Wide Compression: Tensor Ring Nets

Wenqi Wang
Purdue University
wang2041@purdue.edu

Yifan Sun
Technicolor Research
ysun13@cs.ubc.ca

Brian Eriksson
Adobe
eriksson@adobe.com

Wenlin Wang
Duke University
wenlin.wang@duke.edu

Vaneet Aggarwal
Purdue University
vaneet@purdue.edu

Abstract

Deep neural networks have demonstrated state-of-the-art performance in a variety of real-world applications. In order to obtain performance gains, these networks have grown larger and deeper, containing millions or even billions of parameters and over a thousand layers. The trade-off is that these large architectures require an enormous amount of memory, storage, and computation, thus limiting their usability. Inspired by the recent tensor ring factorization, we introduce Tensor Ring Networks (TR-Nets), which significantly compress both the fully connected layers and the convolutional layers of deep neural networks. Our results show that our TR-Nets approach is able to compress LeNet-5 by $11\times$ *without losing accuracy, and can compress the state-of-the-art Wide ResNet by* $243\times$ *with only 2.3% degradation in Cifar10 image classification. Overall, this compression scheme shows promise in scientific computing and deep learning, especially for emerging resource-constrained devices such as smartphones, wearables, and IoT devices.*

1. Introduction

Deep neural networks have made significant improvements in a variety of applications, including recommender systems [45, 53], time series classification [49], nature language processing [16, 21, 50], and image and video recognition [51]. These accuracy improvements require developing deeper and deeper networks, evolving from AlexNet [33] (with $P = 61$ M parameters), VGG19 [41] ($P = 114$ M), and GoogleNet ($P = 11$ M) [43], to 32-layer ResNet ($P = 0.46$ M) [24, 25], 28-layer WideResNet [52] ($P = 36.5$ M), and DenseNets [27]. Unfortunately, with each evolution in architecture comes a significant increase in the number of model parameters.

On the other hand, many modern use cases of deep neural networks are for resource-constrained devices, such as mobile phones [28], wearables and IoT devices [34], etc. In these applications, storage, memory, and test runtime complexity are extremely limited in resources, and compression in these areas is thus essential.

After prior work [8] observed redundancy in trained neural networks, a useful area of research has been compression of network layer parameters (*e.g.,* [9, 23, 22, 18]). While a vast majority of this research has been focused on the compression of fully connected layer parameters, the latest deep learning architectures are almost entirely dominated by convolutional layers. For example, while only 5% of AlexNet parameters are from convolutional layers, over 99% of Wide ResNet parameters are from convolutional layers. This necessitates new techniques that can factorize and compress the multi-dimensional tensor parameters of convolutional layers.

We propose compressing deep neural networks using *Tensor Ring (TR) factorizations* [54], which can be viewed as a generalization of a single Canonical Polyadic (CP) decomposition [26, 30, 6], with two extensions:

1. the outer vector products are generalized to matrix products, and
2. the first and last matrix are additionally multiplied along their outer edges, forming a "ring" structure.

The exact formulation is described in more detail in Section 3. Note that this is also a generalization of the *Tensor Train factorization* [39], which only includes the first extension. This is inspired by previous results in image processing [47], which demonstrate that this general factorization technique is extremely expressive, especially in preserving spatial features.

Specifically, we introduce Tensor Ring Nets (TRN), in which layers of a deep neural network are compressed using tensor ring factorization. For fully connected layers, we compress the weight matrix, and investigate different merge/reshape orders to minimize real-time computation and memory needs. For convolutional layers, we carefully

compress the filter weights such that we do not distort the spatial properties of the mask. Since the mask dimensions are usually very small (5×5, 3×3 or even 1×1) we do not compress along these dimensions at all, and instead compress along the input and output channel dimensions.

To verify the expressive power of this formulation, we train several compressed networks. First, we train LeNet-300-100 and LeNet-5 [36] on the MNIST dataset, compressing LeNet-5 by $11\times$ without degradation and achiving 99.31% accuracy, and compressing LeNet-300-100 by $13\times$ with a degrading of only 0.14% (obtaining overall accuracy of 97.36%). Additionally, we examine the state-of-the-art 28-layer Wide-ResNet [52] on Cifar10, and find that TRN can be used to effectively compress the Wide-ResNet by $243\times$ with only 2.3% decay in performance, obtaining 92.7% accuracy. The compression results demonstrates the capability of TRN to compress state-of-the-art deep learning models for new resources constrained applications.

Section 2 discusses related work in neural network compression. The compression model is introduced in Section 3, which discusses general tensor ring factorizations, and their specific application to fully connected and convolutional layers. The compression method for convolutional layers is a key novelty, as few previous papers extend factorization-based compression methods beyond fully connected layers. Finally, we show our experimental results improve upon the state-of-the-art in compressibility without significant performance degradation in Section 4. Section 6 concludes the paper with possible future directions.

2. Related Work

Past deep neural network compression techniques have largely applied to fully connected layers, which previously have dominated the number of parameters of a model. However, since modern models like ResNet and WideResNet are moving toward wider convolutional layers and omitting fully connected layers altogether, it is important to consider compression schemes that work on both fronts.

Many modern compression schemes focus on post-processing techniques, such as hashing [9] and quantization [20]. A strength of these methods is that they can be applied in addition to any other compression scheme, and are thus orthogonal to other methods. More similar to our work are novel representations like circulant projections [10] and truncated SVD representations [18].

Low-rank tensor approximation of deep neural networks has been widely investigated in the literature for effective model compression, low generative error, and fast prediction speed [42, 28, 35]. Tensor Networks (TNs) [11, 12] have recently drawn considerable attention in multi-dimensional data representation [46, 47, 17, 48], and deep learning [14, 15, 13, 31].

One of the most popular methods of tensor factorization is the Tucker factorization [44], and has been shown to exhibit good performance in data representation [17, 5, 4] and in compressing fully connected layers in deep neural networks [31]. In [28], a Tucker decomposition approach is applied to compress both fully connected layers and convolution layers.

Tensor train (TT) representation [39] is another example of TNs that factorizes a tensor into boundary two matrices and a set of 3^{rd} order tensors, and has demonstrated its capability in data representation [40, 46, 7] and deep learning [37, 51]. In [47], the TT model is compared against TR for multi-dimensional data completion, showing that for the same intermediate rank, TR can be far more expressive than TT, motivating the generalization. In this paper, we investigate TR for deep neural network compression.

3. Tensor Ring Nets (TRN)

In this paper, $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ is a d mode tensor with $\prod_{i=1}^{d} I_i$ degrees of freedom. A *tensor ring decomposition* factors such an $\mathcal{X}$ into d independent 3-mode tensors, $\mathcal{U}^{(1)}, \ldots, \mathcal{U}^{(d)}$ such that each entry inside the tensor $\mathcal{X}$ is represented as

$$\mathcal{X}_{i_1,\cdots,i_d} = \sum_{r_1,\cdots,r_d} \mathcal{U}^{(1)}_{r_d,i_1,r_1} \mathcal{U}^{(2)}_{r_1,i_2,r_2} \cdots \mathcal{U}^{(d)}_{r_{d-1},i_d,r_d}, \quad (1)$$

where $\mathcal{U}^{(i)} \in \mathbb{R}^{R \times I_i \times R}$, and R is the *tensor ring rank*. [1] Under this low-rank factorization, the number of free parameters is reduced to $R^2 \sum_{i=1}^{d} I_i$ in the tensor ring factor form, which is significantly less than $\prod_{i=1}^{d} I_i$ in $\mathcal{X}$.

For notational ease, let $\mathcal{U} = \{\mathcal{U}^{(1)}, \cdots, \mathcal{U}^{(d)}\}$, and define **decomp**$(\mathcal{X}; R, d)$ as the operation to obtain d factors $\mathcal{U}^{(i)}$ with tensor ring rank R from $\mathcal{X}$, and **construct**$(\mathcal{U})$ as the operation to obtain $\mathcal{X}$ from $\mathcal{U}$.

Additionally, for $1 \leq k < j \leq d$, define the **merge** operation as $\mathcal{M} =$ **merge**$(\mathcal{U}, k, j)$ such that $\mathcal{U}_k, \mathcal{U}_{k+1}, \cdots, \mathcal{U}_j$ are merged into one single tensor $\mathcal{M}$ of dimension $R \times I_k \times I_{k+1} \times \cdots \times I_j \times R$, and each entry in $\mathcal{M}$ is

$$\mathcal{M}_{r_{k-1},i_k,i_{k+1},\cdots,i_j,r_j} = \sum_{r_k,\cdots,r_{j-1}} \mathcal{U}^{(k)}_{r_{k-1},i_k,r_k} \mathcal{U}^{(k+1)}_{r_k,i_{k+1},r_{k+1}} \cdots \mathcal{U}^{(j)}_{r_{j-1},i_j,r_j}. \quad (2)$$

Note that construct operator is the merge operation **merge**$(\mathcal{U}, 1, d)$, which results in a tensor of shape $R \times I_1 \times I_2 \times \cdots \times I_d \times R$, followed by summing along mode 1 and mode $d+2$, resulting in a tensor of shape $I_1 \times I_2 \times \cdots \times I_d$; e.g.

$$\textbf{construct}(\mathcal{U}) = \sum_{r=1}^{R} \textbf{merge}(\mathcal{U}, 1, d)_{r,:,:,r}.$$

[1] More generally, $\mathcal{U}^{(i)} \in \mathbb{R}^{R_i \times I_i \times R_{i+1}}$ and each R_i may not be the same. For simplicity, we assume $R_1 = \cdots = R_d = R$.

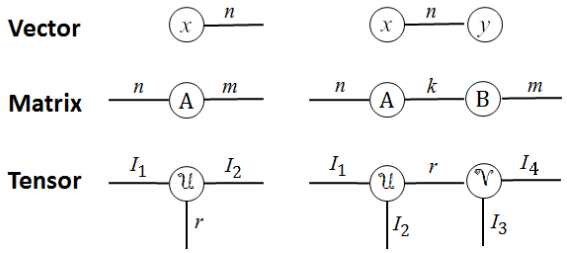


Figure 1: **Tensor diagrams.** Left: A graphical representation of a length n vector x, a $n \times m$ matrix A, and a 3rd order $I_1 \times I_2 \times I_3$ tensor $\mathcal{U}$. Right: factorized forms for a dot product $x^T y$, matrix product AB where A and B have k rows and columns respectively, and the tensor product of $\mathcal{U}$ and $\mathcal{V}$ along a common axis. More explicitly, the tensor product on the bottom right has 4 orders and the i_1, i_2, i_3, i_4-th element is $\sum_{j=1}^{r} \mathcal{U}_{i_1,i_2,j} \mathcal{V}_{i_3,i_4,j}$ for $i_k = 1, \ldots, I_k$, $k = 1, 2, 3, 4$.

Tensor diagrams Figure 1 introduces the popular tensor diagram notation [38], which represents tensor objects as nodes and their axes as edges of an undirected graph. An edge connecting two nodes indicates multiplication along that axis, and a "dangling" edge shows an axis in the remaining product, with the dimension given as the edge weight. This compact notation is useful in representing various factorization methods (Figure 2).

Merge ordering The computation complexity in this paper is measured in flops (counting additions and multiplications). The number of flops for a **construct** depends on the sequence of merging $\mathcal{U}^{(i)}, i = 1, \cdots, d$. (See figure 3). A detailed analysis of the two schemes is given in appendix A, resulting in the following conclusions.

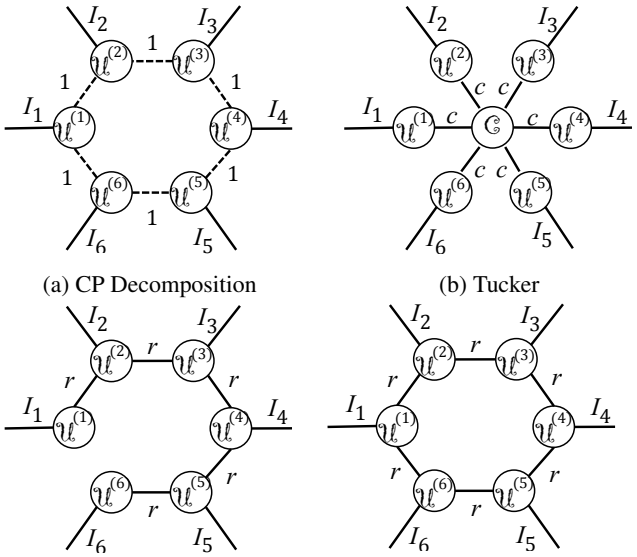
Theorem 1. *Suppose* $I_1 = \cdots = I_d \geq 2$ *and* $I = \prod_{i=1}^{d} I_i$. *Then*

1. *any merge order costs between* $2R^3I$ *and* $4R^3I$ *flops,*
2. *any merge order costs requires storing between* R^2I *and* $2R^2I$ *floats, and*
3. *if* d *is a power of 2, then a hierarchical merge order achieves the minimum flop count.*

Proof. See appendix A. □

Several interpretations can be made from these observations. First, though different merge orderings give different flop counts, the worst choice is at most 2x more expensive than the best choice. However, since we have to make some kind of choice, we note that since every merge order is a combination of hierarchical and sequential merges, striving toward a hierarchical merging is a good heuristic to minimize flop count. Thus, in our paper, we always use this strategy.

A *Tensor Ring Network (TRN)* is a tensor factorization of either fully connected layers (FCL) or convolutional layers (ConvL), trained via back propagation. If a pre-trained model is given, a good initialization can be obtained from the tensor ring decomposition of the layers in the pre-trained model.



(a) CP Decomposition (b) Tucker (c) Tensor Train (TT) (d) Tensor Ring(TR)

Figure 2: **Tensor decompositions.** Tensor diagrams for four popular tensor factorization methods: (a) the CP decomposition (unnormalized), (b) the Tucker decomposition, (c) the Tensor Train (TT) decomposition, and (d) the Tensor Ring (TR) decomposition used in this paper. As shown, TR can be viewed as a generalization of both CP (with $r > 1$) and TT (with an added edge connecting the first and last tensors). In Section 4.3, we also compare against Tucker decomposition compression schemes.

3.1. Fully Connected Layer Compression

In feed-forward neural networks, an input feature vector $\mathbf{x} \in \mathbb{R}^I$ is mapped to an output feature vector $\mathbf{y} = \mathbf{A}\mathbf{x} \in \mathbb{R}^O$ via a fully connected layer $\mathbf{A} \in \mathbb{R}^{I \times O}$. Without loss of generality, $\mathbf{x}$, $\mathbf{A}$, and $\mathbf{y}$ can be reshaped into higher order tensors $\mathcal{X}$, $\mathcal{A}$, and $\mathcal{Y}$ with

$$\mathcal{Y}_{o_1,\ldots,o_{\hat{d}}} = \sum_{i_1,\ldots,i_d} \mathcal{A}_{i_1,\ldots,i_d,o_1,\ldots,o_{\hat{d}}} \mathcal{X}_{i_1,\ldots,i_d} \tag{3}$$

where d and $\hat{d}$ are the modes of $\mathcal{X}$ and $\mathcal{Y}$ respectively, and i_k's ad o_k's span from 1 to I_k and 1 to O_k respectively, and

$$\prod_{i=1}^{d} I_i = I, \quad \prod_{i=1}^{\hat{d}} O_i = O.$$

To compress a feed-forward network, we decompose as $\mathcal{U} = \{\mathcal{U}^{(1)}, \ldots, \mathcal{U}^{(d+\hat{d})}\} = \textbf{decomp}(\mathcal{A}; R, d + \hat{d})$ and replace $\mathcal{A}$ with its decomposed version in (3). A tensor diagram for this operation is given in Figure 4, which shows

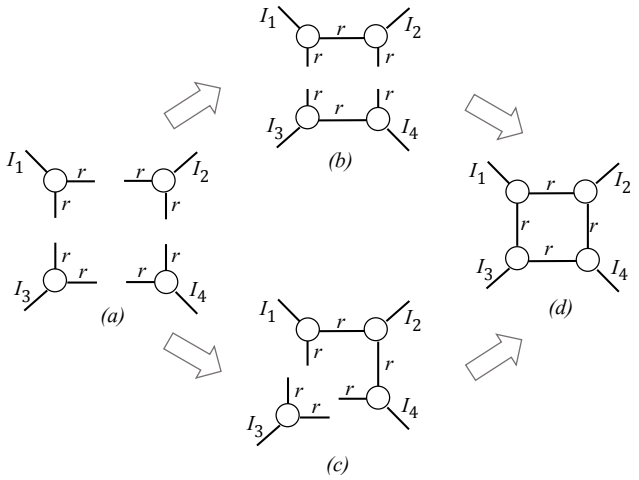


Figure 3: **Merge ordering.** A 4th order tensor is merged from its factored form, either hierarchically via (a)→(b)→(d), or sequentially via (a)→(c)→(d). Note that the computational complexity of forming (b) is $r^3(I_1I_2 + I_3I_4)$ and for (c) is $r^3(I_1I_2 + I_1I_2I_4)$, and (c) is generally more expensive (if $I_1 \approx I_2 \approx I_3 \approx I_4$). This is discussed in detail in Appendix A.

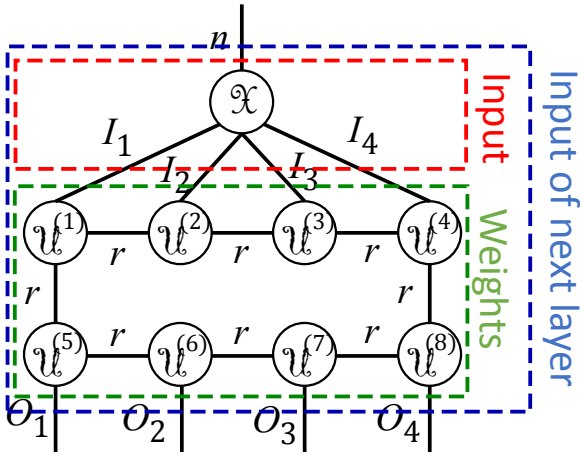


Figure 4: **Fully connected layer.** Tensor diagram of a fully connected TRN, divided into input and weights. The composite tensor is the input into the next layer.

how each multiplication is applied and the resulting dimensions.

Computational cost The computational cost again depends on the order of merging $\mathcal{X}$ and $\mathcal{U}$. Note that there is no need to fully construct the tensor $\mathcal{A}$, and a tensor representation of $\mathcal{A}$ is sufficient to obtain $\mathcal{Y}$ from $\mathcal{X}$. To reduce the computational cost, a *layer separation* approach is proposed by first using hierarchical merging to obtain

$$\begin{aligned}\mathcal{F}^{(1)} &= \textbf{merge}(\mathcal{U}, 1, d) \in \mathbb{R}^{R \times I_1 \times \cdots \times I_d \times R} \\ \mathcal{F}^{(2)} &= \textbf{merge}(\mathcal{U}, d+1, d+\hat{d}) \in \mathbb{R}^{R \times O_1 \times \cdots \times O_{\hat{d}} \times R},\end{aligned} \tag{4}$$

which is upper bounded by $4R^3(I + O)$ flops. By replacing $\mathcal{A}$ in (3) with $\mathcal{F}^{(1)}$ and $\mathcal{F}^{(2)}$ and switching the order of summation, we obtain

$$\mathcal{Z}_{r_d, r_{d+\hat{d}}} = \sum_{i_1,\ldots,i_d} \mathcal{F}^{(1)}_{r_{d+\hat{d}}, i_1, \cdots, i_d, r_d} \mathcal{X}_{i_1,\ldots,i_d}, \tag{5}$$

$$\mathcal{Y}_{o_1,\ldots,o_{\hat{d}}} = \sum_{r_{d+\hat{d}}, r_d} \mathcal{Z}_{r_d, r_{d+\hat{d}}} \mathcal{F}^{(2)}_{r_d, o_1, \cdots, o_{\hat{d}}, r_{d+\hat{d}}}. \tag{6}$$

The summation (5) is equivalent to a feed-forward layer of shape $(I_1 \cdots I_d) \times R^2$, which takes $2R^2I$ flops. Additionally, the summation over $r_{d+\hat{d}}$ and r_d is equivalent to another feed-forward layer of shape $R^2 \times (O_1 \cdots O_{\hat{d}})$, which takes $2R^2O$ flops. Such analysis demonstrates that the *layer separation* approach to a FCL in a tensor ring net is equivalent to a low-rank matrix factorization to a fully-connected layer, thus reducing the computational complexity when R is relatively smaller than I and O.

Define P_{FC} and C_{FC} as the complexity saving in parameters and computation, respectively, for the tensor net decomposition over the typical fully connected layer forward propagation. Thus we have

$$P_{\text{FC}} = \frac{IO}{R^2\left(\sum_i^d I_i + \sum_j^{\hat{d}} O_j\right)}. \tag{7}$$

and

$$C_{\text{FC}} \geq \frac{2BIO}{(4R^3 + 2BR^2)(I + O)}, \tag{8}$$

where B is the batch size of testing samples. Here, we see the compression benefit in computation; when B is very large, (8) converges to $IO/(R^2(I+O))$, which for large I, O and small R is significant. Additionally, though the expensive reshaping step grows cubically with R (as before), it does not grow with batch size; conversely, the multiplication itself (which grows linearly with batch size) is only quadratic in R. In the paper, the parameter is selected by picking small R and large d to achieve the optimal C since R needs to be small enough for computation saving.

3.2. Convolutional Layer Compression

In convolutional neural networks (CNNs), an input tensor $\mathcal{X} \in \mathbb{R}^{H \times W \times I}$ is convoluted with a 4th order kernel tensor $\mathcal{K} \in \mathbb{R}^{D \times D \times I \times O}$ and mapped to a 3rd order tensor $\mathcal{Y} \in \mathbb{R}^{H \times W \times O}$, as follows

$$\begin{aligned}\mathcal{Y}_{h,w,o} &= \sum_{d_1,d_2=1}^{D} \sum_{i=1}^{I} \mathcal{X}_{h',w',i} \mathcal{K}_{d_1,d_2,i,o}, \\ h' &= (h-1)s + d_1 - p, \\ w' &= (w-1)s + d_2 - p,\end{aligned} \tag{9}$$

where s is stride size, p is zero-padding size. Computed as in (9), the flop cost is $D^2 \cdot IO \cdot HW$. [2]

In TRN, tensor ring decomposition is applied onto the kernel tensor $\mathcal{K}$ and factorizes the 4th order tensor into four 3rd tensors. With the purpose to maintain the spatial information in the kernel tensor, we do not factorize the spatial dimension of $\mathcal{K}$ via merging the spatial dimension into one $4th$ order tensor $\mathcal{V}^{(1)}_{R_1,D_1,D_2,R_2}$, thus we have

$$\mathcal{K}_{d_1,d_2,i,o} = \sum_{r_1,r_2,r_3=1}^{R} \mathcal{V}_{r_1,d_1,d_2,r_2} \mathcal{U}_{r_2,i,r_3} \hat{\mathcal{U}}_{r_3,o,r_1}. \quad (10)$$

In the scenario when I and O are large, the tensors $\mathcal{U}$ and $\hat{\mathcal{U}}$ are further decomposed into $\mathcal{U}^{(1)}, \ldots, \mathcal{U}^{(d)}$ and $\mathcal{U}^{(d+1)}, \ldots, \mathcal{U}^{(d+\hat{d})}$ respectively. (See also Figure 5.)

The kernel tensor factorization in (10) combined with the convolution operation in (9) can be equivalently solved in three steps:

$$\mathcal{P}_{h',w',r_2,r_3} = \sum_{i=1}^{I} \mathcal{X}_{h',w',i} \mathcal{U}^{(2)}_{r_2,i,r_3} \quad (11)$$

$$\mathcal{Q}_{h,w,r_3,r_1} = \sum_{d_1,d_2=1}^{D} \sum_{r_2}^{R} \mathcal{P}_{h',w',r_2,r_3} \mathcal{U}^{(1)}_{r_1,d_1,d_2,r_2} \quad (12)$$

$$\mathcal{Z}_{h,w,o} = \sum_{r_1,r_3} \mathcal{Q}_{h,w,r_3,r_1} \mathcal{U}^{(3)}_{r_3,o,r_1}. \quad (13)$$

where (11) is a tensor multiplication along one slice, with flop count HWR^2I, (12) is a 2-D convolution with flop count HWR^3D^2, and (13) is a tensor multiplication along 3 slices with flop count HWR^2O. This is also equivalent to a three-layer convolutional networks without non-linear transformations, where (11) is a convolutional layer from I feature maps to R^2 feature maps with a 1×1 patch, (12) contains R convolutional layers from R feature maps to R feature maps with a $D \times D$ patch, and (13) is a convolutional layer from R^2 feature maps to O feature maps with with a 1×1 patch. This is a common sub-architecture choice in other deep CNNs, like the inception module in GoogleNets [43], but without nonlinearities between 1×1 and $D \times D$ convolution layers.

Complexity: We employ the ratio between complexity in CNN layer and the complexity in tensor ring layer to quantify the capability of TRN in reducing computation (C_{conv}) and parameter (P_{conv}) costs,

$$P_{\text{conv}} = \frac{D^2IO}{D^2R^2 + IR^2 + OR^2}, \qquad C_{\text{conv}} = \frac{IO \cdot D^2}{R^2I + R^3D^2 + R^2O}. \quad (14)$$

[2] For small filter sizes $D \ll \log(HW)$, as is often the case in deep neural networks for image processing, often direct multiplication to compute convolution is more efficient than using an FFT, which for this problem has order $IO(HW(\log(HW)))$ flops. Therefore we only consider direct multiplication as a baseline.

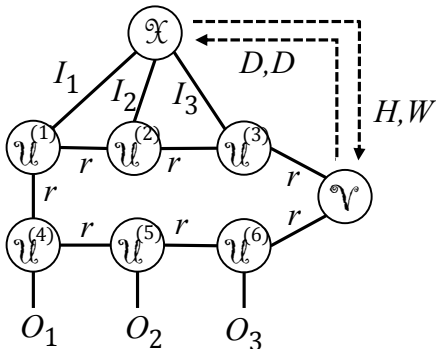


Figure 5: **Convolutional layer.** Dashed lines show the convolution operation (9). Here, $\mathcal{U}^{(1)}$, $\mathcal{U}^{(2)}$ and $\mathcal{U}^{(3)}$ decompose $\mathcal{U}$ and $\mathcal{U}^{(4)}$, $\mathcal{U}^{(5)}$, and $\mathcal{U}^{(6)}$ decompose $\hat{\mathcal{U}}$ in (10). The dashed line between $\mathcal{X}$ and $\mathcal{V}$ represent the convolution operation as expressed in (9). Note that $I_1 \times I_2 \times I_3$ decompose the number of *channels* entering the layer (which is 1 at the first input), where in Figure 4 they decompose the *feature dimension* entering the layer.

If, additionally, the tensors $\mathcal{U}^{(1)}$ and $\mathcal{U}^{(2)}$ are further decomposed to d and $\hat{d}$ tensors, respectively, then

$$P_{\text{conv}} = \frac{D^2IO}{D^2R^2 + R^2(\sum_i^d I_i + \sum_j^{\hat{d}} O_j)}, \qquad C_{\text{conv}} = \frac{BIO \cdot D^2}{4R^3(I+O) + BR^2(I+O) + BR^3D^2}. \quad (15)$$

Note that in the second scenario, we have a further compression in storage requirements, but lose gain in computational complexity, which is a design tradeoff. In our experiments, we further factorize $\mathcal{U}^{(1)}$ and $\mathcal{U}^{(3)}$ in to higher order tensors in order to achieve our gain in model compression.

Initialization In general nonconvex optimization (especially for deep learning), the choice of initial variables can dramatically effect the quality of the model training. In particular, we have found that initializing each parameter randomly from a Gaussian distribution is effective, with a carefully chosen variance. If we initialize all tensor factors as drawn i.i.d. from $\mathcal{N}(0, \sigma^2)$, then after merging d factors the merged tensor elements will have mean 0 and variance $R^d\sigma^{2d}$ (See appendix B). By picking $\sigma = \left(\frac{2}{N}\right)^{1/d} \frac{1}{\sqrt{R}}$, where N is the amount of parameters in the uncompressed layer, the merged tensor will have mean 0, variance $\sqrt{2/N}$, and in the limit will also be Gaussian. Since this latter distribution works well in training the uncompressed models, choosing this value of σ for initialization is well-motivated, and observed to be necessary for faster convergence.

4. Experiments

We now evaluate the effectiveness of TRN-based compression on several well-studied deep neural networks and

datasets: LeNet-300-100 and LeNet-5 on MNIST, and ResNet and WideResNet on Cifar10 and Cifar100. These networks are trained using Tensorflow [3]. All the experiments on LeNet are implemented on Nvidia GTX 1070 GPUs, and all the experiments for ResNet and WideResNet are implemented on Nvidia GTX Titan X GPUs. In all cases, the same tensor ring rank r is used in the networks, and all the networks are trained from randomly initialization using the the proposed initialization method. Overall, we show that this compression scheme can give significant compression gains for small accuracy loss, and even negligible compression gains for no accuracy loss.

4.1. Fully connected layer compression

The goal of compressing the LeNet-300-100 network is to assess the effectiveness of compressing fully connected layers using TRNs; as the name suggests, LeNet-300-100 contains two hidden fully connected layers with output dimension 300 and 100, and an output layer with dimension 10 (= # classes). Table 1 gives the parameter settings for LeNet-300-100, both in its original form (uncompressed) and in its tensor factored form. A compression rate greater than 1 is achieved for all $r \leq 54$, and a reduction in computational complexity for all $r \leq 6$; both are typical choices.

Table 2 shows the performance results on MNIST classification for the original model (as reported in their paper), and compressed models using both matrix factorization and TRNs. For a 0.14% accuracy loss, TRN can compress up to $13\times$, and for no accuracy loss, can compress $1.2\times$. Note also that matrix factorization, at $16\times$ compression, performs worse than TRN at $117\times$ compression, suggesting that the high order structure is helpful. Note also that low rank Tucker approximation in [28] is equivalent to low rank matrix approximation when compressing fully connected layer.

4.2. Convolutional layer compression

We now investigate compression of convolutional layers in a small network. LeNet-5 is a (relatively small) convolutional neural networks with 2 convolution layers, followed by 2 fully connected layers, which achieves 0.79% error rate on MNIST. The dimensions before and after compression are given in Table 3. In this wider network we see a much greater potential for compression, with positive compression rate whenever $r \leq 57$. However, the reduction in complexity is more limited, and only occurs when $r \leq 4$.

However, the performance on this experiment is still positive. By setting $r = 20$, we compress LeNet-5 by $11\times$ and a lower error rate than the original model as well as the Tucker factorization approach. If we also require a reduction in flop count, we incur an error of 2.24%, which is still quite reasonable in many real applications.

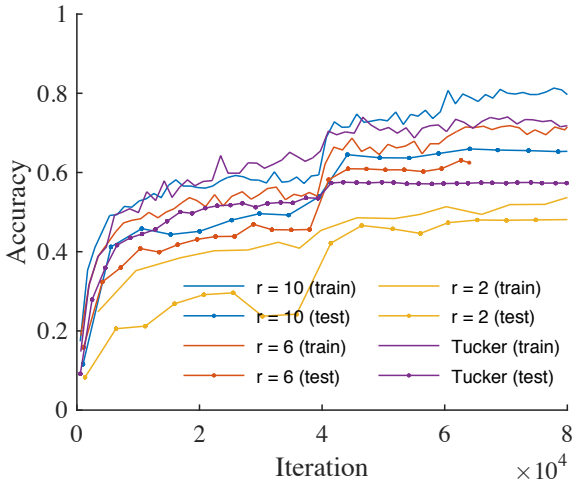


Figure 6: **Evolution.** Evolution of training compressed 32 layer ResNet on Cifar100, using TRNs with different values of r and the Tucker factorization method.

4.3. ResNet and Wide ResNet Compression

Finally, we evaluate the performance of tensor ring nets (TRN) on the Cifar10 and Cifar100 image classification tasks [32]. Here, the input images are colored, of size $32 \times 32 \times 3$, belonging to 10 and 100 object classes respectively. Overall there are 50000 images for training and 10000 images for testing.

Table 5 gives the dimensions of ResNet before and after compression. A similar reshaping scheme is used for WideResNet. Note that for ResNet, we have compression gain for any $r \leq 22$; for WideResNet this bound is closer to $r \leq 150$, suggesting high compression potential.

The results are given in Table 6 demonstrates that TRNs are able to significantly compress both ResNet and WideResNet for both tasks. Picking $r = 10$ for TRN on ResNet gives the same compression ratio as the Tucker compression method [28], but with almost 3% performance lift on Cifar10 and almost 10% lift on Cifar 100. Compared to the uncompressed model, we see only a 2% performance degradation on both datasets.

The compression of WideResNet is even more successful, suggesting that TRNs are well-suited for these extremely overparametrized models. At a $243\times$ compression TRNs give a better performance on Cifar10 than uncompressed ResNet (but with fewer parameters) and only a 2% decay from the uncompressed WideResNet. For Cifar100, this decay increases to 8%, but again TRN of WideResNet achieves lower error than uncompressed ResNet, with overall fewer parameters. Compared against the Tucker compression method [28], at $5\times$ compression rate TRNs incur only 2-3% performance degradation on both datasets, while

	Uncompressed dims.			TRN dimensions		
layer	shape	# params	flops	shape of composite tensor	# params	flops
fc1	784×300	235K	470K	$(4 \times 7 \times 4 \times 7) \times (3 \times 4 \times 5 \times 5)$	$39r^2$	$1177r^3 + 1084r^2$
fc2	300×100	30K	60K	$(3 \times 4 \times 5 \times 5) \times (4 \times 5 \times 5)$	$31r^2$	$457r^3 + 400r^2$
fc3	100×10	1K	2K	$(4 \times 5 \times 5) \times (2 \times 5)$	$21r^2$	$127r^3 + 107r^2$
Total	-	266K	532K	-	$91r^2$	$1761r^3 + 1591r^2$

Table 1: **Fully connected compression.** Dimensions of the three-fully-connected layers in the uncompressed (left) and TRN-compressed (right) models. The computational complexity includes tensor product merging ($O(r^3)$) and feed-froward multiplication ($O(r^2)$).

Method	Params	CR	Err %	Test (s)	Train (s/epoch)	LR
LeNet-300-100 [36]	266K	$1\times$	2.50	0.011 ± 0.002	3.5 ± 1.0	$2e^{-4}$
M-FC[18, 28]($r = 10$)	16.4K	$16.3\times$	3.91	0.016 ± 0.010	6.4 ± 1.2	$1e^{-4}$
M-FC ($r = 20$)	31.2K	$5.3\times$	3.0	0.014 ± 0.010	5.2 ± 1.2	$1e^{-4}$
M-FC ($r = 50$)	75.7K	$3.5\times$	2.62	0.021 ± 0.012	8.1 ± 1.2	$1e^{-4}$
TRN ($r = 3$)	0.8K	$325.5\times$	8.53	0.015 ± 0.007	7.9 ± 1.4	$1e^{-3}$
TRN ($r = 5$)	2.3K	$117.2\times$	3.75	0.015 ± 0.007	7.8 ± 1.4	$2e^{-3}$
TRN ($r = 15$)	20.5K	$13.0\times$	2.64	0.015 ± 0.007	8.1 ± 1.4	$5e^{-4}$
TRN ($r = 50$)	227.5K	$1.2\times$	**2.31**	0.022 ± 0.008	11.1 ± 1.4	$5e^{-5}$

Table 2: **Fully connected results.** LeNet-300-100 on MNIST datase, trained to 40 epochs, using a minibatch size 50. Trained from random weight initialization. ADAM [29] is used for optimization. Testing time is per 10000 samples. CR = Compression ratio. LR = Learning rate.

	Uncompressed dims.			TRN dimensions		
layer	shape	# params	flops	shape	# params	flops
conv1	$5 \times 5 \times 1 \times 20$	0.5K	784K	$5 \times 5 \times 1 \times (4 \times 5)$	$19r^2$	$33408r^2 + 39245r^3$
conv2	$5 \times 5 \times 20 \times 50$	25K	5000K	$5 \times 5 \times (4 \times 5) \times (5 \times 10)$	$34r^2$	$17840r^2 + 5095r^3$
fc1	1250×320	400K	800K	$(5 \times 5 \times 5 \times 10) \times (5 \times 8 \times 8)$	$46r^2$	$1570r^2 + 1685r^3$
fc2	320×10	3K	6K	$(5 \times 8 \times 8) \times 10$	$31r^2$	$330r^2 + 360r^3$
Total	-	429K	6590K	-	$130r^2$	$53148r^2 + 46385r^3$

Table 3: **Small convolution compression.** Dimensions of LeNet-5 layers in its original form (left) and TRN-compressed (right). The computational complexity includes tensor product merging and convolution operation in (12) of $O(r^3)$, and convolution in (11) (13) of $O(r^2)$.

Method	Params	CR	Err %	Test (s)	Train (s/epoch)	LR
LeNet-5 [36]	429K	$1\times$	0.79	0.038 ± 0.027	1.6 ± 1.9	$5e^{-4}$
Tucker [28]	189K	$2\times$	0.85	0.066 ± 0.025	7.7 ± 3	$5e^{-4}$
TRN ($r = 3$)	1.5K	$286\times$	2.24	0.058 ± 0.026	8.3 ± 4.5	$5e^{-4}$
TRN ($r = 5$)	3.6K	$120\times$	1.64	0.072 ± 0.039	10.6 ± 7.1	$5e^{-4}$
TRN ($r = 10$)	11.0K	$39\times$	1.39	0.080 ± 0.025	15.6 ± 4.6	$2e^{-4}$
TRN ($r = 15$)	23.4K	$18\times$	0.81	0.039 ± 0.019	20.1 ± 16.0	$2e^{-4}$
TRN ($r = 20$)	40.7K	$11\times$	**0.69**	0.052 ± 0.028	27.8 ± 7.4	$1e^{-5}$

Table 4: **Small convolution results.** LeNet-5 on MNIST dataset, trained to 20 epochs, using a minibatch size 128. ADAM [29] is used for optimization. Testing time is per 10000 samples. CR = Compression ratio. LR = Learning rate.

Tucker incurs 5% and 11% performance degradation. The compressibility is even more significant for WideResNet, where to achieve the same performance as Tucker [28] at $5\times$ compression, TRNs can compress up to $243\times$ on Cifar10 and $286\times$ on Cifar100. The tradeoff is runtime; we observe the Tucker model trains at about 2 or 3 times faster than TRNs for the WideResNet compression. However, for memory-constrained devices, this tradeoff may still be desirable.

Evolution Figure 6 shows the train and test errors during training of compressed ResNet on the Cifar100 classifica-

	Uncompressed dims.		TRN dimensions	
layer	shape	# params	shape of composite tensor	# params
conv1	$3 \times 3 \times 3 \times 16$	432	$9 \times 3 \times (4 \times 2 \times 2)$	$20r^2$
unit1	ResBlock(3, 16, 16)	4608	$9 \times (4 \times 2 \times 2) \times (4 \times 2 \times 2)$	$50r^2$
	ResBlock(3, 16, 16) $\times$ 4	18432	$9 \times (4 \times 2 \times 2) \times (4 \times 2 \times 2)$	$200r^2$
unit2	ResBlock(3, 16, 32)	13824	$9 \times (4 \times 2 \times 2) \times (4 \times 4 \times 2)$	$56r^2$
	ResBlock(3, 32, 32) $\times$ 4	73728	$9 \times (4 \times 4 \times 2) \times (4 \times 4 \times 2)$	$232r^2$
unit3	ResBlock(3, 32, 64)	55296	$9 \times (4 \times 4 \times 2) \times (4 \times 4 \times 4)$	$64r^2$
	ResBlock(3, 64, 64) $\times$ 4	294912	$9 \times (4 \times 4 \times 4) \times (4 \times 4 \times 4)$	$264r^2$
fc1	64×10	650	$(4 \times 4 \times 4) \times 10$	$22r^2$
Total	-	0.46M	-	$908r^2$

Table 5: **Large convolution compression.** Dimensions of 32 layer ResNes on Cifar10 dataset. Each ResBlock(p,I,O) includes a sequence: input $\rightarrow$ Batch Normalization $\rightarrow$ ReLU $\rightarrow$ $p \times p \times I \times O$ convolution layer $\rightarrow$ Batch Normalization $\rightarrow$ ReLU $\rightarrow$ $p \times p \times O \times O$ convolution layer. The input of length I is inserted once at the beginning and again at the end of each unit. See [24] for more details.

	Cifar10			Cifar100		
Method	Params	CR	Err %	Params	CR	Err %
ResNet(RN)-32L	0.46M	1$\times$	7.50[2]	0.47M	1$\times$	31.9 [2]
Tucker-RN [28]	0.09M	5$\times$	12.3	0.094M	5$\times$	42.2
TT-RN($r = 13$) [19, 37]	0.096M	4.8$\times$	11.7	0.102M	4.6$\times$	37.1
TRN-RN ($r = 2$)	0.004M	115$\times$	22.2	0.012M	39$\times$	51.3
TRN-RN ($r = 6$)	0.03M	15$\times$	19.2	0.041M	12$\times$	36.6
TRN-RN ($r = 10$)	0.09M	5$\times$	9.4	0.097M	5$\times$	33.3
WideResNet(WRL)-28L	36.2M	1$\times$	5.0 [2]	36.3M	1$\times$	21.7 [2]
Tucker-WRN [28]	6.7M	5$\times$	7.8	6.7M	5$\times$	30.8
TT-RN($r = 13$) [19, 37]	0.18M	201$\times$	8.4	0.235M	154$\times$	31.9
TRN-WRN ($r = 2$)	0.03M	1217$\times$	16.3	0.087M	417$\times$	43.9
TRN-WRN ($r = 6$)	0.07M	521$\times$	9.7	0.126M	286$\times$	30.3
TRN-WRN ($r = 10$)	0.15M	243$\times$	**7.3**	0.21M	173$\times$	28.3
TRN-WRN(r=15)	0.30M	122$\times$	**7.0**	0.36M	100$\times$	25.6

Table 6: **Large convolution results.** 32-layer ResNet (first 5 rows) and 28-layer Wide-ResNet (last 4 rows) on Cifar10 dataset and Cifar100 dataset, trained to 200 epochs, using a minibatch size of 128. The model is trained using SGD with momentum 0.9 and a decaying learning rate. CR = Compression ratio.

tion task, for various choices of r and also compared against Tucker tensor factorization. In particular, we note that the generalization gap (between train and test error) is particularly high for the Tucker tensor factorization method, while for TRNs (especially for smaller values of r) it is much smaller. For $r = 10$, both the generalization error and final train and test errors improve upon the Tucker method, suggesting that TRNs are easier to train.

5. Conclusion

We have introduced a tensor ring factorization approach to compress deep neural networks for resource-limited devices. This is inspired by previous work that has shown tensor rings to have high representative power in image completion tasks. Our results show significant compressibility using this technique, with little or no hit in performance on benchmark image classification tasks.

One area for future work is the reduction of computational complexity. Because of the repeated reshaping needs in both fully connected and convolutional layers, there is computational overhead, especially when r is moderately large. This tradeoff is reasonable, considering our considerable compressibility gains, and is appropriate in memory-limited applications, especially if training is offloaded to the cloud. Additionally, we believe that the actual wall-clock-time will decrease as tensor-specific hardware and low-level routines continue to develop–we observe, for example, that numpy's `dot` function is considerably more optimized than Tensorflow's `tensordot`. Overall, we believe this is a promising compression scheme and can open doors to using deep learning in a much more ubiquitous computing environment.

6. Acknowledgment

Wenqi Wang and Vaneet Aggarwal were supported in part by the U.S. National Science Foundation under grant CCF-1527486.

References

[1] https://socratic.org/questions/if-x-and-y-are-independent-random-variables-what-is-var-xy. 11

[2] Tensorflow Resnet, howpublished = https://github.com/tensorflow/models/tree/master/research/resnet. 8

[3] 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 *OSDI*, volume 16, pages 265–283, 2016. 6

[4] M. Ashraphijuo, V. Aggarwal, and X. Wang. Deterministic and probabilistic conditions for finite completability of low-tucker-rank tensor. *arXiv preprint arXiv:1612.01597*, 2016. 2

[5] M. Ashraphijuo, V. Aggarwal, and X. Wang. A characterization of sampling patterns for low-tucker-rank tensor completion problem. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pages 531–535. IEEE, 2017. 2

[6] M. Ashraphijuo, X. Wang, and V. Aggarwal. An approximation of the cp-rank of a partially sampled tensor. In *55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2017. 1

[7] M. Ashraphijuo, X. Wang, and V. Aggarwal. Rank determination for low-rank data completion. *The Journal of Machine Learning Research*, 18(1):3422–3450, 2017. 2

[8] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014. 1

[9] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015. 1, 2

[10] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2857–2865, 2015. 2

[11] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, D. P. Mandic, et al. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. *Foundations and Trends® in Machine Learning*, 9(4-5):249–429, 2016. 2

[12] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama, D. P. Mandic, et al. Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives. *Foundations and Trends® in Machine Learning*, 9(6):431–673, 2017. 2

[13] N. Cohen, O. Sharir, and A. Shashua. Deep SimNets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4782–4791, 2016. 2

[14] N. Cohen, O. Sharir, and A. Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016. 2

[15] N. Cohen and A. Shashua. Convolutional rectifier networks as generalized tensor decompositions. In *International Conference on Machine Learning*, pages 955–963, 2016. 2

[16] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167. ACM, 2008. 1

[17] G. Dai and D.-Y. Yeung. Tensor embedding methods. In *AAAI*, volume 6, pages 330–335, 2006. 2

[18] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014. 1, 2, 7

[19] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016. 8

[20] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. 2

[21] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013. 1

[22] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[23] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015. 1

[24] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 8

[25] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016. 1

[26] F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Studies in Applied Mathematics*, 6(1-4):164–189, 1927. 1

[27] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1

[28] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015. 1, 2, 6, 7, 8

[29] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7

[30] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009. 1

[31] J. Kossaifi, Z. C. Lipton, A. Khanna, T. Furlanello, and A. Anandkumar. Tensor regression networks. *arXiv preprint arXiv:1707.08308*, 2017. 2

[32] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009. 6

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[34] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar. An early resource characterization of deep learning on wearables, smartphones and Internet-of-things devices. In *Proceedings of the 2015 International Workshop on Internet of Things towards Applications*, pages 7–12. ACM, 2015. 1

[35] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In *International Conference on Learning Representations*, 2015. 2

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2, 7

[37] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015. 2, 8

[38] R. Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014. 3

[39] I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. 1, 2

[40] H. N. Phien, H. D. Tuan, J. A. Bengua, and M. N. Do. Efficient tensor completion: Low-rank tensor train. *arXiv preprint arXiv:1601.01083*, 2016. 2

[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[42] J. Sokolić, R. Giryes, G. Sapiro, and M. R. Rodrigues. Generalization error of deep neural networks: Role of classification margin and data structure. In *Sampling Theory and Applications (SampTA), 2017 International Conference on*, pages 147–151. IEEE, 2017. 2

[43] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1, 5

[44] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966. 2

[45] A. Van den Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. In *Advances in neural information processing systems*, pages 2643–2651, 2013. 1

[46] W. Wang, V. Aggarwal, and S. Aeron. Tensor completion by alternating minimization under the tensor train (TT) model. *arXiv preprint arXiv:1609.05587*, 2016. 2

[47] W. Wang, V. Aggarwal, and S. Aeron. Efficient low rank tensor ring completion. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1, 2

[48] W. Wang, V. Aggarwal, and S. Aeron. Tensor train neighborhood preserving embedding. *IEEE Transactions on Signal Processing*, PP(99):1–1, 2018. 2

[49] W. Wang, C. Chen, W. Wang, P. Rai, and L. Carin. Earliness-aware deep convolutional networks for early time series classification. *arXiv preprint arXiv:1611.04578*, 2016. 1

[50] W. Wang, Z. Gan, W. Wang, D. Shen, J. Huang, W. Ping, S. Satheesh, and L. Carin. Topic compositional neural language model. *arXiv preprint arXiv:1712.09783*, 2017. 1

[51] Y. Yang, D. Krompass, and V. Tresp. Tensor-train recurrent neural networks for video classification. *arXiv preprint arXiv:1707.01786*, 2017. 1, 2

[52] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 1, 2

[53] S. Zhang, L. Yao, and A. Sun. Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435*, 2017. 1

[54] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki. Tensor ring decomposition. *arXiv preprint arXiv:1606.05535*, 2016. 1