mixer and MetaFormer: The two sub-figures on the left are about MLP-Mixer models on Cifar-10 and ImageNet-1K. The two sub-figures on the right are about MetaFormer models on Cifar-10 and ImageNet-1K. All subfigures' x-axes are the same as the ones in FIGURE 15, and their y-axes are all about the ratio of the TTD model's inference time to the baseline model's inference time.

For inference, TTD models take more time than the baseline. We evaluated models optimized by the Train-TTD-Train method on a TESLA V100 GPU and calculated the ratio of the TTD model's inference time to the baseline model's inference time. The first sub-figure in FIGURE 16 shows the TTD model takes 2.0–3.4×, 2.1–3.6×, 2.3–4.7×, 2.8–9.9×, and 4.6–14.7× more than the baseline when $\epsilon =$ 1.5, 1.25, 1.0, 0.75, and 0.5, respectively. We can find if a decomposition target is fixed, there is a trade-off between compression ratio and inference time. As the ϵ increases, we will get smaller TT-rank values, which leads to smaller TT cores. Thus, the inference time of a TT model becomes shorter. For MLP-Mixer-8 on ImageNet-1K, the second sub-figure shows TTD models take 8.9–10.7× more than the baseline if one mixer's channel layers are decomposed. For the case which decomposes all token-mixing layers, it takes 66.3× than the baseline. For MetaFormer models on Cifar-10, it takes 7.2×, 11.1×, 37.1×, and 6.8× more than the baseline when the 1^{st}, 2^{nd}, 3^{rd} and 4^{th} stages are decomposed, respectively. For its inference time on ImageNet-1K, it takes 2.6×, 3.0×, 11.5×, and 2.7× more than the baseline if the 1^{st}, 2^{nd}, 3^{rd} and 4^{th} stages are decomposed, respectively. The reason is that TT-format inference contains a high computational cost. After being implemented with a TTD method, the value of MAC operation changes from MN to $MN\sum_1^{d-1} r_i r_{i+1}$, as shown in FIGURE 3. That is, the TTD method essentially transforms parameter-intensive network models into compute-intensive models. If a deep learning accelerator runs a big model with limited on-chip memory resources, TTD is a good choice to help it avoid a large size of memory footprint. However, it may be necessary to design a domain-specific accelerator for handling the increased computation cost and improving energy efficiency, by taking advantage of reduced external memory accesses, such as [38], [39].

E. TTD GENERAL RULES

With a fixed number of TT-cores, increasing TT-ranks will be helpful for accuracy improvement. TT has a huge optimization space, including the shape and number of TT-cores. This subsection explores how different TT-ranks affect the decomposition results with a fixed number of TT-cores.

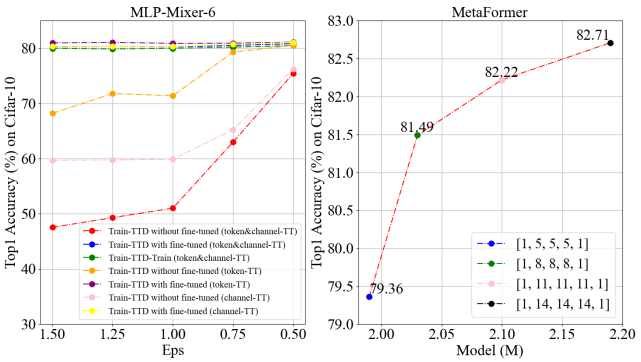


FIGURE 17. TTD experiments with a fixed number of TT-cores for MLP-Mixer and MetaFormer on Cifar-10: The left graph is about MLP-Mixer-6. Different TTD methods are implemented for token and channel-mixing components. Its y-axis is about top-1 accuracy on Cifar-10, and the x-axis is about prescribed accuracy ϵ. The right graph is about MetaFormer. The TTD-Train method is applied to it. Its x-axis is about model size, and its y-axis is about the top-1 accuracy on Cifar-10.

TABLE 4. Average TT-ranks for TT-layers (the MLP-Mixer) in a mixer layer with different ϵ (the MLP-Mixer): the TT-ranks are represented as $[R_1, R_2, R_3, R_4]$. Here $R_1 = R_4 = 1$, and R_2, R_3 are shown in the table.

Prescribed Accuracy	Mixer Layer ($[R_2, R_3]$) 1	2	3	4	5	6
$\epsilon = 1.50$	[1, 1]	[1, 1]	[1, 1]	[1, 1]	[1, 1]	[1, 1]
$\epsilon = 1.25$	[5, 1]	[5, 1]	[6, 1]	[6, 1]	[5, 1]	[4, 1]
$\epsilon = 1.00$	[12, 1]	[13, 1]	[15, 1]	[14, 1]	[14, 1]	[11, 1]
$\epsilon = 0.75$	[21, 6]	[21, 6]	[25, 6]	[25, 6]	[24, 6]	[19, 6]
$\epsilon = 0.50$	[30, 8]	[31, 8]	[37, 9]	[36, 9]	[36, 9]	[29, 8]

For MLP-Mixer-6, the experiment sets three TT-cores and different prescribed accuracy as $\epsilon =$ 1.5, 1.25, 1.0, 0.75, and 0.50. These factors influence TT-ranks R_1, R_2, R_3 and R_4. The first TT-rank (R_1) and the final TT-rank (R_4) must be 1 to match the decomposition shape. The changes about R_2 and R_3 are shown in TABLE 4. They increase obviously as the ϵ becomes smaller. The average accuracy of the MLP-Mixer-6 is calculated when the model

is decomposed in a mixer-by-mixer manner with different ϵ in FIGURE 17. No matter decomposing which kind of mixing component, the top-1 accuracy increases obviously as the ϵ reduces in all TTD methods. Moreover, we also explored the TTD-Train method with balanced TT-ranks toward the MetaFormer. In the experiment, we used the original Cifar-10 input size, set four TT-cores, and fixed the TT-ranks in a balanced manner ($R_2 = R_3 = R_4$). There are four sets of TT-ranks to choose: [1, 5, 5, 5, 1], [1, 8, 8, 8, 1], [1, 11, 11, 11, 1], [1, 14, 14, 14, 1]. All the TT-FC layers in the four stages adopt the same set of TT-ranks. The results also show that increasing TT-ranks improves the model's accuracy.

V. CONCLUSION

This article is the first study to explore TTD toward deep MLPs. The study adopts different TTD methods to analyze TTD performance and reveals that deep MLPs' different components have different learning capabilities and robustness under decomposition. The proposed Train-TTD-Train method makes effective compression toward deep MLPs and exerts the learning capability of channel-mixing components. Compared with the baseline model on Cifar-10, the Train-TTD-Train optimizes a deep MLP model with a 0.56% higher accuracy and 15.44% memory reduction. On ImageNet-1K, Train-TTD-Train also shows a better trade-off than other TTD methods. Our future study considers speeding up the inference process by exploring domain-specific acceleration for tensor-train.

ACKNOWLEDGMENT

The authors are grateful to the ArtIC members of the Tokyo Institute of Technology for their generous help and advice on this study. This work was partially supported by JST CREST Grant Number JPMJCR18K3, Japan.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1–9.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

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

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16 × 16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–22.

[6] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 10012–10022.

[7] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 10347–10357.

[8] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy, "MLP-mixer: An all-MLP architecture for vision," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 24261–24272.

[9] M. S. Qureshi, A. Aljarbouh, M. Fayaz, M. Bilal, W. Khan, and J. Khan, "An efficient methodology for water supply pipeline risk index prediction for avoiding accidental losses," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 5, p. 385, 2020.

[10] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Dec. 1989.

[11] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Comput.*, vol. 22, no. 12, pp. 3207–3220, Dec. 2010.

[12] H. Touvron, P. Bojanowski, M. Caron, M. Cord, A. El-Nouby, E. Grave, G. Izacard, A. Joulin, G. Synnaeve, J. Verbeek, and H. Jégou, "ResMLP: Feedforward networks for image classification with data-efficient training," 2021, *arXiv:2105.03404*.

[13] H. Liu, Z. Dai, D. So, and Q. V. Le, "Pay attention to MLPs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 9204–9215.

[14] W. Yu, M. Luo, P. Zhou, C. Si, Y. Zhou, X. Wang, J. Feng, and S. Yan, "MetaFormer is actually what you need for vision," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 10819–10829.

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

[16] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–9.

[17] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–14.

[18] J. Qi and J. Tejedor, "Exploiting hybrid models of tensor-train networks for spoken command recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2022, pp. 3114–3118.

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

[20] D. Bigoni, A. P. Engsig-Karup, and Y. M. Marzouk, "Spectral tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 38, no. 4, pp. A2405–A2439, Jan. 2016.

[21] D. Savostyanov and I. Oseledets, "Fast adaptive interpolation of multi-dimensional arrays in tensor train format," in *Proc. Int. Workshop Multidimensional (nD) Syst.*, Sep. 2011, pp. 1–8.

[22] A. Novikov, P. Izmailov, V. Khrulkov, M. Figurnov, and I. Oseledets, "Tensor train decomposition on TensorFlow (T3F)," *J. Mach. Learn. Res.*, vol. 21, no. 30, pp. 1–7, 2020.

[23] D. Suess and M. Holzäpfel, "mpnum: A matrix product representation library for Python," *J. Open Source Softw.*, vol. 2, no. 20, p. 465, Dec. 2017, doi: 10.21105/joss.00465.

[24] M. Usvyatsov, R. Ballester-Ripoll, and K. Schindler, "tntorch: Tensor network learning with PyTorch," *J. Mach. Learn. Res.*, vol. 23, no. 208, pp. 1–6, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1197.html

[25] D. Lian, Z. Yu, X. Sun, and S. Gao, "AS-MLP: An axial shifted MLP architecture for vision," in *Proc. Int. Conf. Learn. Represent.*, 2022, pp. 1–19.

[26] J. Guo, Y. Tang, K. Han, X. Chen, H. Wu, C. Xu, C. Xu, and Y. Wang, "Hire-MLP: Vision MLP via hierarchical rearrangement," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 826–836.

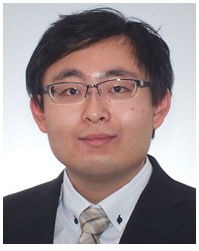
[27] S. Chen, E. Xie, C. Ge, D. Liang, and P. Luo, "CycleMLP: A MLP-like architecture for dense prediction," in *Proc. Int. Conf. Learn. Represent.*, 2022, pp. 1–21.

[28] Q. Hou, Z. Jiang, L. Yuan, M.-M. Cheng, S. Yan, and J. Feng, "Vision permutator: A permutable MLP-like architecture for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 1, pp. 1328–1334, Jan. 2023.

[29] N. Li, Y. Pan, Y. Chen, Z. Ding, D. Zhao, and Z. Xu, "Heuristic rank selection with progressively searching tensor ring network," *Complex Intell. Syst.*, vol. 8, no. 2, pp. 771–785, Apr. 2022.

[30] A. Krizhevsky, V. Nair, and G. Hinton. *CIFAR-10 (Canadian Institute for Advanced Research)*. Accessed: 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html
[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.
[32] G. Leclerc, A. Ilyas, L. Engstrom, S. M. Park, H. Salman, and A. Madry. (2022). *FFCV*. [Online]. Available: https://github.com/libffcv/ffcv/
[33] D. G. A. Smith and J. Gray, "Opt_einsum—A Python package for optimizing contraction order for einsum-like expressions," *J. Open Source Softw.*, vol. 3, no. 26, p. 753, Jun. 2018.
[34] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 702–703.
[35] H. Zhang, M. Cisse, Y. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk management," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–13.
[36] S. Yun, D. Han, S. Chun, S. J. Oh, Y. Yoo, and J. Choe, "CutMix: Regularization strategy to train strong classifiers with localizable features," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 6023–6032.
[37] J. Khan, M. Fayaz, A. Hussain, S. Khalid, W. K. Mashwani, and J. Gwak, "An improved alpha beta filter using a deep extreme learning machine," *IEEE Access*, vol. 9, pp. 61548–61564, 2021.
[38] R. Guo, Z. Yue, X. Si, T. Hu, H. Li, L. Tang, Y. Wang, L. Liu, M.-F. Chang, Q. Li, S. Wei, and S. Yin, "A 5.99-to-691.1 TOPS/W tensor-train in-memory-computing processor using bit-level-sparsity-based optimization and variable-precision quantization," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 242–244.
[39] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, and B. Yuan, "TIE: Energy-efficient tensor train-based inference engine for deep neural network," in *Proc. 46th Int. Symp. Comput. Archit.*, Jun. 2019, pp. 264–278.

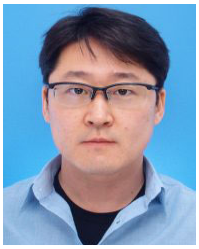
JIALE YAN received the B.E. degree in electronic science and technology from the Harbin Institute of Technology, Harbin, China, in 2016, and the M.S. degree in integrated circuit engineering from Tsinghua University, Beijing, China, in 2019. He is currently pursuing the Ph.D. degree with the Tokyo Institute of Technology, Japan. His research interests include deep learning and computer architecture design.



KOTA ANDO (Member, IEEE) received the B.E. degree in electronics and the M.S. degree in information technology from Hokkaido University, Sapporo, Japan, in 2016 and 2018, respectively, and the Ph.D. degree in engineering from the Tokyo Institute of Technology, Yokohama, Japan, in 2021.

He worked as an Assistant Professor at the Tokyo Institute of Technology, from 2021 to 2022. During his Ph.D. study, he was a JSPS Research Fellow, from 2018 to 2021. He is currently an Assistant Professor with Hokkaido University. His research interests include reconfigurable architectures, memory-centric processing, and hardware-aware algorithms for efficient deep learning processing.

Dr. Ando has been a member of IEICE, since 2016. He received the Best Student Presentation Award from the Technical Committee on Reconfigurable Systems of IEICE, Japan, in 2016 and 2017; the Best Student Poster Award from the Technical Committee on Integrated Circuits and Devices of IEICE, in 2018; and the Best Paper Award at the 2018 International Conference on Field-Programmable Technology.



JAEHOON YU (Member, IEEE) received the B.E. degree in electrical and electronic engineering and the M.S. (Informatics) degree in communications and computer engineering from Kyoto University, Kyoto, Japan, in 2005 and 2007, respectively, and the Ph.D. (Informatics) degree in information systems engineering from Osaka University, Osaka, Japan, in 2013. From 2013 to 2019, he was an Assistant Professor at Osaka University. He is currently an Associate Professor with the Tokyo Institute of Technology, Japan. His research interests include computer vision, machine learning, and system-level design. He is a member of IEICE and IPSJ.

MASATO MOTOMURA (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Kyoto University, Kyoto, Japan, in 1985, 1987, and 1996, respectively. In 1987, he joined the NEC Central Research Laboratories, where he worked on various hardware architectures including string search engines, multithreaded on-chip parallel processors, embedded DRAM field-programmable gate array (FPGA) hybrid systems, memory-based processors, and reconfigurable systems. From 2001 to 2008, he was at NEC Electronics, where he led a research and business development of dynamically reconfigurable processor (DRP) that he invented. He was also a Visiting Researcher at the MIT Laboratory for computer science, from 1991 to 1992. From 2011 to 2019, he was a Professor at Hokkaido University. He has been a Professor with the Tokyo Institute of Technology, Japan, since 2019. His current research interests include reconfigurable and parallel architectures for deep neural networks, machine learning, annealing machines, and intelligent computing in general. He is a member of IEICE, IPSJ, and EAJ. He was a recipient of the IEEE JSSC Annual Best Paper Award, in 1992; the IPSJ Annual Best Paper Award, in 1999; the IEICE Achievement Award, in 2011; the ISSCC Silkroad Award as the corresponding author, in 2018; the Ichimura Academic Award; and the Yamasaki Award, in 2022.