Highly Parallel Deep Ensemble Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper, we propose a novel highly parallel deep ensemble learning, which leads to highly compact and parallel deep neural networks. The main idea is to first represent the data in tensor form, apply a linear transform along certain dimension and split the transformed data into different independent spectral data sets; then the matrix product in conventional neural networks is replaced by tensor product, which in effect imposes certain transformed-induced structure on the original weight matrices, e.g., a block-circulant structure. The key feature of the proposed spectral tensor network is that it consists of parallel branches with each branch being an independent neural network trained using one spectral subset of the training data. Besides, the joint data/model parallel amiable for GPU implementation. The outputs of the parallel branches, which are trained on different independent spectral, are combined for ensemble learning to produce an overall network with substantially stronger generalization capability than that of those parallel branches. Moreover, benefiting from the reducing size of inputs, the proposed spectral tensor network exhibits an inherent network compression, and as a result, reduction in computation complexity, which leads to the acceleration of training process. The high parallelism from the massive independent operations of the parallel spectral subnetworks enable a further acceleration in training and inference process. We evaluate the proposed spectral tensor networks on the MNIST, CIFAR-10 and ImageNet data sets, to highlight that they simultaneously achieve network compression, reduction in computation and parallel speedup.

1 Introduction

Deep neural networks (DNNs) [1] have made impressive successes in many applications, such as computer vision [2][3][4], online game [5][6][7], natural language processing [8][9][10], autonomous driving [11][12][13], and robotics [14][15][16]. However, DNNs are memory-intensive and computation-intensive, which are two major challenges for wider adoption, e.g., in Internet of Things (IoT) applications [17]. Modern DNNs may have billions of parameters that consume excessive amount of memory and usually require long training time. For example, AlexNet [18] consists of three fully-connected layers and five convolutional layers, containing 60 million parameters and consuming about 250 MB of memory and about 40 hours for training on ImageNet data set [19].

To mitigate the impact of memory and computation intensive, there are two types of techniques most investigated by researchers, namely model compression and parallel training. The model compression methods, which represents the conventional neural layers in DNNs by compact layers, such as the strucred linear layers, tensor layers, *etc*., are one of the most efficient methods to reduce memory consumption of DNNs. Besides, the proposed parallel training methods can be generally categorized into data parallelism and model parallelism methods. It should be noted that tensor layer provides potential parallelism for efficient GPU computing as well as the reduction of the

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Table 1: Different tensor layers. An $N^2 \times N^2$ weight matrix is organized into a d-th order tensor with the size of each dimension n, i.e., $N^4 = n^d$, and r is the tensor rank.

Methods	Model size	Inference time
Fully connected layer	$O(N^4)$	$O(N^4)$
Low-rank matrix [29]	$O(N^2 r)$	$O(N^2 r)$
CP tensor layer [31]	$O(nr)$	$O(nr)$
Tucker tensor layer [32]	$O(nr + r^3)$	$O(nr)$
HT tensor layer [32]	$O(dn^2 r + r^d)$	$O(dnr^2 N^2)$
Tensor-train layer [29]	$O(dnr^2)$	$O(dr^3 N^2)$

memory consumption. Thus, it naturally motivates us to employ tensor layer to achieve both the model compression and parallel computing.

In this paper, we propose a unified approach that simultaneously achieves both model compression and parallel learning *without* communication overhead. The key technique is a novel spectral tensor layer that enables a joint *data/model-parallel* implementation of a DNN as follows: 1) The training data set is split into multiple orthogonal spectral sets; 2) The neural network is split into parallel branches with each branch being a conventional neural network, that are trained asynchronously and independently on the corresponding spectral sets; 3) The outputs of the parallel branches are finally combined to yield an overall neural network with substantially stronger generalization capability than that of those parallel branches.

The remainder of this paper is organized as follows. Section 2 presents an brief overview of the study on the model compression and parallel computing. Section 3 presents multiple spectral tensor layers, including fully connected layers, convolution layers and recurrent layers. Section 4 presents the experimental results and we conclude this paper in Section 5.

2 Related Works

Network compression: Consider a linear layer that is a central building block of modern DNNs, where an input $\boldsymbol{x} \in \mathbb{R}^{N^2}$ (e.g., an $N \times N$ image) is transformed by a weight matrix $\boldsymbol{W} \in \mathbb{R}^{N^2 \times N^2}$ [1], i.e., $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} \in \mathbb{R}^{N^2}$. The memory size for storing W is $O(N^4)$ and the computational complexity of the matrix-vector product is also $O(N^4)$, both are very demanding for smartphones, robots, and embedded devices.

Many works [20][21][22][23] showed that over 95% of the parameters are redundant, thus the network can be greatly compressed. The *structured linear layer* imposes certain structures on $\boldsymbol{W}$, including circulant [21][22][24], circulant-block [25][26], and Teoplitz-like structures [21][27][23]. For example, for a circulant weight matrix $\boldsymbol{W}$ [21][22][27][24], the memory size becomes $O(N^2)$ and the computational complexity becomes $O(N^2 \log N)$, at the expense of a slight drop in the inference performance.

On the other hand, the *tensor layer* exploits different types of low-rank tensor representations of the weight matrix $\boldsymbol{W}$ [28][29][30][31][32]. For example, if the weight matrix $\boldsymbol{W} \in \mathbb{R}^{N^2 \times N^2}$ has rank $r \ll N^2$, such that $\boldsymbol{W} = \boldsymbol{A}\boldsymbol{B}$, $\boldsymbol{A} \in \mathbb{R}^{N^2 \times r}$, $\boldsymbol{B} \in \mathbb{R}^{r \times N^2}$, then the memory size becomes $O(2N^2 r)$, and the computational complexity becomes $O(2N^2 r)$. Table 1 summarizes the model size and computational complexity for various tensor representation schemes.

Parallel training: Existing works on parallel machine learning take advantage of either data-parallelism [33][34][35] or model-parallelism [36][37][38] or both [39]. In the data-parallel approach, the data are distributed among multiple processors that apply the same model [33]. The processors periodically exchange the outputs and gradients. In the model-parallel approach, exact copies of the whole data set are processed by multiple processors that operate in parallel on different parts of the same model [36]. There are frequent aggregation of model parts and distribution of gradients [40].

From the above discussion, we see that the computation associated with a DNN can be speed up via two separate ways: one is through model compression so that reduced number of weight parameters leads to reduced number of multiplications and additions; and the other is through parallel computing,

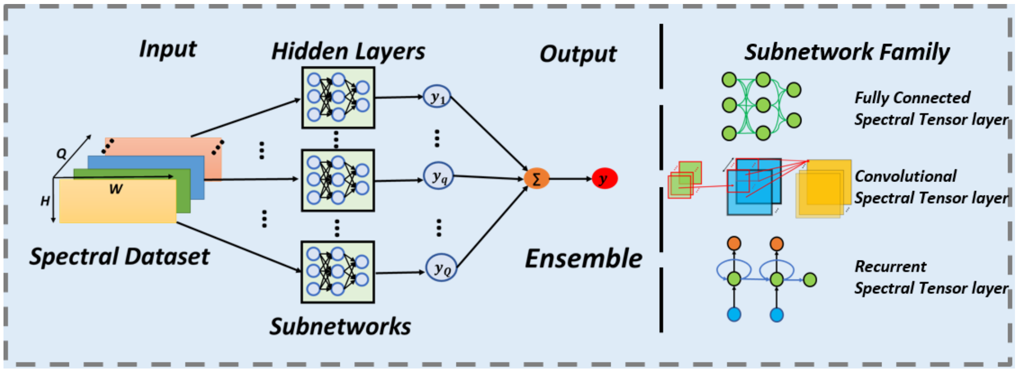

Figure 1: The framework of highly parallel deep ensemble learning.

where the speedup is due to distributing the computation among parallel processors, at the expense of communication overhead.

3 Highly Parallel Spectral Tensor Networks

3.1 Overview

As shown in Fig.1, a batch of data is firstly pre-processed to the spectral dataset and splitted into Q independent subsets. Then, Q different subnetwork, namely spectral tensor layer, works on Q independent joint spectral dataset to output own result. Last, the final result is the ensemble result from Q subnetworks. Tne subnetwork family includes the fully connected spectral tensor layer, convolutional spectral tensor layer and recurrent spectral tensor layer. The feature of joint data and model parallel makes it suitable for computing on GPUs. The operations in the forward pass, such as t-product further provides potential high parallelism for acceleration on GPUs.

3.2 Notations and Basic Tensor Operations

Scalars, vectors, matrices and tensors are denoted by lowercase, boldface lowercase, boldface capital, and calligraphic letters, e.g., $a \in \mathbb{R}$, $\boldsymbol{a} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{n_1 \times n_2}$, $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, respectively. We use $\mathcal{A}(:,:,k)$, $\mathcal{A}(:,j,:)$, $\mathcal{A}(i,:,:)$ to denote the frontal, lateral, and horizontal slices.

Given an invertible discrete linear transform $\mathcal{L}:\ \mathbb{R}^{n_3} \to \mathbb{C}^{n_3}$, let $\mathcal{L}$ and its inverse $\mathcal{L}^{-1}$ be taken along the third-dimension of third-order tensors. That is, for $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, $\widetilde{\mathcal{A}} = \mathcal{L}(\mathcal{A}) \in \mathbb{C}^{n_1 \times n_2 \times n_3}$, with $\widetilde{\mathcal{A}}(i,j,:) = \mathcal{L}(\mathcal{A}(i,j,:)), i = 1, ..., n_1, j = 1, ..., n_2$. And for $\widetilde{\mathcal{A}} \in \mathbb{C}^{n_1 \times n_2 \times n_3}$, $\mathcal{A} = \mathcal{L}^{-1}(\widetilde{\mathcal{A}})$, with $\mathcal{A}(i,j,:) = \mathcal{L}^{-1}(\widetilde{\mathcal{A}}(i,j,:))$, $i = 1, ..., n_1, j = 1, ..., n_2$. The general spectral tensor product [41][42][43] is defined as

$$\mathcal{C} = \mathcal{A} \bullet \mathcal{B} = \mathcal{L}^{-1}(\ \mathcal{L}(\mathcal{A}) \bigtriangleup \mathcal{L}(\mathcal{B})\ ), \tag{1}$$

where $\bigtriangleup$ denotes the frontal-slice wise multiplication, i.e., for $\widetilde{\mathcal{A}} \in \mathbb{C}^{n_1 \times n' \times n_3}, \widetilde{\mathcal{B}} \in \mathbb{C}^{n' \times n_2 \times n_3}$, if $\widetilde{\mathcal{C}} = \widetilde{\mathcal{A}} \bigtriangleup \widetilde{\mathcal{B}}$, then $\widetilde{\mathcal{C}}(:,:,k) = \widetilde{\mathcal{A}}(:,:,k)\, \widetilde{\mathcal{B}}(:,:,k)$, $k = 1, ..., n_3$. The t-product [44] is a special case of (1) where the transform $\mathcal{L}$ is the DFT [45].

For a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, we define the following operations. $\text{bcirc}(\mathcal{A}) \in \mathbb{R}^{n_1 n_3 \times n_2 n_3}$ organizes its n_3 frontal slices into a *block-circulant* matrix

$$\text{bcirc}(\mathcal{A}) = \begin{bmatrix} \mathcal{A}(:,:,1) & \mathcal{A}(:,:,n_3) & \cdots & \mathcal{A}(:,:,2) \\ \mathcal{A}(:,:,2) & \mathcal{A}(:,:,1) & \cdots & \mathcal{A}(:,:,3) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{A}(:,:,n_3) & \mathcal{A}(:,:,n_3-1) & \cdots & \mathcal{A}(:,:,1) \end{bmatrix}. \tag{2}$$

Further, $\text{unfold}(\cdot)$ is defined as

$$\text{unfold}(\mathcal{A}) = [\mathcal{A}(:,:,1)^T, \cdots, \mathcal{A}(:,:,n_3)^T]^T \in \mathbb{R}^{n_1 n_3 \times n_2},$$

and $\mathrm{fold}(\cdot)$ organizes it back to $\mathcal{A}$, such that

$$\mathrm{fold}(\mathrm{unfold}(\mathcal{A})) = \mathcal{A}. \tag{3}$$

Given $\mathcal{A} \in \mathbb{R}^{n_1 \times n' \times n_3}$ and $\mathcal{B} \in \mathbb{R}^{n' \times n_2 \times n_3}$, the t-product [45] can be expressed as follows

$$\mathcal{A} *_t \mathcal{B} = \mathrm{fold}(\mathrm{bcirc}(\mathcal{A}) \cdot \mathrm{unfold}(\mathcal{B})) \in \mathbb{R}^{n_1 \times n_2 \times n_3}. \tag{4}$$

The $\mathrm{vec}(\cdot)$ operation maps a matrix in $\mathbb{R}^{n_1 \times n_2}$ into a vector in $\mathbb{R}^{n_1 n_2}$, while $\mathrm{vec}^{-1}(\cdot)$ is the inverse mapping.

3.3 Fully Connected Spectral Tensor Layer

A conventional N-layer fully connected network takes m input vectors each of size $\ell_0' \times 1$ and represents them as a matrix $\boldsymbol{X}^0 \in \mathbb{R}^{\ell_0' \times m}$. For example, if each input vector represents a color image of size $n \times n \times 3$, then $\ell_0' = 3n^2$. Each input vector will be classified to one of the L classes. The network parameters at the j-th layer include a weight matrix $\boldsymbol{W}^j \in \mathbb{R}^{\ell_j' \times \ell_{j-1}'}$ and an offset $\boldsymbol{B}^j = \underbrace{[\boldsymbol{b}^j, ..., \boldsymbol{b}^j]}_{m} \in \mathbb{R}^{\ell_j' \times m}$, and the forward pass can be represented as [1]

$$\boldsymbol{X}^j = \sigma(\boldsymbol{W}^j \cdot \boldsymbol{X}^{j-1} + \boldsymbol{B}^j), \;\; j = 1, ..., N-1, \tag{5}$$

where $\boldsymbol{X}^j \in \mathbb{R}^{\ell_j' \times m}$, and $\sigma(\cdot)$ is an element-wise activation function, e.g., linear, sigmoid, ReLU, and softmax. The last, i.e., N-th, layer produces the output $\boldsymbol{Y} \in \mathbb{R}^{L \times m}$ corresponding to the m input vectors, where

$$\boldsymbol{X}^N = \boldsymbol{W}^N \cdot \boldsymbol{X}^{N-1}, \tag{6}$$

$$\boldsymbol{Y} = f(\boldsymbol{X}^N), \tag{7}$$

and the output function $f(\boldsymbol{X})$ operates on the columns of $\boldsymbol{X}$, i.e., $f(\boldsymbol{X}(:, s))$ maps $\boldsymbol{X}(:, s)$ to an output score vector $\boldsymbol{Y}(:, s) \in \mathbb{R}^L$ representing the probabilities that the s-th input data vector $\boldsymbol{X}^0(:, s)$ belongs to different classes. For example $f(\cdot)$ can be a softmax.

For our proposed fully connected spectral tensor network, the m input data vectors are organized as a tensor $\mathcal{X}^0 \in \mathbb{R}^{\ell_0 \times m \times Q}$. For the $n \times n \times 3$ color image example, we can set $Q = 3n$ and $\ell_0 = n$, namely each image is a lateral slice of $\mathcal{X}^0$. Using the weight tensor $\mathcal{W}^j \in \mathbb{R}^{\ell_j \times \ell_{j-1} \times Q}$ and offset tensor $\mathcal{B}^j \in \mathbb{R}^{\ell_j \times m \times Q}$, a fully connected tensor layer corresponding to (5) and (6) becomes

$$\mathcal{X}^j = \varrho(\mathcal{W}^j \bullet \mathcal{X}^{j-1} + \mathcal{B}^j), \; j = 1, ..., N-1, \tag{8}$$

$$\mathcal{X}^N = \mathcal{W}^N \bullet \mathcal{X}^{N-1}, \tag{9}$$

where $\mathcal{X}^j \in \mathbb{R}^{\ell_j \times m \times Q}$, the spectral tensor product $\bullet$ is given in (1), and the tensor-activation function $\varrho(\cdot)$ under transform $\mathcal{L}$ is defined by applying the conventional element-wise activation function $\sigma(\cdot)$ in the spectral domain, i.e.,

$$\varrho(\mathcal{X}) = \mathcal{L}^{-1}(\sigma(\mathcal{L}(\mathcal{X}))). \tag{10}$$

A salient feature of the proposed spectral tensor network is the *fully parallel implementation*. Specifically, for $\mathcal{X} \in \mathbb{R}^{\ell \times m \times Q}$, denote $\widetilde{\mathcal{X}} \in \mathbb{C}^{\ell \times m \times Q}$ as the transform of $\mathcal{X}$ along the third dimension, i.e., $\widetilde{\mathcal{X}}(i, s, :) = \mathcal{L}(\mathcal{X}(i, s, :))$, $i = 1, ..., \ell$, $s = 1, ..., m$. Denote further $\widetilde{\boldsymbol{X}}_q = \widetilde{\mathcal{X}}(:, :, q)$. Then according to (1), (8)-(9) can be split into Q branches of matrix computations

$$\widetilde{\boldsymbol{X}}_q^j = \sigma\left(\widetilde{\boldsymbol{W}}_q^j \cdot \widetilde{\boldsymbol{X}}_q^{j-1} + \widetilde{\boldsymbol{B}}_q^j\right), \tag{11}$$

$$\widetilde{\boldsymbol{X}}_q^N = \widetilde{\boldsymbol{W}}_q^N \cdot \widetilde{\boldsymbol{X}}_q^{N-1}, \; q = 1, ..., Q, \tag{12}$$

where $\widetilde{\boldsymbol{X}}_q^j \in \mathbb{C}^{\ell_j \times m}$, $\widetilde{\boldsymbol{B}}_q^j = \underbrace{[\widetilde{\boldsymbol{b}}_q^j, ..., \widetilde{\boldsymbol{b}}_q^j]}_{m} \in \mathbb{C}^{\ell_j \times m}$, and $\widetilde{\boldsymbol{W}}_q^j \in \mathbb{C}^{\ell_j \times \ell_{j-1}}$.

We further assume that the weight tensor $\mathcal{W}^j$ in (8)-(9) has low tubal-rank [45][46] such that $\mathcal{W}^j = \mathcal{C}^j \bullet \mathcal{D}^j$, where $\mathcal{C}^j \in \mathbb{C}^{\ell_j \times r \times Q}$, $\mathcal{D}^j \in \mathbb{C}^{r \times \ell_{j-1} \times Q}$, and $r \ll \min\{\ell_0, ..., \ell_N\}$. Correspondingly, the weight matrix of each branch has low-rank structure, i.e.,

$$\widetilde{\boldsymbol{W}}_q^j = \widetilde{\boldsymbol{C}}_q^j \cdot \widetilde{\boldsymbol{D}}_q^j, \; q = 1, ..., Q, \tag{13}$$

where $\widetilde{\boldsymbol{C}}_q^j \in \mathbb{C}^{\ell_j \times r}$ and $\widetilde{\boldsymbol{D}}_q^j \in \mathbb{C}^{r \times \ell_{j-1}}$. Then, (11)-(12) become

$$\widetilde{\boldsymbol{Z}}_q^j = \widetilde{\boldsymbol{D}}_q^j \cdot \widetilde{\boldsymbol{X}}_q^{j-1},\ j = 1, ..., N, \tag{14}$$

$$\widetilde{\boldsymbol{X}}_q^j = \sigma\left(\widetilde{\boldsymbol{C}}_q^j \cdot \widetilde{\boldsymbol{Z}}_q^j + \widetilde{\boldsymbol{B}}_q^j\right),\ j = 1, ..., N-1, \tag{15}$$

$$\widetilde{\boldsymbol{X}}_q^N = \widetilde{\boldsymbol{C}}_q^N \cdot \widetilde{\boldsymbol{Z}}_q^N, \tag{16}$$

where $\widetilde{\boldsymbol{Z}}_q^j \in \mathbb{C}^{r \times m}$, $q = 1, ..., Q$.

Therefore, an N-layer fully connected spectral tensor network in (8)-(9) is split into a $2N$-layer network, such that each layer in (11) is now implemented by two sub-layers, namely a linear layer (14) and a nonlinear layer (15), while the N-th layer in (12) is implemented by two linear sub-layers, namely (14) and (16). There are multiple parallel matrix multiplications with the same size and along the same dimension in Q subnetworks. Therefore, we employ the batch matrix multiplication using GPUs to accelerate the computations.

Finally, we specify the network output. At each branch q, the output function $f(\cdot)$ is applied to the last layer output $\widetilde{\boldsymbol{X}}_q^N$, as in (7), i.e.,

$$\boldsymbol{Y}_q = f(\widetilde{\boldsymbol{X}}_q^N),\ q = 1, ..., Q. \tag{17}$$

Finally the network output is the weighted sum of the outputs of the Q branches, i.e.,

$$\boldsymbol{Y} = \sum_{q=1}^{Q} \omega_q \boldsymbol{Y}_q, \quad \text{s.t. } \omega_q \geq 0, \ \sum_{q=1}^{Q} \omega_q = 1. \tag{18}$$

The loss function can be a cross-entropy function as follows:

$$\text{Loss} = -\sum_{s=1}^{m} \sum_{c=1}^{L} \mathbb{1}(\boldsymbol{y}_s(c) = 1) \cdot \ln(\boldsymbol{Y}(c, s)). \tag{19}$$

Once the spectral tensor network in Fig. 1 is trained, for inference, given a new data sample $\boldsymbol{x} \in \mathbb{R}^{\ell_0 Q}$, we first matricize it into $\boldsymbol{X} \in \mathbb{R}^{\ell_0 \times Q}$ and then take transform along each row to obtain $\widetilde{\boldsymbol{X}}$. We input the q-th column of $\widetilde{\boldsymbol{X}}$, i.e., $\widetilde{\boldsymbol{X}}(:, q)$, to the q-th sub-network and obtain the output $\boldsymbol{y}_q$, $q = 1, ..., Q$. The final output is then $\boldsymbol{y} = \sum_{q=1}^{Q} \omega_q \boldsymbol{y}_q$.

3.4 Convolutional Spectral Tensor Layer

A convolutional neural network [1][47] takes m input images each of dimension $H_0 \times W_0 \times C_0$, i.e., each image has a size $H_0 \times W_0$ and the number of channels is C_0, and represents them as a fourth-order tensor $\mathsf{X}^0 \in \mathbb{R}^{H_0 \times W_0 \times C_0 \times m}$. The input to the j-th layer is $\mathsf{X}^{j-1} \in \mathbb{R}^{H_{j-1} \times W_{j-1} \times C_{j-1} \times m}$, which is processed by a convolutional kernel $\mathsf{W}^j \in \mathbb{R}^{H_j' \times W_j' \times C_{j-1} \times C_j}$ and an offset $\mathsf{B}^j \in \mathbb{R}^{H_j'' \times W_j'' \times C_j \times m}$, to yield $\mathsf{Y}^j \in \mathbb{R}^{H_j'' \times W_j'' \times C_j \times m}$, with $H_j'' = H_{j-1} - H_j' + 1$, $W_j'' = W_{j-1} - W_j' + 1$, where

$$\begin{aligned} &\mathsf{Y}^j(h, w, c, :) = \sum_{d=1}^{C_{j-1}} \sum_{\ell=0}^{W_j'-1} \sum_{i=0}^{H_j'-1} \mathsf{X}^{j-1}(h+i, w+\ell, d, :) \cdot \mathsf{W}^j(i, \ell, d, c) + \mathsf{B}^j(h, w, c, :), \\ &h = 1, ..., H_j'',\ w = 1, ..., W_j'',\ c = 1, ..., C_j. \end{aligned} \tag{20}$$

To introduce the convolutional spectral tensor network, we represent (20) as a matrix product form that is similar to (5)

$$\boldsymbol{Y}^j = \boldsymbol{W}^j \cdot \boldsymbol{X}^{j-1} + \boldsymbol{B}^j, \tag{21}$$

where $\boldsymbol{Y}^j \in \mathbb{R}^{H_j'' W_j'' C_j \times m}$, $\boldsymbol{W}^j \in \mathbb{R}^{H_j'' W_j'' C_j \times H_{j-1} W_{j-1} C_{j-1}}$, and $\boldsymbol{X}^{j-1} \in \mathbb{R}^{H_{j-1} W_{j-1} C_{j-1} \times m}$ are formed from $\mathsf{Y}^j, \mathsf{W}^j$ and X^{j-1}. In particular,

$$\begin{aligned} \boldsymbol{Y}^j(:, s) &= \text{vec}(\text{unfold}(\mathsf{Y}^j(:, :, :, s))), \\ \boldsymbol{X}^{j-1}(:, s) &= \text{vec}(\text{unfold}(\mathsf{X}^{j-1}(:, :, :, s))),\ s = 1, ..., m. \end{aligned} \tag{22}$$

Assume that $C_j = nB$ for $j = 0, ..., N$, then similar to (8)-(9), (21) leads to a convolutional tensor layer

$$\mathcal{Y}^j = \mathcal{W}^j \bullet \mathcal{X}^{j-1} + \mathcal{B}^j, \tag{23}$$

where $\mathcal{Y}^j \in \mathbb{R}^{H_j'' W_j'' n \times m \times B}$, $\mathcal{W}^j \in \mathbb{R}^{H_j'' W_j'' n \times H_{j-1} W_{j-1} n \times B}$, $\mathcal{X}^j \in \mathbb{R}^{H_{j-1} W_{j-1} n \times m \times B}$, and $\mathcal{B}^j \in \mathbb{R}^{H_j'' W_j'' n \times m \times B}$.

Consider the case when $\mathcal{L}$ is DFT, according to (4), (23) can be written as (21) where $\boldsymbol{Y}^j = \text{unfold}(\mathcal{Y}^j)$, $\boldsymbol{X}^{j-1} = \text{unfold}(\mathcal{X}^{j-1})$, $\boldsymbol{B}^j = \text{unfold}(\mathcal{B}^j)$, and $\boldsymbol{W}^j = \text{bcirc}(\mathcal{W}^j) \in \mathbb{R}^{H_j'' W_j'' nB \times H_{j-1} W_{j-1} nB}$ has a block-circulant structure, namely $B \times B$ blocks organized in a circulant form and each block has size $H_j'' W_j'' n \times H_{j-1} W_{j-1} n$. Recall that $\boldsymbol{W}^j$ in (21) is derived from the convolutional kernel $\mathsf{W}^j \in \mathbb{R}^{H_j' \times W_j' \times nB \times nB}$ in (20), following a linear mapping that is consistent with (22). Therefore, the block-circulant structure of $\boldsymbol{W}^j$ in (21) implies a block-circulant structure of each matrix $\mathsf{W}^j(i, \ell, :, :)$ in (20).

Similar to the fully connected case in Section 3.3, the proposed convolutional spectral tensor network also features a *fully parallel implementation*. Specifically, for $\mathcal{X} \in \mathbb{R}^{HWn \times m \times B}$, denote $\widetilde{\mathcal{X}} = \mathcal{L}(\mathcal{X}) \in \mathbb{C}^{HWn \times m \times B}$ as the transform of $\mathcal{X}$ along the third dimension. Denote $\widetilde{\boldsymbol{X}}_b = \widetilde{\mathcal{X}}(:, :, b)$. Then, (23) can be split into B parallel branches of matrix computations as follows

$$\widetilde{\boldsymbol{Y}}_b^j = \widetilde{\boldsymbol{W}}_b^j \cdot \widetilde{\boldsymbol{X}}_b^{j-1} + \widetilde{\boldsymbol{B}}_b^j, \; b = 1, ..., B, \tag{24}$$

where $\widetilde{\boldsymbol{Y}}_b^j \in \mathbb{C}^{H_j'' W_j'' n \times m}$, $\widetilde{\boldsymbol{W}}_b^j \in \mathbb{C}^{H_j'' W_j'' n \times H_{j-1} W_{j-1} n}$, and $\widetilde{\boldsymbol{X}}_b^{j-1} \in \mathbb{C}^{H_{j-1} W_{j-1} n \times m}$. To fully utilize the parallelism of B branches, we employ batch matrix multiplication on GPUs to accelerate (24).

We convert (24) back to the convolutional form in (20), using an inverse mapping of (22) as follows

$$\begin{aligned} \widetilde{\mathsf{Y}}_b^j(:, :, :, s) &= \text{fold}(\text{vec}^{-1}(\widetilde{\boldsymbol{Y}}_b^j(:, s))), \\ \widetilde{\mathsf{X}}_b^{j-1}(:, :, :, s) &= \text{fold}(\text{vec}^{-1}(\widetilde{\boldsymbol{X}}_b^{j-1}(:, s))), \; s = 1, ..., m. \end{aligned} \tag{25}$$

For the b-th branch in (24), the input is $\widetilde{\mathsf{X}}_b^{j-1} \in \mathbb{C}^{H_{j-1} \times W_{j-1} \times n \times m}$, the output feature map is $\widetilde{\mathsf{Y}}_b \in \mathbb{C}^{H_j'' \times W_j'' \times n \times m}$, and the kernel weight is $\widetilde{\mathsf{W}}_b^j \in \mathbb{C}^{H_j' \times W_j' \times n \times n}$. Then for $b = 1, ..., B$, (24) can be rewritten as

$$\begin{aligned} &\widetilde{\mathsf{Y}}_b^j(h, w, c, :) = \sum_{d=1}^{n} \sum_{\ell=0}^{W_j'-1} \sum_{i=0}^{H_j'-1} \widetilde{\mathsf{X}}_b^{j-1}(h+i, w+\ell, d, :) \cdot \widetilde{\mathsf{W}}_b^j(i, \ell, d, c) + \widetilde{\mathsf{B}}_b^j(h, w, c, :), \\ &h = 1, ..., H_j'', \; w = 1, ..., W_j'', \; c = 1, ..., n. \end{aligned} \tag{26}$$

Similar to the fully connected tensor networks in Section 3.3, we apply the activation function in the spectral domain as follows

$$\widetilde{\mathsf{Z}}_b^j = \sigma(\widetilde{\mathsf{Y}}_b^j) \in \mathbb{C}^{H_j'' \times W_j'' \times n \times m}. \tag{27}$$

Then, the pooling operation is performed at the j-th layer of each branch, resulting in the output $\widetilde{\mathsf{X}}_b^j \in \mathbb{C}^{H_j \times W_j \times n \times m}$.

At the last layer, the output function $f(\cdot)$ is applied to $\widetilde{\mathsf{X}}_b^N$ as in (22).

3.5 Recurrent Spectral Tensor Layer

To present the recurrent spectral tensor layer, We take the same case in Section 3.3. Different from the fully connected layer, the network parameters of the recurrent layer at the j-th layer include a weight matrix $\boldsymbol{W}_X^j \in \mathbb{R}^{\ell_j' \times \ell_{j-1}'}$, $\boldsymbol{W}_H^j \in \mathbb{R}^{\ell_{j-1}' \times \ell_{j-1}'}$, $\boldsymbol{W}_Y^j \in \mathbb{R}^{\ell_{j-1}' \times \ell_{j-1}'}$ and an offset $\boldsymbol{B}^j = \underbrace{[\boldsymbol{b}^j, ..., \boldsymbol{b}^j]}_{m} \in \mathbb{R}^{\ell_j' \times m}$, and the forward pass can be represented as [1]

$$\begin{aligned} \boldsymbol{H}_t^j &= \sigma_1(\boldsymbol{W}_X^j \cdot \boldsymbol{X}_t^{j-1} + \boldsymbol{W}_H^j \cdot \boldsymbol{H}_t^{j-1} + \boldsymbol{B}^j), \\ \boldsymbol{X}_t^j &= \sigma_2(\boldsymbol{W}_Y^j \cdot \boldsymbol{H}_t^j), \;\; j = 1, ..., N, t = 1, 2, ..., T, \end{aligned} \tag{28}$$

where $\boldsymbol{X}_t^j \in \mathbb{R}^{\ell'_j \times m}$ is the input matrix at the j-t time step, and $\boldsymbol{Y}_t^N = \boldsymbol{X}_t^N$.

For the introduced recurrent spectral tensor layer, the forward can be computed as

$$\begin{aligned} \mathcal{H}_t^j &= \varrho_1(\mathcal{W}_X^j \bullet \mathcal{X}_t^{j-1} + \mathcal{W}_H^j \bullet \mathcal{H}_t^{j-1} + \mathcal{B}^j), \\ \mathcal{X}_t^j &= \varrho_2(\mathcal{W}_Y^j \bullet \mathcal{H}_t^j), \;\; j = 1, ..., N, t = 1, 2, ..., T, \end{aligned} \tag{29}$$

where $\mathcal{X}_t^j \in \mathbb{R}^{\ell_j \times m \times Q}$.

Furthermore, the implementation of Q branches for computations can be written as

$$\begin{aligned} \widetilde{H}_{q,t}^j &= \varrho_1(\widetilde{W}_X^j \bullet \widetilde{X}_{q,t}^{j-1} + \widetilde{W}_H^j \bullet \widetilde{H}_{q,t}^{j-1} + \widetilde{B}^j), \\ \widetilde{X}_{q,t}^j &= \varrho_2(\widetilde{W}_Y^j \bullet \widetilde{H}_{q,t}^j), \;\; j = 1, ..., N, t = 1, 2, ..., T, \end{aligned} \tag{30}$$

where $\widetilde{\boldsymbol{X}}_{q,t}^j \in \mathbb{C}^{\ell_j \times m}$ is the input for the q-th subnetwork at the t-th time step, likewise for $\widetilde{\boldsymbol{H}}_{q,t}^j$. There are $2Q$ parallel matrix multiplications with the same size and along the same dimension, thus we use the batch matrix multiplication on GPUs to accelerate (30).

4 Performance Evaluation

We first describe the experimental settings, then present the results on the MNIST, CIFAR-10 and ImageNet data sets.

4.1 Data Sets and Performance Metrics

We verify the performance of the proposed spectral tensor networks on the following three widely used data sets: 1) MNIST [48] contains grayscale images of handwritten digits. Each image has 28×28 pixels. The training set has $60,000$ images and the testing set has $10,000$ images. 2) CIFAR-10 [49] contains $60,000$ color images in 10 classes, where each image has size $32 \times 32 \times 3$.3) ImageNet-1K [19]: It contains $12,000,000$ training images and $50,000$ testing images with size of $224 \times 224 \times 3$, labeled with the presence or absence of 1000 object categories that do not overlap with each other.

We are interested in the following performance metrics:1) *Compression ratio*: the ratio of the conventional network size to the spectral tensor network size, which is the also the total reduction in computation due to the reduced number of non-zero network weights; 2) Parallel speedup: the ratio of the training time of a conventional network to that of the spectral tensor network, due to the fully parallel training of all sub-networks; 3) *Convergence*: the loss value versus the training iterations: 4) *Accuracy*: the percentage of correctly estimated labels. Both the training and testing processes are executed on a DGX-2 server [50] that has two 64 core AMD CPUs, 8 NVIDIA A100 GPUs and 2 TB of memory. The operating system is Ubuntu 20.04 with CUDA 10.1. We use PyTorch [51] to implement neural networks.

We summarize the compression ratio, the reduction of computation, and the parallel speedup in Table 2. They are theoretical upper bounds, while their actual values depend on data sets and implementations. For the compression ratio and reduction in computation, each fully connected network / convolutional network has two columns: the right one corresponds to select-the-best weighting, and the left one to other weighting methods.

4.2 Evaluation of Fully Connected Spectral Tensor Networks

For comparison, we consider a conventional fully connected network (FC) [1], the tNN [44], and the fully connected spectral tensor network (FC-tensor) in Section 3. All three methods use the ReLU activation function as $\sigma(\cdot)$ in the hidden layers, the softmax function as the output function $f(\cdot)$ in the last layer, and the cross-entropy loss function in (19). We use $N = 8$ layers in each method and the DCT transform in tNN and the proposed FC-tensor method. The learning rate was set to be 0.01, the batch size was set to be 64, and we used the Adam optimizer [52].

For the MNIST data set, the conventional FC method has $n = 28$, $\ell'_0 = ... = \ell'_7 = 784$, and $L = 10$. Both the tNN method and our FC-tensor method have $n = 28, Q = 28$, $\ell_0 = ... = \ell_7 = 28$, and $L = 10$. Our FC-tensor method has $r = 8$. For the CIFAR-10 data set, the following parameters are

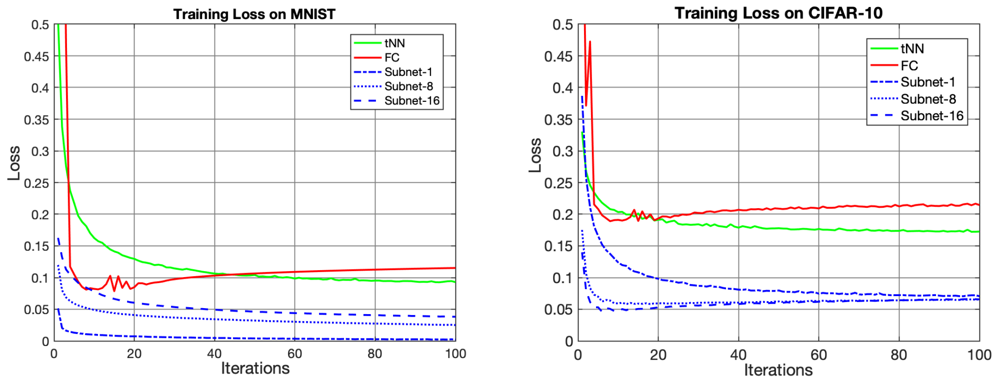

Figure 2: Training loss of fully connected networks on the MNIST data set (left) and CIFAR-10 data set (right).

Table 2: Upper bounds for model compression and parallel speedup. For fully connected networks, n denotes the input size of each sub-network, r is a rank value, and there are Q branches. For convolutional networks, there are B branches.

-	Fully Connected	Convolutional (1D)
Compression ratio	$O(nQ/2r), O(nQ^2/2r)$	$O(B), O(B^2)$
Reduction in computation	$O(nQ/2r), O(nQ^2/2r)$	$O(B), O(B^2)$
Parallel speedup	$O(Q)$	$O(B)$

different: $n = 32, Q = 32$, $\ell_0' = ... = \ell_7' = 1,024$, and $\ell_0 = ... = \ell_7 = 32$. Therefore, our methods achieve a compression ratio of $49\times$ and $64\times$ for the two data sets, respectively.

The training loss over iterations is shown in Fig. 2, with the left one for the MNIST datas set and the right one for the CIFAR-10 data set. Our scheme converges faster than tNN and FC, while the training process is more stable than FC. The possible reason is that the FC-tensor has much less parameters so that a more stable model can be learned from the same amount of data samples[1]. The loss values of our sub-networks are lower than both tNN and FC.

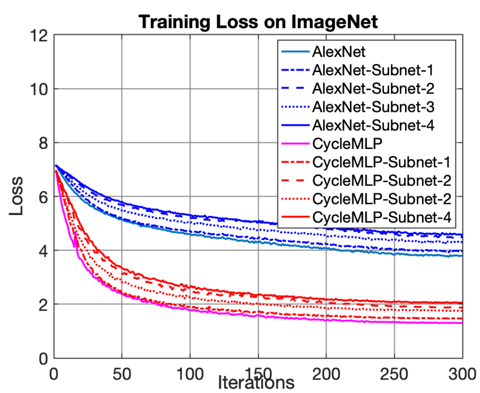

Figure 3: Training loss on ImageNet-1K data set.

Table 3: MNIST and CIAFR-10 data sets.

Methods	MNIST	CIFAR-10
FC [1]	98.71%	**59.19%**
tNN [44]	97.59%	44.50%
FC-tensor (**average**)	97.43%	47.24%
FC-tensor (**weighted sum**)	98.02%	48.13%
FC-tensor (**geometric**)	**99.01%**	48.33%

In Table 3, we report accuracy results on both MNIST and CIAFR-10 data sets. Among the four schemes for weighting the sub-networks , the geometric weighting gives the best performance. For the MNIST data set, all three methods achieve a relative high accuracy, i.e., over 97%, while our FC-tensor method reaches 98.36%. For the CIFAR-10 data set, all three methods achieve a relative low accuracy, i.e., below 60%. This is consistent with the known fact that fully connected layers are

[1] Note that we use the same number of layers and the same batch size.

Table 3: Results on the ImageNet-1K data set.

Methods	Accuracy	Size	Training Time
AlexNet [18]	$\mathbf{63.44}\%$	244 MB	40.8 h
AlexNet-spectral (**average**)	61.26%	130 MB	31.9 h
AlexNet-spectral (**weighted sum**)	58.01%	130 MB	31.9 h
AlexNet-spectral (**geometric**)	62.26%	130 MB	31.9 h
AlexNet-spectral (**select-the-best**)	56.45%	32.5 MB	31.9 h
CycleMLP [53]	$\mathbf{83.23}\%$	103 MB	93.6 h
CycleMLP-spectral (**average**)	78.80%	76 MB	60.4 h
CycleMLP-spectral (**weighted sum**)	77.54%	76 MB	60.4 h
CycleMLP-spectral (**geometric**)	83.20%	76 MB	60.4 h
CycleMLP-spectral (**select-the-best**)	72.45%	19 MB	60.4 h

not enough for the classification task on CIFAR-10. Note that both tNN and FC-tensor achieve lower accuracy than the FC method.

4.3 Evaluation on ImageNet Data set

The ImageNet data set [19] is split into $B = 4$ spectral subsets, where each image is organized into a tensor of size $56 \times 56 \times 3 \times 16$ and then processed into a spectrum tensor using DCT transform. Note that the three RGB channels are processed independently.

Our proposed spectral tensor methods have the same structure in Fig. 1, where each branch is replaced by either AlexNet [18] or CycleMLP. We use the DCT transform in our spectral methods. We set the learning rate 0.01 and the batch size 128. We follow the standard practice in the community by reporting the top-1 accuracy on the testing set.

For the ImageNet data set, the training loss over training iterations is shown in Fig. 3. Our spectral sub-networks have similar loss curve to their original networks. In Table 3, we report the accuracy, model size, and training time. For the AlextNet structure, our spectral network achieves $1.88\times$ model compression and $1.28\times$ speedup in training time, at the cost of an accuracy drop of 1.18%. For the CycleMLP structure, our spectral network achieves $1.36\times$ model compression and $1.55\times$ speedup in training time, at the cost of an accuracy drop of 0.03%.

5 Conclusions

In this paper, we have proposed a spectral tensor form of deep neural networks that is inherently compressive and allows communication-free parallel/distributed implementations. The data is organized into tensors and a linear transform is applied along certain dimension, resulting in different spectral subsets. The overall network consists of parallel branches of networks, each independently performs training and inference on a spectral data subset. We tested the proposed spectral networks, including fully connected, convolutional, AlexNet, and CycleMLP, on the MNIST, CIFAR-10 and ImageNet data sets, and results show that they can achieve relatively high accuracy with substantial network compression, computation reduction, and parallel speedup, compared with conventional networks. Limited by the pages, We do not provide experiments using the recurrent spectral tensor layer.

For future works, we would like to explore an ensemble-style approach *model soup* [54] that takes average over multiple trained models and achieves state-of-the-art performance on the ImageNet data set.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT Press, 2016.

[2] Yang Liu, Peng Sun, Nickolas Wergeles, and Yi Shang, "A survey and performance evaluation of deep learning methods for small object detection," *Expert Systems with Applications*, vol. 172, pp. 114602, 2021.

[3] Xiyang Dai, Yinpeng Chen, Bin Xiao, Dongdong Chen, Mengchen Liu, Lu Yuan, and Lei Zhang, "Dynamic head: Unifying object detection heads with attentions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7373–7382.

[4] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, et al., "Sparse r-cnn: End-to-end object detection with learnable proposals," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14454–14463.

[5] Xiangjun Wang, Junxiao Song, Penghui Qi, Peng Peng, Zhenkun Tang, Wei Zhang, Weimin Li, Xiongjun Pi, Jujie He, Chao Gao, et al., "Scc: an efficient deep reinforcement learning agent mastering the game of starcraft ii," in *International Conference on Machine Learning*. PMLR, 2021, pp. 10905–10915.

[6] Tianhao Zhang, Yueheng Li, Chen Wang, Guangming Xie, and Zongqing Lu, "Fop: Factorizing optimal joint policy of maximum-entropy multi-agent reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12491–12500.

[7] Bo Liu, Qiang Liu, Peter Stone, Animesh Garg, Yuke Zhu, and Anima Anandkumar, "Coach-player multi-agent reinforcement learning for dynamic team composition," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6860–6870.

[8] Zhiqi Huang, Fenglin Liu, Xian Wu, Shen Ge, Helin Wang, Wei Fan, and Yuexian Zou, "Audio-oriented multimodal machine comprehension via dynamic inter-and intra-modality attention," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021, vol. 35, pp. 13098–13106.

[9] Tianyang Zhao, Zhao Yan, Yunbo Cao, and Zhoujun Li, "Asking effective and diverse questions: a machine reading comprehension based framework for joint entity-relation extraction," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 3948–3954.

[10] Cong Sun, Zhihao Yang, Lei Wang, Yin Zhang, Hongfei Lin, and Jian Wang, "Biomedical named entity recognition using bert in the machine reading comprehension framework," *Journal of Biomedical Informatics*, vol. 118, pp. 103799, 2021.

[11] Aditya Prakash, Kashyap Chitta, and Andreas Geiger, "Multi-modal fusion transformer for end-to-end autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7077–7087.

[12] Yingfeng Cai, Tianyu Luan, Hongbo Gao, Hai Wang, Long Chen, Yicheng Li, Miguel Angel Sotelo, and Zhixiong Li, "Yolov4-5d: An effective and efficient object detector for autonomous driving," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–13, 2021.

[13] Sudeep Fadadu, Shreyash Pandey, Darshan Hegde, Yi Shi, Fang-Chieh Chou, Nemanja Djuric, and Carlos Vallespi-Gonzalez, "Multi-view fusion of sensor data for improved perception and prediction in autonomous driving," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 2349–2357.

[14] Huu-Thiet Nguyen, Chien Chern Cheah, and Kar-Ann Toh, "An analytic layer-wise deep learning framework with applications to robotics," *Automatica*, vol. 135, pp. 110007, 2022.

[15] Radouan Ait Mouha et al., "Deep learning for robotics," *Journal of Data Analysis and Information Processing*, vol. 9, no. 02, pp. 63, 2021.

[16] Yinong Chen and Gennaro De Luca, "Technologies supporting artificial intelligence and robotics application development," *Journal of Artificial Intelligence and Technology*, vol. 1, no. 1, pp. 1–8, 2021.

[17] G. Menghani, "Efficient deep learning: A survey on making deep learning models smaller, faster, and better," *arXiv preprint arXiv:2106.08962*, 2021.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, "ImageNet: A large-scale hierarchical image database," in *IEEE CVPR*. Ieee, 2009, pp. 248–255.

[20] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, 2013, pp. 2148–2156.

[21] V. Sindhwani, T. Sainath, and S. Kumar, "Structured transforms for small-footprint deep learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 3088–3096.

[22] Y. Cheng, F. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *IEEE International Conference on Computer Vision*, 2015, pp. 2857–2865.

[23] M. Moczulski, M. Denil, J. Appleyard, N. De Freitas, Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and S. Reed, "ACDC: A structured efficient linear layer," in *ICLR*, 2015, vol. 55.

[24] A. Prabhu, A. Farhadi, and M. Rastegari, "Butterfly transform: An efficient fft based neural architecture design," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12024–12033.

[25] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, and G. Yuan, "Circnn: accelerating and compressing deep neural networks using block-circulant weight matrices," in *Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 395–408.

[26] S. Liao and B. Yuan, "Circconv: A structured convolution with low complexity," in *AAAI Conference on Artificial Intelligence*, 2019, vol. 33, pp. 4287–4294.

[27] B. Gong, B. Jou, F. Yu, and S.-F. Chang, "Tamp: A library for compact deep neural networks with structured matrices," in *ACM International Conference on Multimedia*, 2016, pp. 1206–1209.

[28] T. N Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *IEEE ICASSP*, 2013, pp. 6655–6659.

[29] A. Novikov, D. Podoprikhin, A. Osokin, and D.P. Vetrov, "Tensorizing neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 442–450.

[30] W. Wang, Y. Sun, B. Eriksson, W. Wang, and V. Aggarwal, "Wide compression: Tensor ring nets," in *IEEE CVPR*, 2018, pp. 9329–9338.

[31] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *ICLR*, 2015.

[32] M. Yin, S. Liao, X.-Y. Liu, X. Wang, and B. Yuan, "Towards extremely compact rnns for video recognition with fully decomposed hierarchical tucker structure," in *IEEE CVPR*, 2021, pp. 12085–12094.

[33] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–33, 2020.

[34] Vipul Gupta, Dhruv Choudhary, Peter Tang, Xiaohan Wei, Xing Wang, Yuzhen Huang, Arun Kejariwal, Kannan Ramchandran, and Michael W Mahoney, "Training recommender systems at scale: Communication-efficient model and data parallelism," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2928–2936.

[35] Xiangyu Ye, Zhiquan Lai, Shengwei Li, Lei Cai, Ding Sun, Linbo Qiao, and Dongsheng Li, "Hippie: A data-paralleled pipeline approach to improve memory-efficiency and scalability for large dnn training," in *50th International Conference on Parallel Processing*, 2021, pp. 1–10.

[36] A. L. Gaunt, M. A. Johnson, M. Riechert, D. Tarlow, R. Tomioka, D. Vytiniotis, and S. Webster, "AMPNet: Asynchronous model-parallel training for dynamic neural networks," *arXiv preprint arXiv:1705.09786*, 2017.

[37] An Xu, Zhouyuan Huo, and Heng Huang, "On the acceleration of deep learning model parallelism with staleness," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2088–2097.

[38] Kabir Nagrecha, "Model-parallel model selection for deep learning systems," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2929–2931.

[39] E. P. Xing, Q. Ho, P. Xie, and D. Wei, "Strategies and principles of distributed machine learning on big data," *Engineering*, vol. 2, no. 2, pp. 179–195, 2016.

[40] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., "TensorFlow: A system for large-scale machine learning," in *USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 265–283.

[41] E. Kernfeld, M. Kilmer, and S. Aeron, "Tensor–tensor products with invertible linear transforms," *Linear Algebra and its Applications*, vol. 485, pp. 545–570, 2015.

[42] M. E. Kilmer, L. Horesh, H. Avron, and E. Newman, "Tensor-tensor algebra for optimal representation and compression of multiway data," *Proceedings of the National Academy of Sciences*, vol. 118, no. 28, 2021.

[43] X.-Y. Liu and X. Wang, "Fourth-order tensors with multidimensional discrete transforms," *arXiv preprint arXiv:1705.01576*, pp. 1–37, 2017.

[44] E. Newman, L. Horesh, H. Avron, and M. Kilmer, "Stable tensor neural networks for rapid deep learning," *arXiv preprint arXiv:1811.06569*, 2018.

[45] M.E. Kilmer and C.D Martin, "Factorization strategies for third-order tensors," *Linear Algebra and its Applications*, vol. 435, no. 3, pp. 641–658, 2011.

[46] M.E. Kilmer, K. Braman, N. Hao, and R.C. Hoover, "Third-order tensors as operators on matrices: A theoretical and computational framework with applications in imaging," *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 1, pp. 148–172, 2013.

[47] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[48] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[49] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, University of Tront*, 2009.

[50] J. Choquette et al., "NVIDIA A100 tensor core GPU: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[51] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, and et al, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8026–8037.

[52] D.P Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.

[53] S. Chen, E. Xie, C. Ge, D. Liang, and P. Luo, "CycleMLP: A MLP-like architecture for dense prediction," *ICLR*, 2022.

[54] Mitchell Wortsman, Gabriel Ilharco, and et al, "Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time," *preprint arXiv:2203.05482*, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [No]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] **See Section 4.1.**
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] **See Section 4.2 and 4.3.**
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We do not supply experiments of recurrent spectral tensor layer.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] **See Section 4.1.**
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] **See Section 4.3.**
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] **All the consent is open-source.**
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] **We do not use any personally identifiable information or offensive content.**
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]