Convolutional Neural Network Compression through Generalized Kronecker Product Decomposition

Marawan Gamal Abdel Hameed[1,2*], **Marzieh S. Tahaei**[1†], **Ali Mosleh**[1], **Vahid Partovi Nia**[1]

[1]Noah's Ark Lab, Huawei Technologies Canada
[2]University of Waterloo
marawan.abdelhameed@uwaterloo.ca, {marzieh.tahaei, ali.mosleh, vahid.partovinia}@huawei.com

Abstract

Modern Convolutional Neural Network (CNN) architectures, despite their superiority in solving various problems, are generally too large to be deployed on resource constrained edge devices. In this paper, we reduce memory usage and floating-point operations required by convolutional layers in CNNs. We compress these layers by generalizing the Kronecker Product Decomposition to apply to multidimensional tensors, leading to the *Generalized Kronecker Product Decomposition* (GKPD). Our approach yields a plug-and-play module that can be used as a drop-in replacement for any convolutional layer. Experimental results for image classification on CIFAR-10 and ImageNet datasets using ResNet, MobileNetv2 and SeNet architectures substantiate the effectiveness of our proposed approach. We find that GKPD outperforms state-of-the-art decomposition methods including Tensor-Train and Tensor-Ring as well as other relevant compression methods such as pruning and knowledge distillation.

Introduction

Convolutional Neural Networks (CNNs) have achieved state-of-the-art performance on a wide range of computer vision tasks such as image classification (He et al. 2016), video recognition (Feichtenhofer et al. 2019) and object detection (Ren et al. 2015). Despite achieving remarkably low generalization errors, modern CNN architectures are typically over-parameterized and consist of millions of parameters. As the size of state-of-the-art CNN architectures continues to grow, it becomes more challenging to deploy these models on resource constrained edge devices that are limited in both memory and energy. Motivated by studies demonstrating that there is significant redundancy in CNN parameters (Denil et al. 2013), model compression techniques such as pruning, quantization, tensor decomposition and knowledge distillation have emerged to address this problem.

Decomposition methods have gained more attention in recent years as they can achieve higher compression rates in comparison to other approaches. Namely, Tucker (Kim et al. 2016), CP (Lebedev et al. 2015), Tensor-Train (Garipov et al. 2016) and Tensor-Ring (Wang et al. 2018) decompositions have been widely studied for DNNs. However, these methods still suffer significant accuracy loss for computer vision tasks.

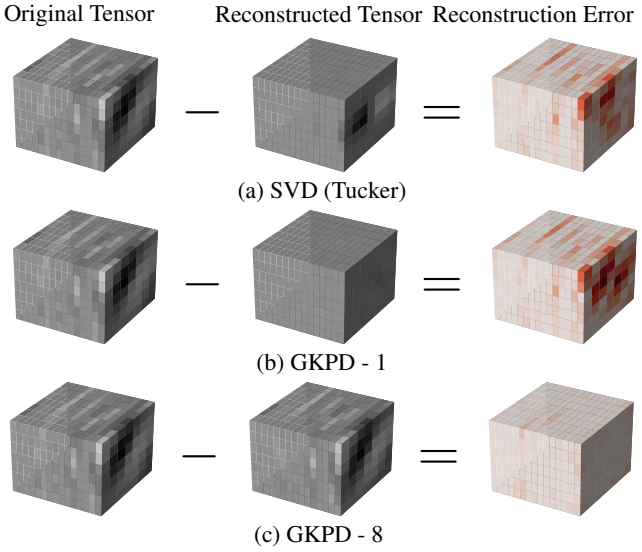


Figure 1: A compression rate of $2\times$ achieved for an arbitrary tensor from the first layer of ResNet18 using SVD (Tucker) in (a), and the proposed GKPD in (b) and (c). A larger summation, GKPD-8 achieves a lower reconstruction error in comparison with both a smaller summation, GKPD-1, as well as SVD (Tucker) decomposition.

Kronecker Product Decomposition (KPD) is another decomposition method that has recently shown to be very effective when applied to RNNs (Thakker et al. 2019). KPD leads to model compression via replacing a large matrix with two smaller Kronecker factor matrices that best approximate the original matrix. In this work, we generalize KPD to tensors, yielding the *Generalized Kronecker Product Decomposition* (GKPD), and use it to decompose convolution tensors. GKPD involves finding the summation of Kronecker products between factor tensors that best approximates the original tensor. We provide a solution to this problem called the *Multidimensional Nearest Kronecker Product Problem*. By formulating the convolution operation directly in terms

*Work done during an internship at Huawei Noah's Ark Lab
†Corresponding Author

of the Kronecker factors, we show that we can avoid reconstruction at runtime and thus obtain a significant reduction in memory footprints and floating-point operations (FLOPs). Once all convolution tensors in a pre-trained CNN have been replaced by their compressed counterparts, we retrain the network. If a pretrained network is not available, we show that we are still able to train our compressed network from a random initialization. Furthermore, we show that these randomly initialized networks retain universal approximation capability by building on (Hornik 1991) and (Zhou 2020).

Applying GKPD to an arbitrary tensor leads to multiple possible decompositions, one for each configuration of Kronecker factors. As shown in Figure 1, we find that for any given compression factor, choosing a decomposition that consists of a larger summation of smaller Kronecker factors (as opposed to a smaller summation of larger Kronecker factors) leads to a lower reconstruction error as well as improved model accuracy.

To summarize, the main contributions of this paper are:

- Generalizing the Kronecker Product Decomposition to multidimensional tensors
- Introducing the Multidimenesional Nearest Kronecker Product Problem and providing a solution
- Providing experimental results for image classification on CIFAR-10 and ImageNet using compressed ResNet (He et al. 2016), MobileNetv2 (Sandler et al. 2018) and SeNet (Hu, Shen, and Sun 2018) architectures.

Related Work on DNN Model Compression

Quantization methods focus on reducing the precision of parameters and/or activations into lower-bit representations. For example, the work in (Han, Mao, and Dally 2015) quantizes the parameter precision from 32 bits to 8 bits or lower. Model weights have been quantized even further into binary (Courbariaux, Bengio, and David 2015; Rastegari et al. 2016; Courbariaux et al. 2016; Hubara et al. 2017), and ternary (Li, Zhang, and Liu 2016; Zhu et al. 2016) representations. In these methods, choosing between a uniform (Jacob et al. 2018) or nonuniform (Han, Mao, and Dally 2015; Tang, Hua, and Wang 2017; Zhang et al. 2018b) quantization interval affects the compression rate and the acceleration.

Pruning methods began by exploring unstructured network weights and deactivating small weights through applying sparsity regularization to the weight parameters (Liu et al. 2015; Han, Mao, and Dally 2015; Han et al. 2015) or considering statistics information from layers to guide the parameter selections in ThiNet (Luo, Wu, and Lin 2017). Unstructured pruning results in irregularities in the weight parameters which impact the expected acceleration rate of the pruned network. Hence, several works aim at zeroing out structured groups of the convolutional filters through group sparsity regularization (Zhou, Alvarez, and Porikli 2016; Wen et al. 2016; Alvarez and Salzmann 2016). Sparsity regularization has been combined with other forms of regularizers such as low-rank (Alvarez and Salzmann 2017), ordered weighted ℓ_1 (Zhang et al. 2018a), and out-in-channel sparsity (Li et al. 2019) regularizers to further improve the pruning performance.

Decomposition methods factorize the original weight matrix or tensor into lightweight representations. This results in much fewer parameters and consequently fewer computations. Applying truncated singular value decomposition (SVD) to compress the weight matrix of fully-connected layers is one of the earliest works in this category (Denton et al. 2014). In the same vein, canonical polyadic (CP) decomposition of the kernel tensors was proposed in (Lebedev et al. 2015). This work uses nonlinear least squares to decompose the original convolution kernel into a set of rank-1 tensors (vectors). An alternative tensor decomposition approach to convolution kernel compression is Tucker decomposition (Tucker 1963). Tucker decomposition has shown to provide more flexible interaction between the factor matrices through a core tensor. The idea of reshaping weights of fully-connected layers into high-dimensional tensors and representing them in Tensor-Train format (Oseledets 2011) was extended to CNNs in (Garipov et al. 2016). Tensor-Ring decomposition has also become another popular option to compress CNNs (Wang et al. 2018). For multidimensional data completion with a same intermediate rank, TR can be far more expressive than TT (Wang, Aggarwal, and Aeron 2017). Kronecker factorization was also used to replace the weight matrices and weight tensors within fully-connected and convolution layers (Zhou et al. 2015). This work however limited the representation to a single Kronecker product and trained the model with random initialization. As shown in Fig.1 and in the next sections of this paper, summation can significantly improve the representation power of the network and thus leads to a performance increase.

Other model compression forms can also be achieved through sharing convolutional weight matrices in a more structured manner as ShaResNet (Boulch 2018) which reuses convolutional mappings within the same scale level or FSNet (Yang et al. 2020) which shares filter weights across spatial locations. NNs can also be compressed using Knowledge Distillation (KD) where a large (teacher) pre-trained network is used to train a smaller (student) network (Mirzadeh et al. 2020; Heo et al. 2019). Designing lightweight CNNs such as MobileNet (Sandler et al. 2018) and SqueezeNet (Iandola et al. 2016) is another form of model compression.

Preliminaries

Given matrices $\mathbf{A} \in \mathbb{R}^{m_1 \times n_1}$ and $\mathbf{B} \in \mathbb{R}^{m_2 \times n_2}$, their Kronecker product is the $m_1 m_2 \times n_1 n_2$ matrix

$$\mathbf{A} \otimes \mathbf{B} \triangleq \begin{bmatrix} a_{1,1}\mathbf{B} & \dots & a_{1,n_1}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m_1,1}\mathbf{B} & \dots & a_{m_1,n_1}\mathbf{B} \end{bmatrix}. \quad (1)$$

As shown in Van Loan (2000), any matrix $\mathbf{W} \in \mathbb{R}^{m_1 m_2 \times n_1 n_2}$ can be decomposed into a sum of Kronecker products as

$$\mathbf{W} = \sum_{r=1}^{R} \mathbf{A}_r \otimes \mathbf{B}_r, \quad (2)$$

where

$$R = \min(m_1 n_1, m_2 n_2) \quad (3)$$

is the rank of a reshaped version of matrix $\mathbf{W}$. We call this R the *Kronecker rank* of $\mathbf{W}$. Note that the Kronecker rank is not unique, and is dependent on the dimensions of factors $\mathbf{A}$ and $\mathbf{B}$.

To compress a given $\mathbf{W}$, we can represent it using a small number $\widehat{R} < R$ of Kronecker products that best approximate the original tensor. The factors are found by solving the Nearest Kronecker Product problem

$$\min_{\{\mathbf{A}_r\},\{\mathbf{B}_r\}} \left\| \mathbf{W} - \sum_{r=1}^{\widehat{R}} \mathbf{A}_r \otimes \mathbf{B}_r \right\|_F^2 . \tag{4}$$

As this approximation replaces a large matrix with a sequence of two smaller ones, memory consumption is reduced by a factor of

$$\frac{m_1 m_2 n_1 n_2}{\widehat{R}(m_1 n_1 + m_2 n_2)}. \tag{5}$$

Furthermore, if a matrix $\mathbf{W}$ is decomposed into its Kronecker factors then the projection $\mathbf{W}\mathbf{x}$ can be performed without explicit reconstruction of $\mathbf{W}$. Instead, the factors can be used directly to perform the computation as a result of the following equivalency relationship:

$$\mathbf{y} = (\mathbf{A} \otimes \mathbf{B})\mathbf{x} \equiv \mathbf{Y} = \mathbf{B}\mathbf{X}\mathbf{A}^\top, \tag{6}$$

where $\text{vec}(\mathbf{X}) = \mathbf{x}$, $\text{vec}(\mathbf{Y}) = \mathbf{y}$ and $\text{vec}(\cdot)$ vectorizes matrices $\mathbf{X} \in \mathbb{R}^{n_2 \times n_1}$ and $\mathbf{Y} \in \mathbb{R}^{m_2 \times m_1}$ by stacking their columns.

Method

In this section, we extend KPD to tensors yielding GKPD. First, we define the multidimensional Kronecker product, then we introduce the Multidimensional Nearest Kronecker Product problem and its solution. Finally, we describe our *KroneckerConvolution* module that uses GKPD to compress convolution tensors and avoids reconstruction at runtime.

Generalized Kronecker Product Decomposition

We now turn to generalizing the Kronecker product to operate on tensors. Let $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{a_1 \times a_2 \times \cdots \times a_N}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{b_1 \times b_2 \times \cdots \times b_N}$ be two given tensors. Intuitively, tensor $(\boldsymbol{\mathcal{A}} \otimes \boldsymbol{\mathcal{B}}) \in \mathbb{R}^{a_1 b_1 \times a_2 b_2 \times \cdots \times a_N b_N}$ is constructed by *moving around* tensor $\boldsymbol{\mathcal{B}}$ in a non-overlapping fashion, and at each position scaling it by a corresponding element of $\boldsymbol{\mathcal{A}}$ as shown in Figure 2. Formally, the Multidimensional Kronecker product is defined as follows

$$(\boldsymbol{\mathcal{A}} \otimes \boldsymbol{\mathcal{B}})_{i_1,i_2,\ldots,i_N} \triangleq \boldsymbol{\mathcal{A}}_{j_1,j_2,\ldots,j_N} \boldsymbol{\mathcal{B}}_{k_1,k_2,\ldots,k_N}, \tag{7}$$

where

$$j_n = \left\lfloor \frac{i_n}{b_n} \right\rfloor \text{ and } \quad k_n = i_n \bmod b_n \tag{8}$$

represent the integer quotient and the remainder term of i_n with respect to divisor b_n, respectively.

As with matrices, any multidimensional tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{w_1 \times w_2 \times \cdots \times w_N}$ can be decomposed into a sum of Kronecker products

$$\boldsymbol{\mathcal{W}} = \sum_{r=1}^{R} \boldsymbol{\mathcal{A}}_r \otimes \boldsymbol{\mathcal{B}}_r, \tag{9}$$

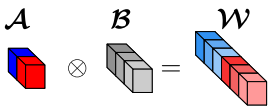


Figure 2: Illustration of Kronecker Decomposition of a single convolution filter (with spatial dimensions equal to one for simplicity).

where

$$R = \min(a_1 a_2 \cdots a_N, b_1 b_2 \cdots b_N) \tag{10}$$

denotes the Kronecker rank of tensor $\boldsymbol{\mathcal{W}}$. Thus, we can approximate $\boldsymbol{\mathcal{W}}$ using GKPD by solving the Multidimensional Nearest Kronecker Product problem

$$\min_{\{\boldsymbol{\mathcal{A}}_r\},\{\boldsymbol{\mathcal{B}}_r\}} \left\| \boldsymbol{\mathcal{W}} - \sum_{r=1}^{\widehat{R}} \boldsymbol{\mathcal{A}}_r \otimes \boldsymbol{\mathcal{B}}_r \right\|_F^2 , \tag{11}$$

where $\widehat{R} < R$. For the case of matrices (2D tensors), Van Loan and Pitsianis (1992) solved this problem using SVD. We extend their approach to the multidimensional setting. Our strategy will be to define rearrangement operators

$$\begin{aligned} \mathbf{R}_w &: \mathbb{R}^{w_1 \times w_2 \times \cdots \times w_N} \to \mathbb{R}^{a_1 a_2 \ldots a_N \times b_1 b_2 \ldots b_N} \\ \mathbf{r}_a &: \mathbb{R}^{a_1 \times a_2 \times \cdots \times a_N \to a_1 a_2 \ldots a_N} \\ \mathbf{r}_b &: \mathbb{R}^{b_1 \times b_2 \times \cdots \times b_N \to b_1 b_2 \ldots b_N} \end{aligned}$$

and solve

$$\min_{\{\boldsymbol{\mathcal{A}}_r\},\{\boldsymbol{\mathcal{B}}_r\}} \left\| \mathbf{R}_w(\boldsymbol{\mathcal{W}}) - \sum_{r=1}^{\widehat{R}} \mathbf{r}_a(\boldsymbol{\mathcal{A}}_r) \mathbf{r}_b(\boldsymbol{\mathcal{B}}_r)^\top \right\|_F^2 \tag{12}$$

instead. By carefully defining the rearrangement operators, the sum of squares in (12) is kept identical to that in (11). The former corresponds to finding the best low-rank approximation which has a well known solution using SVD. We define the rearrangement operators as follows:

$$\begin{aligned} R_w(\boldsymbol{\mathcal{W}})_{i,:} &= \text{vec}(\text{unfold}(\boldsymbol{\mathcal{W}}, \mathbf{d}_{\boldsymbol{\mathcal{B}}})_i) \\ \mathbf{r}_a(\boldsymbol{\mathcal{A}}) &= \text{unfold}(\boldsymbol{\mathcal{A}}, \mathbf{d}_{\boldsymbol{\mathcal{I}}_{\boldsymbol{\mathcal{A}}}}) \\ \mathbf{r}_b(\boldsymbol{\mathcal{B}}) &= \text{vec}(\boldsymbol{\mathcal{B}}) \end{aligned}$$

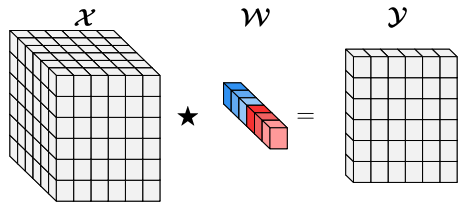
where

$$\text{unfold}(\boldsymbol{\mathcal{W}}, \mathbf{d}) : \mathbb{R}^{w_1 \times w_2 \times \cdots \times w_N} \to \mathbb{R}^{N_p \times d_1 \times d_2 \ldots d_N}$$

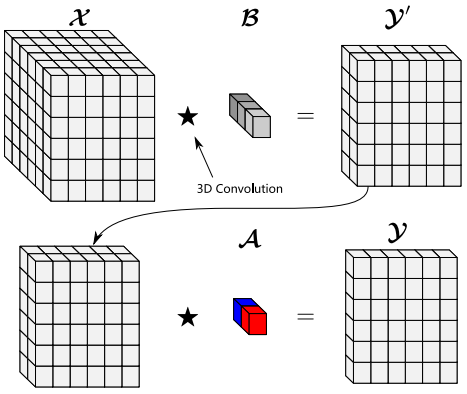
extracts N_p non-overlapping patches of shape $\mathbf{d}$ from tensor $\boldsymbol{\mathcal{W}}$, $\text{vec}(\cdot)$ flattens its input into a vector, tensor $\boldsymbol{\mathcal{I}}_{\boldsymbol{\mathcal{A}}}$ has the same number of dimensions as $\boldsymbol{\mathcal{A}}$ with each dimension equal to unity and $\mathbf{d}_{\boldsymbol{\mathcal{B}}}$ is a vector describing the shape of tensor $\boldsymbol{\mathcal{B}}$. While the ordering of patch extraction and flattening is not important, it must remain consistent across the rearrangement operators.

KroneckerConvolution Layer

The convolution operation in CNNs between a weight tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{F \times C \times K_w \times K_h}$ and an input $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{C \times H \times W}$ is a



(a) Conv2d



(b) KroneckerConv2d

Figure 3: Illustration of the KroneckerConvolution operation. Although (a) and (b) result in identical outputs, the latter is more efficient in terms of memory and FLOPs.

multilinear map that can be described in scalar form as

$$\mathcal{Y}_{f,x,y} = \sum_{i=1}^{K_h} \sum_{j=1}^{K_w} \sum_{c=1}^{C} \mathcal{W}_{f,c,i,j} \mathcal{X}_{c,i+x,j+y}. \quad (13)$$

Assuming $\mathcal{W}$ can be decomposed to KPD factors $\mathcal{A} \in \mathbb{R}^{F_1 \times C_1 \times K_{w1} \times K_{h1}}$ and $\mathcal{B} \in \mathbb{R}^{F_2 \times C_2 \times K_{w2} \times K_{h2}}$, we can rewrite (13) as

$$\mathcal{Y}_{f,x,y} = \sum_{i=1}^{K_h} \sum_{j=1}^{K_w} \sum_{c=1}^{C} (\mathcal{A} \otimes \mathcal{B})_{f,c,i,j} \mathcal{X}_{c,i+x,j+y}. \quad (14)$$

Due to the structure of tensor $\mathcal{A} \otimes \mathcal{B}$, we do not need to explicitly reconstruct it to carry out the summation in (14). Instead, we can carry out the summation by *directly* using elements of tensors $\mathcal{A}$ and $\mathcal{B}$ as shown in Lemma 1. This key insight leads to a large reduction in both memory and FLOPs. Effectively, this allows us to replace a large convolutional layer (with a large weight tensor) with two smaller ones, as we demonstrate in the rest of this section.

Lemma 1. *Suppose tensor $\mathcal{W} \in \mathbb{R}^{w_1 \times w_2 \times \cdots \times w_N}$ can be decomposed into KPD factors such that $\mathcal{W} = \mathcal{A} \otimes \mathcal{B}$. Then, the multilinear map involving $\mathcal{W}$ can be written directly in terms of its factors $\mathcal{A} \in \mathbb{R}^{a_1 \times a_2 \times \cdots \times a_N}$ and $\mathcal{B} \in \mathbb{R}^{b_1 \times b_2 \times \cdots \times b_N}$ as follows*

$$\mathcal{W}_{i_1,i_2,\ldots,i_N} \mathcal{X}_{i_1,i_2,\ldots,i_N} = \mathcal{A}_{j_1,j_2,\ldots,j_N} \mathcal{B}_{k_1,k_2,\ldots,k_N} \mathcal{X}_{g(j_1,k_1),g(j_2,k_2),\ldots,g(j_N,k_N)},$$

where $\mathcal{X} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_N}$ is an input tensor, $g(j_n, k_n) \triangleq j_n b_n + k_n$ is a re-indexing function; and j_n, k_n are as defined in (8)*. The equality also holds for any valid offsets to the input's indices*

$$\mathcal{W}_{i_1,i_2,\ldots,i_N} \mathcal{X}_{i_1+o_1,\, i_2+o_2,\, \ldots,\, i_N+o_N} = \mathcal{A}_{j_1,j_2,\ldots,j_N} \mathcal{B}_{k_1,k_2,\ldots,k_N} \mathcal{X}_{g(j_1,k_1)+o_1,\, g(j_2,k_2)+o_2,\, \ldots,\, g(j_N,k_N)+o_N},$$

where $o_i \in \mathbb{N}$.

Proof. See Supplementary Material. □

Algorithm 1: Forward Pass

Input:
$\mathcal{X} \in \mathbb{R}^{C \times W \times H}$
$\mathcal{A} \in \mathbb{R}^{F_1 \times C_1 \times K_{h1} \times K_{w1}}$
$\mathcal{B} \in \mathbb{R}^{F_2 \times C_2 \times K_{h2} \times K_{w2}}$
$\mathbf{s}_{\mathcal{W}} \in \mathbb{R}^4$ // Stride of original convolution

Output:
$\mathcal{Y} \in \mathbb{R}^{F \times W \times H}$

$\mathcal{X}' \leftarrow \text{Unsqueeze}(\mathcal{X}) \in \mathbb{R}^{1 \times C \times W \times H}$;
/* 3D Conv with stride of $(C_2, 1, 1)$ */
$\mathcal{Y}' \leftarrow \text{Conv3d}(\mathcal{B}, \mathcal{X}') \in \mathbb{R}^{F_2 \times C_1 \times W \times H}$;
/* Batched 2D Conv with stride $\mathbf{s}_{\mathcal{W}}$ and dilation $\mathbf{d}_{\mathcal{B}} = \text{Shape}(\mathcal{B})$. Note that we perform multiple 2D convolutions along the first dimension of size F_2 using the same weight kernel $\mathcal{A}$ */
$\mathcal{Y}'' \leftarrow \text{BatchConv2d}(\mathcal{A}, \mathcal{Y}') \in \mathbb{R}^{F_2 \times F_1 \times W \times H}$;
$\mathcal{Y} \leftarrow \text{Reshape}(\mathcal{Y}'') \in \mathbb{R}^{F_1 F_2 \times W \times H}$;

Applying Lemma 1 to the summation in (14) yields

$$\mathcal{Y}_{f,x,y} = \sum_{i_1,i_2} \sum_{j_1,j_2} \sum_{c_1,c_2} \mathcal{A}_{f_1,c_1,i_1,j_1} \mathcal{B}_{f_2,c_2,i_2,j_2} \mathcal{X}_{g(c_1,c_2),g(i_1,i_2)+x,g(j_1,j_2)+y},$$

where indices i_1, j_1, c_1 enumerate over elements in tensor $\mathcal{A}$ and i_2, j_2, c_2 enumerate over elements in tensor $\mathcal{B}$. Finally, we can separate the convolution operation into two steps by exchanging the order of summation as follows:

$$\mathcal{Y}_{f,x,y} = \sum_{i_1,j_1,c_1} \mathcal{A}_{f_1,c_1,i_1,j_1} \sum_{i_2,j_2,c_2} \mathcal{B}_{f_2,c_2,i_2,j_2} \mathcal{X}_{g(c_1,c_2),g(i_1,i_2)+x,g(j_1,j_2)+y}. \quad (15)$$

The inner summation in (15) corresponds to a 3D convolution and the outer summation corresponds to *multiple* 2D convolutions, as visualized in Fig. 3 for the special case of $F = 1$.

Overall, (15) can be carried out efficiently in tensor form using Algorithm 1. Effectively, the input is collapsed in two stages instead of one as in the multidimensional convolution operation. Convolving a multi-channel input with a single filter in $\mathcal{W}$ yields a scalar value at a particular output

location. This is done by first scaling all elements in the corresponding multidimensional patch, then collapsing it by means of summation. Since tensor $\boldsymbol{\mathcal{W}}$ is comprised of multidimensional patches $\boldsymbol{\mathcal{B}}$ scaled by elements in $\boldsymbol{\mathcal{A}}$, we can equivalently collapse each *sub-patch* in the input using tensor $\boldsymbol{\mathcal{B}}$ followed by a subsequent collapsing using tensor $\boldsymbol{\mathcal{A}}$ to obtain the same scalar value.

Complexity of KroneckerConvolution

The GKPD of a convolution layer is not unique. Different configurations of Kronecker factors will lead to different reductions in memory and number of operations. Namely, for a KroneckerConvolution layer using $\widehat{R}$ Kronecker products with factors $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{F_1 \times C_1 \times K_{w1} \times K_{h1}}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{F_2 \times C_2 \times K_{w2} \times K_{h2}}$ the memory reduction is

$$\frac{F_1 C_1 K_{w1} K_{h1} F_2 C_2 K_{w2} K_{h2}}{\widehat{R}(F_1 C_1 K_{w1} K_{h1} + F_2 C_2 K_{w2} K_{h2})}, \tag{16}$$

whereas the reduction in FLOPs is

$$\frac{F_1 C_1 K_{w1} K_{h1} F_2 C_2 K_{w2} K_{h2}}{\widehat{R}(F_2 \cdot F_1 C_1 K_{w1} K_{h1} + C_1 \cdot F_2 C_2 K_{w2} K_{h2})}. \tag{17}$$

For the special case of using separable 3×3 filters, and $\widehat{R} = 1$ the reduction in FLOPs becomes

$$\frac{3 F_1 C_2}{F_1 + C_2}, \tag{18}$$

implying that F_1 and C_2 should be sufficiently large in order to obtain a reduction in FLOPs. In contrast, memory reduction is unconditional in the KroneckerConvolution layer.

Universal Approximation via Kronecker Products

Universal approximation applied to shallow networks have been around for a long time (Hornik 1991),(Ripley 1996, pp 173–180) whilst such studies for deep networks are more recent (Zhou 2020). In this section, we build off of these foundations to show that neural networks with weight tensors represented using low Kronecker rank summations of Kronecker products, remain universal approximators. For brevity, we refer to such networks as "Kronecker networks". First, we show that a shallow Kronecker network is a universal approximator. For simplicity, this is shown only for one output. Then, we can generalize the resulting approximator via treating each output dimension separately.

Consider a single layer neural network constructed using n hidden units and an L-Lipschitz activation function $a(\cdot)$

$$\hat{f}_{\mathbf{W}}(x) \triangleq \mathbf{w}_2^\top a\left(\mathbf{W}\mathbf{x}\right) = \sum_{j=1}^{n} w_{2j} a\left(\mathbf{w}_{1j}^\top \mathbf{x} + w_{0j}\right),$$

that is defined on a compacta K in $\mathbb{R}^d$. As shown in (Hornik 1991), such a network serves as a universal approximator, i.e., for a given positive number ϵ there exists an n such that

$$\left\| f - \hat{f}_{\mathbf{W}} \right\|_{2,\mu}^2 \triangleq \int_K \left| f(\mathbf{x}) - \hat{f}_{\mathbf{W}}(\mathbf{x}) \right|^2 d\mu \leq \epsilon. \tag{19}$$

Similarly, a shallow Kronecker network consisting of n hidden units

$$\hat{f}_{\mathbf{W}_{\widehat{R}}}(x) \triangleq \mathbf{w}_2^\top a\left(\mathbf{W}_{\widehat{R}}\mathbf{x}\right),\ \mathbf{W}_{\widehat{R}} = \sum_{r=1}^{\widehat{R}} \mathbf{A}_r \otimes \mathbf{B}_r, \tag{20}$$

is comprised of a weight matrix $\mathbf{W}_{\widehat{R}}$ made of a summation of Kronecker products between factors $\mathbf{A}_r \in \mathbb{R}^{a_1 \times a_2}$ and $\mathbf{B}_r \in \mathbb{R}^{b_1 \times b_2}$. From (20), we can see that any shallow neural network with n hidden units can be represented exactly using a Kronecker network with a full Kronecker rank $R = \min(a_1 a_2, b_1 b_2)$. Thus, shallow Kronecker networks with full Kronecker rank also serve as universal approximators. In Theorem 1 we show that a similar result holds for shallow Kronecker networks $f_{\mathbf{W}_{\widehat{R}}}$, with low Kronecker ranks $\widehat{R} < R$, provided that the $R - \widehat{R}$ smallest singular values of the reshaped matrix $R_w(\mathbf{W})$ of the approximating neural network $\hat{f}_{\mathbf{W}}$ are small enough.

Theorem 1. *Any shallow Kronecker network with a low Kronecker rank $\widehat{R}$ and n hidden units defined on a compacta $K \subset \mathbb{R}^d$ with L-Lipschitz activation is dense in the class of continuous functions $C(K)$ for a large enough n given*

$$\sum_{r=\hat{R}+1}^{R} \sigma_r^2 < \epsilon (L \left\|K\right\|^2 \left\|\mathbf{w}_2\right\|^2)^{-1},$$

where σ_r is the r^{th} singular value of the reshaped version of the weight matrix $R_w(\mathbf{W})$, in an approximating neural network $\hat{f}_{\mathbf{W}}$ with n hidden units satisfying $\|f - \hat{f}_{\mathbf{W}}\|_{2,\mu}^2 < \epsilon$, for $f \in C(K)$.

Proof. See Supplementary Material. □

In Theorem 2, we extend the preceding result to deep convolutional neural networks, where each convolution tensor is represented using a summation of Kronecker products between factor tensors.

Theorem 2. *Any deep Kronecker convolution network with Kronecker rank $\hat{R}_j$ in layer j on compacta $K \subset \mathbb{R}^d$ with L-Lipschitz activation, is dense in the class of continuous functions $C(K)$ for a large enough number of layers J, given*

$$\prod_{j=1}^{J} \left(\sum_{r=\hat{R}_j+1}^{R_j} \sigma_{r,j}^2 \right) < \epsilon (L^J \left\|\mathbf{w}_2\right\|^2 \left\|K\right\|^2)^{-1},$$

where $\sigma_{r,j}$ is the r^{th} singular value of the matrix $R_w(\boldsymbol{\mathcal{W}}^j)$ of the reshaped weight tensor in the j^{th} layer of an approximating convolutional neural network.

Proof. See Supplementary Material. □

The result is achieved by extending the recent universal approximation bound (Zhou 2020) for the GKPD networks. One can derive the convergence rates using (Zhou 2020, Theorem 2) as well. These results assure that the performance degradation of Kronecker networks is small, in comparison to uncompressed networks, for an appropriate choice of Kronecker rank $\widehat{R}$.

Configuration Setting

As GKPD provides us with a set of possible decompositions for each layer in a network, a selection strategy is needed. For a given compression rate, there is a trade-off between using a larger number of terms $\widehat{R}$ in the GKPD summation (11) together with a more compressed configuration and a smaller $\widehat{R}$ with a less compressed configuration. To guide our search, we select the decomposition that best approximates the original uncompressed tensor obtained from a pretrained network. This means different layers in a network will be approximated by a different number of Kronecker products. Before searching for the best decomposition, we limit our search space to configurations that satisfy a desired reduction in FLOPs. Unless otherwise stated all GKPD experiments use this approach.

Experiments

To validate our method, we provide model compression experimental results for image classification tasks using a variety of popular CNN architectures such as ResNet (He et al. 2016), and SEResNet which benefits from the squeeze-and-excitation blocks (Hu, Shen, and Sun 2018). We also choose to apply our compression method on MobileNetV2 (Sandler et al. 2018) as a model that is optimized for efficient inference on embedded vision applications through depthwise separable convolutions and inverted residual blocks. We provide implementation details in the Supplementary Material.

Table 1 shows the top-1 accuracy on the CIFAR-10 (Krizhevsky 2009) dataset using compressed ResNet18 and SEResNet50. For each architecture, the compressed models obtained using the proposed GKPD are named with the "Kronecker" prefix added to the original model's name. The configuration of each compressed model is selected such that the number of parameters is similar to MobileNetV2. We observe that for ResNet18 and SEResNet50, the number of parameters and FLOPs can be highly lowered at the expense of a small decrease in accuracy. Specifically, KroneckerResNet18 achieves a compression of 5× and a 4.7× reduction in FLOPs with only 0.08% drop in accuracy. KroneckerSEResNet50 obtains a compression rate of 9.3× and a 9.7× reduction in FLOPs with only 0.7% drop in accuracy. Moreover, we see that applying the proposed GKPD method on higher-capacity architectures such as ResNet18 and SEResNet50 can lead to higher accuracy than a hand-crafted efficient network such as MobileNetV2. Specifically, with the same number of parameters as that of MobileNetV2, we achieve a compressed ResNet18 (KroneckerResNet18) and a compressed SEResNet50 (KroneckerSEResNet50) with 0.80% and 0.27% higher accuracy than MobileNetV2.

Table 2 shows the performance of GKPD when used to achieve extreme compression rates. The same baseline architectures are compressed using different configurations. We also use GKPD to compress the already efficient MobileNetV2. When targeting very small models (e.g., 0.29M parameters) compressing MobileNetV2 with a compression factor of 7.9× outperforms extreme compression of SEResNet50 with a compression factor of 73.79×.

Model	Params (M)	FLOPs (M)	Accuracy(%)
MobileNetV2 (Baseline)	2.30	96	94.18
ResNet18 (Baseline)	11.17	557	95.05
KroneckerResNet18	2.2	117	94.97
SEResNet50 (Baseline)	21.40	1163	95.15
KroneckerSeResNet50	2.30	120	94.45

Table 1: Top-1 accuracy measured on CIFAR-10 for the baseline models MobileNetV2, ResNet18 and SEResNet as well their compressed versions using GKPD. The number of parameters in compressed models are approximately matched with that of MobileNetV2.

Model	Params (M)	Compression	Accuracy(%)
KroneckerResNet18	0.48	23.27×	92.62
KroneckerSeResNet50	0.93	23.01×	93.66
KroneckerSeResNet50	0.29	73.79×	91.85
KroneckerMobileNetV2	0.73	3.15×	93.80
KroneckerMobileNetV2	0.29	7.90×	93.01
KroneckerMobileNetV2	0.18	12.78×	91.48

Table 2: Top-1 accuracy measured on CIFAR-10 highly compressed ResNet18 (He et al. 2016), MobileNetV2 (Sandler et al. 2018) and SEResNet (Hu, Shen, and Sun 2018).

In the following subsections, we present comparative assessments using different model compression methods.

Comparison with Decomposition-based Methods

In this section, we compare GKPD to other tensor decomposition compression methods. We use a classification model pretrained on CIFAR-10 and apply model compression methods based on Tucker (Kim et al. 2016), Tensor-Train (Garipov et al. 2016), and Tensor-Ring (Wang et al. 2018), along with our proposed GKPD method. We choose ResNet32 architecture in this set of experiments since it has been reported to be effectively compressed using Tensor-Ring in (Wang et al. 2018).

The model compression results obtained using different decomposition methods aiming for a 5× compression rate are shown in Table 3. As this table suggests, GKPD outperforms all other decomposition methods for a similar compression factor. We attribute the performance of GKPD to its higher representation power. This is reflected in its ability to better reconstruct weight tensors in a pretrained network in comparison to other decomposition methods. Refer to Supplementary Material for a comparative assessment of reconstruction errors for different layers of the ResNet architecture.

Comparison with other Compression Methods

We compare our proposed model compression method with two state-of-the-art KD-based compression methods; (Mirzadeh et al. 2020) and (Heo et al. 2019). These methods are known to be very effective on relatively smaller networks such as ResNet26. Thus, we perform our compression

Model	Params (M)	Compression	Accuracy (%)
Resnet32	0.46	1×	92.55
TuckerResNet32	0.09	5×	87.7
TensorTrainResNet32	0.096	4.8×	88.3
TensorRingResNet32	0.09	5×	90.6
KroneckerResNet32	0.09	5×	**91.52**

Table 3: Top-1 Accuracy on CIFAR-10 of compressed ResNet32 models using various decomposition approaches.

Model	Params (M)	Compression	Accuracy (%)
ResNet26	0.37	1×	92.94
Mirzadeh et al. (2020)	0.17	2.13×	91.23
Heo et al. (2019)	0.17	2.13×	90.34
KroneckerResNet26	0.14	2.69×	**93.16**
Mirzadeh et al. (2020)	0.075	4.88×	88.0
Heo et al. (2019)	0.075	4.88×	87.32
KroneckerResNet26	0.069	5.29×	**91.28**

Table 4: Top-1 accuracy measured on CIFAR-10 for the baseline model ResNet26 and its compressed versions obtained using the KD-based methods; (Mirzadeh et al. 2020), (Heo et al. 2019), and the proposed GKPD method.

method on ResNet26 architecture in these experiments. Table 4 presents the top-1 accuracy obtained for different compressed models with two different compression rates. As this table suggests, the proposed method results in greater than 2% and 3.7% improvements in top-1 accuracy once we aim for compression rates of $\sim 2\times$ and $\sim 5\times$, respectively, compared to using the KD-based model compression methods.

Model Compression with Random Initialization

To study the effect of replacing weight tensors in neural networks with a summation of Kronecker products, we conduct experiments using randomly initialized Kronecker factors as opposed to performing GKPD on a pretrained network. By replacing all weight tensors in a predefined network architecture with a randomly initialized summation of Kronecker products, we obtain a compressed model. To this end, we run assessments on a higher capacity architecture i.e, ResNet50 on a larger scale dataset i.e, ImageNet (Krizhevsky, Sutskever, and Hinton 2012). Table 5 lists the top-1 accuracy for ResNet50 baseline and its compressed variation. We achieve a compression rate of $2.13\times$ with a 2% accuracy drop compared to the baseline model.

We also perform model compression using two state-of-the-art model compression methods; ThiNet (Luo, Wu, and Lin 2017) and FSNet (Yang et al. 2020). ThiNet and FSNet are based on pruning and filter sharing techniques, respectively. They both reportedly, lead to a good accuracy on large datasets. Table 5 also lists the top-1 accuracy for ResNet50 compressed using these two methods. As the table shows, our proposed method outperforms the other two techniques for a $\sim 2\times$ compression rate. Note that the performance obtained using our method is based on a random initialization, while the compression achieved with ThiNet benefits from

Model	Params (M)	Compression	Accuracy (%)
ResNet50	25.6	1×	75.99
FSNet	13.9	2.0×	73.11
ThiNet	12.38	2.0×	71.01
KroneckerResNet50	12.0	2.13×	**73.95**

Table 5: Top-1 accuracy measured on ImageNet for the baseline model ResNet50 and its compressed versions obtained using ThiNet (Luo, Wu, and Lin 2017), FSNet (Yang et al. 2020), and the proposed GKPD method.

Model	Params (M)	FLOPs (M)	Accuracy(%)
ResNet18	11.17	0.58	95.05
KroneckerResNet18 ($\widehat{R}=4$)	1.41	0.17	92.96
KroneckerResNet18 ($\widehat{R}=8$)	1.42	0.16	93.74
KroneckerResNet18 ($\widehat{R}=16$)	1.44	0.26	94.30
KroneckerResNet18 ($\widehat{R}=32$)	1.51	0.32	94.58

Table 6: Top-1 image classification accuracy of compressed ResNet18 on CIFAR-10, where $\widehat{R}$ denotes the number of Kronecker products used in the GKPD of each layer.

a pretrained model. These results indicate that the proposed GKPD can lead to a high performance even if a pretrained model is not available.

Experimental Analysis of Kronecker Rank

Using a higher Kronecker rank $\widehat{R}$ can increase the representation power of a network. This is reflected by the ability of GKPD to better reconstruct weight tensors using a larger number of Kronecker products in (11). In Table 6 we study the effect of using a larger $\widehat{R}$ in Kronecker networks while keeping the overall number of parameters constant. We find that using a larger $\widehat{R}$ does indeed improve performance.

Conclusion

In this paper we propose GKPD, a generalization of Kronecker Product Decomposition to multidimensional tensors for compression of deep CNNs. In the proposed GKPD, we extend the Nearest Kronecker Product problem to the multidimensional setting and use it for optimal initialization from a baseline model. We show that for a fixed number of parameters, using a summation of Kronecker products can significantly increase the representation power in comparison to a single Kronecker product. We use our approach to compress a variety of CNN architectures and show the superiority of GKPD to some state-of-the-art compression methods. GKPD can be combined with other compression methods like quantization and knowledge distillation to further improve the compression-accuracy trade-off. Designing new architectures that can benefit most from Kronecker product representation is an area for future work.

Acknowledgments

The authors thank Ali Ghodsi and Guillaume Rabusseau for useful discussions and suggestions.

References

Alvarez, J. M.; and Salzmann, M. 2016. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, 2270–2278.

Alvarez, J. M.; and Salzmann, M. 2017. Compression-aware training of deep networks. *Advances in neural information processing systems*, 30: 856–867.

Boulch, A. 2018. Reducing parameter number in residual networks by sharing weights. *Pattern Recognition Letters*, 103: 53–59.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, 3123–3131.

Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*.

Denil, M.; Shakibi, B.; Dinh, L.; Ranzato, M.; and de Freitas, N. 2013. Predicting Parameters in Deep Learning. In *Advances in Neural Information Processing Systems*, 2148–2156.

Denton, E. L.; Zaremba, W.; Bruna, J.; LeCun, Y.; and Fergus, R. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, 1269–1277.

Feichtenhofer, C.; Fan, H.; Malik, J.; and He, K. 2019. Slowfast networks for video recognition. In *IEEE/CVF International Conference on Computer Vision*, 6202–6211.

Garipov, T.; Podoprikhin, D.; Novikov, A.; and Vetrov, D. 2016. Ultimate tensorization: compressing convolutional and FC layers alike. *arXiv preprint arXiv:1611.03214*.

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

Han, S.; Pool, J.; Tran, J.; and Dally, W. J. 2015. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Heo, B.; Lee, M.; Yun, S.; and Choi, J. Y. 2019. Knowledge Distillation with Adversarial Samples Supporting Decision Boundary. In *AAAI Conference on Artificial Intelligence*, 3771–3778. AAAI Press.

Hornik, K. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2): 251–257.

Hu, J.; Shen, L.; and Sun, G. 2018. Squeeze-and-excitation networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 7132–7141.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1): 6869–6898.

Iandola, F. N.; Han, S.; Moskewicz, M. W.; Ashraf, K.; Dally, W. J.; and Keutzer, K. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size. *arXiv preprint arXiv:1602.07360*.

Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; and Kalenichenko, D. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2704–2713.

Kim, Y.; Park, E.; Yoo, S.; Choi, T.; Yang, L.; and Shin, D. 2016. Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications. In *International Conference on Learning Representations*.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images. Technical report.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, 1106–1114.

Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I. V.; and Lempitsky, V. S. 2015. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition. In *International Conference on Learning Representations*.

Li, F.; Zhang, B.; and Liu, B. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711*.

Li, J.; Qi, Q.; Wang, J.; Ge, C.; Li, Y.; Yue, Z.; and Sun, H. 2019. OICSR: Out-in-channel sparsity regularization for compact deep neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7046–7055.

Liu, B.; Wang, M.; Foroosh, H.; Tappen, M.; and Pensky, M. 2015. Sparse convolutional neural networks. In *IEEE conference on computer vision and pattern recognition*, 806–814.

Luo, J.; Wu, J.; and Lin, W. 2017. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. In *IEEE International Conference on Computer Vision*, 5068–5076.

Mirzadeh, S.; Farajtabar, M.; Li, A.; Levine, N.; Matsukawa, A.; and Ghasemzadeh, H. 2020. Improved Knowledge Distillation via Teacher Assistant. In *AAAI Conference on Artificial Intelligence*, 5191–5198.

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

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, 525–542.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28: 91–99.

Ripley, B. D. 1996. *Pattern recognition and neural networks*. Cambridge university press.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE conference on computer vision and pattern recognition*, 4510–4520.

Tang, W.; Hua, G.; and Wang, L. 2017. How to train a compact binary neural network with high accuracy? In *AAAI conference on artificial intelligence*.

Thakker, U.; Beu, J.; Gope, D.; Zhou, C.; Fedorov, I.; Dasika, G.; and Mattina, M. 2019. Compressing RNNS for IoT devices by 15-38x using Kronecker products. *arXiv preprint arXiv:1906.02876*.

Tucker, L. R. 1963. Implications of factor analysis of three-way matrices for measurement of change. *Problems in measuring change*, 15(122-137): 3.

Van Loan, C.; and Pitsianis, N. 1992. *Approximation with Kronecker products*, 293–314.

Van Loan, C. F. 2000. The ubiquitous Kronecker product. *Journal of computational and applied mathematics*, 123(1-2): 85–100.

Wang, W.; Aggarwal, V.; and Aeron, S. 2017. Efficient low rank tensor ring completion. In *IEEE International Conference on Computer Vision*, 5697–5705.

Wang, W.; Sun, Y.; Eriksson, B.; Wang, W.; and Aggarwal, V. 2018. Wide Compression: Tensor Ring Nets. In *IEEE Conference on Computer Vision and Pattern Recognition*, 9329–9338.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29: 2074–2082.

Yang, Y.; Yu, J.; Jojic, N.; Huan, J.; and Huang, T. S. 2020. FSNet: Compression of Deep Convolutional Neural Networks by Filter Summary. In *International Conference on Learning Representations*.

Zhang, D.; Wang, H.; Figueiredo, M.; and Balzano, L. 2018a. Learning to share: Simultaneous parameter tying and sparsification in deep learning. In *International Conference on Learning Representations*.

Zhang, D.; Yang, J.; Ye, D.; and Hua, G. 2018b. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *European conference on computer vision*, 365–382.

Zhou, D.-X. 2020. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2): 787–794.

Zhou, H.; Alvarez, J. M.; and Porikli, F. 2016. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, 662–677. Springer.

Zhou, S.; Wu, J.-N.; Wu, Y.; and Zhou, X. 2015. Exploiting local structures with the kronecker layer in convolutional networks. *arXiv preprint arXiv:1512.09194*.

Zhu, C.; Han, S.; Mao, H.; and Dally, W. J. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*.