Model Compression via Generalized Kronecker Product Decomposition

by

Marawan Gamal

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master Of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis is based on the previously published work listed below, where I was a major contributor to the conceptualization, implementation, experimentation and writing.

Marawan Gamal Abdel Hameed, Marzieh S. Tahaei, Ali Mosleh, and Vahid Partovi Nia. Convolutional neural network compression through generalized kronecker product decomposition. In Proceedings of the AAAI Conference on Artificial Intelligence, 2022,

Acknowledgements

I would like to thank my supervisors Prof. David Clausi and Prof. John Zelek for their guidance and support throughout my degree. Secondly, I would also like to thank Prof. Bryan Tripp and Prof. Sirisha Rambhatla for their valuable feedback on my work.

Additionally, I would like to thank my co-authors, Marzieh S. Tahaei for guiding my work at early stage as well as Ali Mosleh and Vahid Partovi Nia.

Lastly, I would like to thank my parents for supporting my path into research, as well as my colleagues at the VIP lab for their help throughout my masters.

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 practice, this limits many real-world applications by requiring them to off-load computations to cloud-based systems. Such a limitation introduces concerns related to privacy as well as bandwidth capabilities. The design of efficient models as well as automated compression methodologies such as quantization, pruning, knowledge distillation and tensor decomposition have been proposed to allow models to operate in such resource-constrained environments. In particular, tensor decomposition approaches have gained interest in recent years as they can achieve a wide variety of compression rates while maintaining efficient memory access patterns. However, they typically cause significant reduction in model performance on classification tasks after compression.

To address this challenge, a new method that improves performance of decomposition-based model compression has been designed and tested on a variety of classification tasks. Specifically, we compress convolutional 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 to simultaneously reduce its memory usage and number of floating-point-operations. Experimental results for image classification on CIFAR-10 and ImageNet datasets using ResNet, MobileNetv2 and SeNet architectures as well as action recognition on HMDB-51 using I3D-ResNet50 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.

The proposed GKPD method serves as a means of deploying state-of-the-art CNN models without sacrificing significant accuracy degradation. Furthermore, the capability of utilizing GKPD as a drop-in replacement for convolutional layers allows its use for CNN model compression with minimal development time, in contrast to approaches such as efficient architecture design.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Convolutional neural networks (CNNs) have achieved state-of-the-art performance on a wide range of computer vision tasks such as image classification [1, 2, 3], action recognition [4, 5, 6] and object detection [7, 8]. Despite achieving remarkably low generalization errors, modern CNN architectures are typically over-parameterized, consisting of millions of parameters. As the size of state-of-the-art CNN architectures continues to grow, deploying these models on resource constrained edge devices becomes more challenging. Motivated by studies demonstrating that there is significant redundancy in CNN parameters [9], 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, when compressing DNN parameters. Namely, Tucker [10], CP [11], Tensor-Train [12, 13], and Tensor-Ring [14] decompositions have been widely studied for DNNs. However, these methods still suffer significant accuracy loss for computer vision tasks.

Kronecker product decomposition (KPD) is another decomposition method that has recently shown to be very effective when applied to RNNs [15]. 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 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.

Applying GKPD to an arbitrary tensor leads to multiple possible decompositions, one for each configuration of Kronecker factors. In Figure 1.1, we plot reconstruction errors using different configurations, of the tensor in the first layer of a ResNet18 model [1], pre-trained on ImageNet [16]. As shown in this figure, 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 following are the main contributions of this thesis.

- Providing a solution to the multidimensional nearest Kronecker product problem and providing a solution using SVD, leading to the *Generalized Kronecker Product Decomposition* (GKPD).
- Improving the state-of-the art for compression of image classification models on CIFAR-10 and ImageNet using compressed ResNet [1], MobileNetv2 [17] and SeNet [18] architectures as well as action recognition on HMDB-51 using I3D-ResNet50 [6].

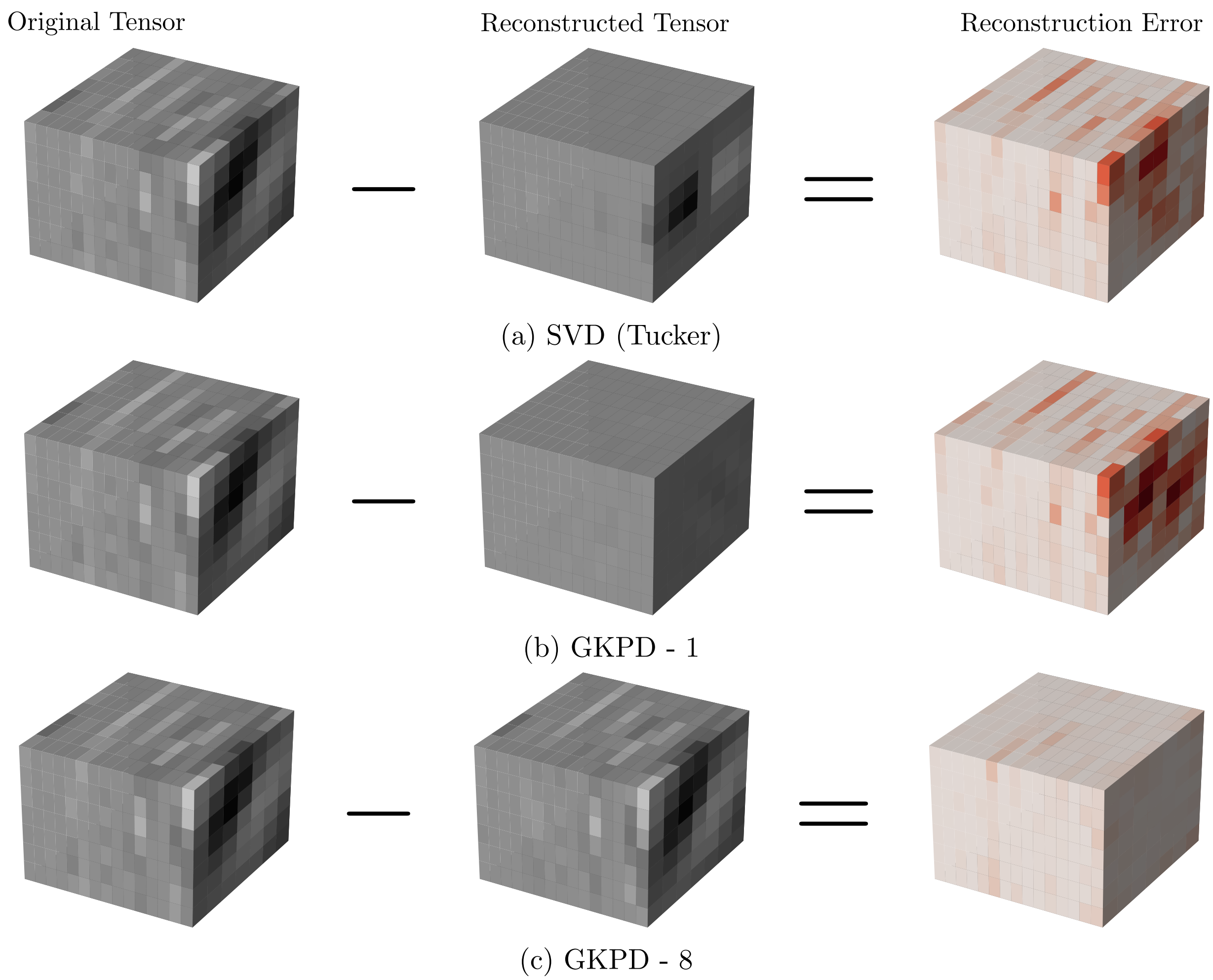


Figure 1.1: A compression rate of 2× achieved for an arbitrary tensor from the first layer of ResNet18 using SVD (Tucker) in (a), and the proposed GKPD in (b) and (c). A larger summation, GKPD-8 achieves a lower reconstruction error in comparison with both a smaller summation, GKPD-1, as well as SVD (Tucker) decomposition.

Chapter 2

Background

The past few years have seen an increasing trend in the computational burden presented by DNN models. Thus, inspiring the development of a variety of efficient architecture designs such as MobileNetv2 [17], ShuffleNet [19] and Xception [20] as well as efficient models found by means of architecture search such as in EfficientNet [21], MobileNetv3 [22] and ProxylessNAS [23]. While approaches such as Xception [20] and EfficientNet [21] find suitable models based on design criteria (such as model size, depth and width), approaches that find models suitable for particular hardware have also been proposed [24, 23, 25].

On the other hand, there have been a variety of approaches proposed to compress existent models, including quantization, pruning, knowledge distillation and tensor decomposition. Below, we provide a summary of some notable methods falling under these categories.

Quantization methods involve reducing the precision of parameters and activations into lower-bit representations, such as in [26] which quantizes 32-bit parameters to 8-bit representations. Some works have pushed this further by quantizing model parameters to binary [27, 28, 29, 30] and ternary [31, 32] representations. An important success of quantization methods is their ability to lead to speedups in DNN training [30]. However, it has been shown to be challenging to go below half-precision while maintaining a similar level of performance [33].

Pruning methods can be further categorized into unstructured [34, 26, 35] and structured [36, 37, 38]. The former has the potential to lead to a large number of pruned parameters with minimal loss in generalization performance. However, in practice structured methods are preferred as unstructured methods do not lead to expected model acceleration. This is due to the irregularity of the imposed sparsity constraints, leading to sparse matrix operations that are challenging to accelerate [39].

Knowledge distillation approaches commonly compress a large (teacher) network by using it to train a smaller (student) network [40]. These approachess are inspired by the observation that large models tend to better extract structure from large redundant datasets, whereas once this structure is extracted it can be more efficiently represented using a smaller model [41]. This concept has been further extended in approaches such as the one proposed by Mirzadeh et al [42] which uses intermediate teacher assistant networks, and Heo et al [43] which favours the transfer of decision boundary information.

Low-rank approximations rely on representing large matrices or tensors using a set of factors. This approach typically results in a reduced number of parameters and computations. One of the first works in this area began by applying truncated singular value decomposition (SVD) to convolution tensors reshaped to matrices [44]. This inspired others to apply tensor decomposition approaches to DNN model compression, such as Canonical Polyadic (CP) [11], Tucker [10] Tensor-Train (TT) [12] and Tensor Ring (TR) [14]. On the other hand, there has been a closely related line of work focusing on representing convolution filters using low-rank bases [45] and initializing these bases to minimize either the filter reconstruction error or projection error. This approach was extended in [46] to allow for a varying bases size and to further improve compression results. In a similar vein, sharing filter weights across spatial locations was proposed by FSNet [47]. Representing matrices and tensors using a Kronecker factorization was also explored by Zhou et al [48], but was limited to rank-1 approximations that were randomly initialized. As shown in Figure 1.1 and in the next sections of this thesis, using a larger summation of Kronecker products can significantly improve the representation power of a network and thus leads to a performance increase.

Kronecker factorizations have also been used to approximate the Fisher information matrix (FIM) when training DNN models using second-order optimization methods [49, 50]. Specifically, approximations for models using fully-connected layers [49] as well as convolutional layers [50] have been derived. This line of work targets accelerated training, by leveraging curvature information to provide better model parameter updates. The more informed parameter updates will often lead to a reduction in the number of steps required for convergence. Such approximations have been further improved in [51] by recognizing the low-rank nature of the Kronecker factors that emerges when using mini-batches to approximate true statistics, leading to less time spent inverting the FIM approximation.

Chapter 3

Generalized Kronecker Product Decomposition

In this chapter, we introduce the proposed Generalized Kronecker Product Decomposition (GKPD). We start by providing background on the Kronecker Product Decomposition in Section 3.1, then present its generalization to multi-way tensors in Section 3.2. Furthermore we derive an algorithm for performing convolution operations using GKPD factorized tensors in Section 3.3.

3.1 Preliminaries

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

$$A \otimes B \triangleq \begin{pmatrix} a_{11}B & \dots & a_{1n_1}B \\ \vdots & \ddots & \vdots \\ a_{m_1 1}B & \dots & a_{m_1 n_1}B \end{pmatrix}. \tag{3.1}$$

As shown by Loan et al [52], any matrix $W \in \mathbb{R}^{m_1 m_2 \times n_1 n_2}$ can be decomposed into a sum of Kronecker products as

$$W = \sum_{r=1}^{R} A_r \otimes B_r, \tag{3.2}$$

where

$$R = \min(m_1 n_1, m_2 n_2) \tag{3.3}$$

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

To compress a given matrix W, we can represent it using a small number $\widehat{R} < R$ of Kronecker products that best approximate the original tensor. The best factors are found by solving the *Nearest Kronecker Product* problem [53]

$$\min_{\{A_r\},\{B_r\}} \left\| W - \sum_{r=1}^{\widehat{R}} A_r \otimes B_r \right\|_{\mathrm{F}}^2 . \tag{3.4}$$

where $\|\|_{\mathrm{F}}$ denotes the Frobenius norm. As this approximation replaces a large matrix with a sequence of two smaller ones, memory consumption is reduced by a factor of

$$\frac{m_1 m_2 n_1 n_2}{\widehat{R}(m_1 n_1 + m_2 n_2)}. \tag{3.5}$$

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

$$\mathbf{y} = (A \otimes B)\mathbf{x} \equiv Y = BXA^\top, \tag{3.6}$$

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

3.2 Generalized Kronecker Product Decomposition

We now turn to generalizing the Kronecker product to operate on tensors. Let $\mathcal{A} \in \mathbb{R}^{a_1 \times \cdots \times a_N}$ and $\mathcal{B} \in \mathbb{R}^{b_1 \times \cdots \times b_N}$ be two given tensors. Intuitively, tensor $(\mathcal{A} \otimes \mathcal{B}) \in \mathbb{R}^{a_1 b_1 \times \cdots \times a_N b_N}$ is constructed by *moving around* tensor $\mathcal{B}$ in a non-overlapping fashion, and at each position scaling it by a corresponding element of $\mathcal{A}$ as shown in Figure 3.1. Formally, the multidimensional Kronecker product [48] is defined as follows

$$(\mathcal{A} \otimes \mathcal{B})_{i_1 \cdots i_N} \triangleq \mathcal{A}_{j_1 \cdots j_N} \mathcal{B}_{k_1 \cdots k_N}, \tag{3.7}$$

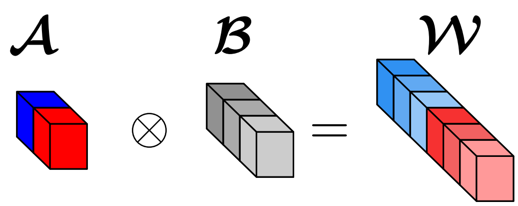


Figure 3.1: Illustration of Kronecker decomposition of a single convolution filter (with spatial dimensions equal to one for simplicity).

where

$$j_n = \left\lfloor \frac{i_n}{b_n} \right\rfloor \text{ and } \; k_n = i_n \bmod b_n \tag{3.8}$$

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

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

$$\mathcal{W} = \sum_{r=1}^{R} \mathcal{A}_r \otimes \mathcal{B}_r, \tag{3.9}$$

where

$$R = \min(a_1 a_2 \cdots a_N, b_1 b_2 \cdots b_N) \tag{3.10}$$

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

$$\min_{\{\mathcal{A}_r\},\{\mathcal{B}_r\}} \left\| \mathcal{W} - \sum_{r=1}^{\widehat{R}} \mathcal{A}_r \otimes \mathcal{B}_r \right\|_{\mathrm{F}}^2, \tag{3.11}$$

where $\widehat{R} < R$. For the case of matrices (2D tensors), [53] solved this problem using SVD. We extend their approach to the multidimensional setting. Our strategy will be to define reshaping operators

$$R_W : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{N_p \times d'_1 \cdots d'_N} \tag{3.12}$$

$$\mathbf{r}_a : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{d_1 \cdots d_N} \tag{3.13}$$

$$\mathbf{r}_b : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{d_1 \cdots d_N} \tag{3.14}$$

and solve

$$\min_{\{\mathcal{A}_r\},\{\mathcal{B}_r\}} \left\| R_W(\mathcal{W}) - \sum_{r=1}^{\widehat{R}} \mathbf{r}_a(\mathcal{A}_r)\mathbf{r}_b(\mathcal{B}_r)^\top \right\|_{\mathrm{F}}^2 \tag{3.15}$$

instead. By carefully defining the reshaping operators, the sum of squares in (3.15) is kept identical to that in (3.11). The former corresponds to finding the best low-rank approximation, which has a well known solution using SVD. We define the reshaping operators as follows:

$$R_W(\mathcal{W}) = \textsc{Mat}(\textsc{Unfold}(\mathcal{W}, \mathbf{d}^{(\mathcal{B})})) \tag{3.16}$$

$$\mathbf{r}_a(\mathcal{A}) = \textsc{Unfold}(\mathcal{A},\, \mathbf{d}^{(\mathcal{I}_\mathcal{A})}) \tag{3.17}$$

$$\mathbf{r}_b(\mathcal{B}) = \textsc{Vec}(\mathcal{B}) \tag{3.18}$$

where

$$\textsc{Unfold}:\ \mathbb{R}^{d_1\times\cdots\times d_N} \to \mathbb{R}^{N_p\times d'_1\times\cdots\times d'_N} \tag{3.19}$$

takes as input a tensor $\mathcal{W}$ and sub-patch shape vector $\mathbf{d}' = (d'_1, \cdots, d'_N)$, then extracts N_p non-overlapping patches of shape $\mathbf{d}'$ from tensor $\mathcal{W}$. Thus tensor $\mathcal{W}$ dimensions must be divisible by $\mathbf{d}'$). The operation

$$\textsc{Vec}:\ \mathbb{R}^{d_1\times\cdots\times d_N} \to \mathbb{R}^{d_1\cdots d_N} \tag{3.20}$$

flattens its input into a vector, and

$$\textsc{Mat}:\ \mathbb{R}^{d_1\times\cdots\times d_N} \to \mathbb{R}^{d_1\times d_2\cdots d_N} \tag{3.21}$$

reshapes a tensor to a matrix. Tensor $\mathcal{I}_\mathcal{A}$ has the same number of dimensions as $\mathcal{A}$ with each dimension equal to unity and $\mathbf{d}^{(\mathcal{B})}$ is a vector describing the shape of tensor $\mathcal{B}$. While the ordering of patch extraction in (3.19) and flattening (3.20) is not important, it must remain consistent across the reshaping operators (see Lemma 1).

Finally, upon conversion of the problem in (3.11) to the low-rank matrix approximation problem (3.15), we can use SVD to select optimal approximating vectors then reshape them back to tensors by applying the inverse of (3.13) and (3.14). We demonstrate the optimality of our approach through the following lemma and theorem, which make use of index mappings defined below.

Definition 1. *(Index mapping)*
A function $g: \mathbb{R}^{g_1^{(a)}\times\cdots\times g_N^{(a)}} \to \mathbb{R}^{g_1^{(b)}\times\cdots\times g_N^{(b)}}$ *satisfying*

$$\prod_{n=1}^{N} g_n^{(a)} = \prod_{n=1}^{N} g_n^{(b)}$$

is called a reshaping operation and induces an index mapping I_g *such that*

$$g(\mathbf{x})_{\mathbf{i}^{(b)}} = \mathbf{x}_{\mathbf{i}^{(a)}}, \quad \mathbf{i}^{(b)} = I_g(\mathbf{i}^{(a)}), \tag{3.22}$$

for any $\mathbf{i}^{(a)} \in \mathbb{N}^{g_1^{(a)} \times \cdots \times g_N^{(a)}}$.

Lemma 1. *(Sum of squares preserving reshapings)*
Let $\mathcal{W} \in \mathbb{R}^{w_1 \times \cdots \times w_N}$, $\mathcal{A} \in \mathbb{R}^{a_1 \times \cdots \times a_N}$ *and* $\mathcal{B} \in \mathbb{R}^{b_1 \times \cdots \times b_N}$. *Then,*

$$\min_{\{\mathcal{A}_r\},\{\mathcal{B}_r\}} \left\| \mathcal{W} - \sum_{r=1}^{\widehat{R}} \mathcal{A}_r \otimes \mathcal{B}_r \right\|_{\mathrm{F}}^2 \equiv \min_{\{\mathcal{A}_r\},\{\mathcal{B}_r\}} \left\| R_W(\mathcal{W}) - \sum_{r=1}^{\widehat{R}} \mathbf{r}_a(\mathcal{A}_r)\mathbf{r}_b(\mathcal{B}_r)^\top \right\|_{\mathrm{F}}^2 \tag{3.23}$$

where,

$$\textsc{Vec} : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{d_1 \cdots d_N} \quad \text{(3.20 revisted)}$$
$$\textsc{Mat} : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{d_1 \times d_2 \cdots d_N} \quad \text{(3.21 revisted)}$$
$$\textsc{Unfold} : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{N_p \times d'_1 \times \cdots \times d'_N} \quad \text{(3.19 revisted)}$$
$$R_W : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{N_p \times d'_1 \cdots d'_N} \quad \text{(3.12 revisted)}$$
$$\mathbf{r}_a : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{d_1 \cdots d_N} \quad \text{(3.13 revisted)}$$
$$\mathbf{r}_b : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{d_1 \cdots d_N} \quad \text{(3.14 revisted)}$$

are reshaping operations with $d_i, d'_i \in \mathbb{N}$ *and index mappings*

$$I_{\textsc{Unfold}}(\mathbf{i}^{(a)}, \mathbf{d}^{(\mathcal{A})}, \mathbf{d}^{(\mathcal{B})}) \triangleq \begin{pmatrix} \left[P^{(u)}\mathbf{c}\right]_{p(\mathbf{i}^{(a)}, \mathbf{d}^{(\mathcal{A})}, \mathbf{d}^{(\mathcal{B})})} \\ \mathbf{i}^{(a)} \bmod \mathbf{d}^{(\mathcal{A})} \end{pmatrix}, \tag{3.24}$$

$$I_{\textsc{Vec}}(\mathbf{i}^{(a)}, \mathbf{d}^{(\mathcal{A})}) \triangleq \left[P^{(v)}\mathbf{c}\right]_{p(\mathbf{i}^{(a)}, \mathbf{d}^{(\mathcal{A})}, \mathbf{d}^{(\mathcal{I}_\mathcal{A})})}, \tag{3.25}$$

$$I_{\textsc{Mat}}(\mathbf{i}^{(a)}, \mathbf{d}^{(\mathcal{A})}) \triangleq \begin{pmatrix} \mathbf{i}_1^{(a)} \\ I_{\textsc{Vec}}(\mathbf{i}_{2:}^{(a)}, \mathbf{d}_{2:}^{(\mathcal{A})}) \end{pmatrix}, \tag{3.26}$$

$$\tag{3.27}$$

where $P^{(u)}$ and $P^{(v)}$ denote permutation matrices, vector $\mathbf{c}$ *contains an integer enumeration starting at one and ending at* $\prod_i \left[\frac{\mathbf{d}^{(\mathcal{A})}}{\mathbf{d}^{(\mathcal{B})}}\right]_i$, mod *denotes an element-wise modulo operation, and*

$$p(\mathbf{i}^{(a)}, \mathbf{d}^{(\mathcal{A})}, \mathbf{d}^{(\mathcal{B})}) = \mathrm{sum}\left(\left\lfloor \frac{\mathbf{i}^{(a)}}{\mathbf{d}^{(\mathcal{B})}} \right\rfloor \times \begin{pmatrix} 1 \\ L\left[\frac{\mathbf{d}^{(\mathcal{A})}}{\mathbf{d}^{(\mathcal{B})}}\right]_{2:} \end{pmatrix}\right) \tag{3.28}$$

with the multiplication, division and floor operations being element-wise.

Theorem 1. *(Optimality of GKPD)*
Any tensor $\mathcal{W} \in \mathbb{R}^{w_1 \times \cdots \times w_N}$ *can be represented exactly as*

$$\mathcal{W} = \sum_{r=1}^{R} \mathcal{A} \otimes \mathcal{B} \tag{3.29}$$

where $\mathcal{A}_r \in \mathbb{R}^{a_1 \times \cdots \times a_N}$, $\mathcal{B}_r \in \mathbb{R}^{b_1 \times \cdots \times b_N}$ *and* $R \in \mathbb{N}$

3.3 GKPD Factorized Convolutions

An N-dimensional convolution operation in CNNs between a weight tensor $\mathcal{W} \in \mathbb{R}^{F \times C \times D_1 \times \cdots D_N}$ and an input $\mathcal{X} \in \mathbb{R}^{C \times S_1 \times \cdots S_N}$ is a multilinear map that can be described in scalar form as

$$\mathcal{Y}_{fd_1 \cdots d_N} = \sum_{c, \delta_1, \cdots, \delta_N} \mathcal{W}_{fc\delta_1 \cdots \delta_N} \mathcal{X}_{c, d_1+\delta_1, \cdots, d_N+\delta_N}. \tag{3.30}$$

Assuming $\mathcal{W}$ can be decomposed to KPD factors

$$\mathcal{A} \in \mathbb{R}^{f^{(a)} \times c^{(a)} \times d_1^{(a)} \times \cdots \times d_N^{(a)}} \quad \text{and} \quad \mathcal{B} \in \mathbb{R}^{f^{(b)} \times c^{(b)} \times d_1^{(b)} \times \cdots \times d_N^{(b)}} \tag{3.31}$$

we can rewrite (3.30) as

$$\mathcal{Y}_{fd_1 \cdots d_N} = \sum_{r, c^{(a)}, c^{(b)}, \delta^{(a)}, \delta^{(b)}} (\mathcal{A} \otimes \mathcal{B})_{fcd_1 \cdots d_N} \mathcal{X}_{c, d_1+\delta_1, \cdots, d_N+\delta_N}. \tag{3.32}$$

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 (3.32). Instead, we can carry out the summation by *directly* using elements of tensors $\mathcal{A}$ and $\mathcal{B}$ as shown in Lemma 2. This key insight leads to a large reduction in both memory and FLOPs. Effectively, this allows us to replace a large convolutional layer (i.e. a convolution with a large weight tensor) with two smaller ones, as we demonstrate in the rest of this section.

Lemma 2. *(Linear Projections with Kronecker factorized tensors) Given tensor* $\mathcal{W} \in \mathbb{R}^{w_1 \times \cdots \times w_N}$ *and its GKPD factors* $\mathcal{A} \in \mathbb{R}^{a_1 \times \cdots \times a_N}$, $\mathcal{B} \in \mathbb{R}^{b_1 \times \cdots \times b_N}$ *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 as follows:*

$$\mathcal{W}_{i_1 \cdots i_N} \mathcal{X}_{i_1 \cdots i_N} = \mathcal{A}_{j_1 \cdots j_N} \mathcal{B}_{k_1 \cdots k_N} \mathcal{X}_{g(j_1, k_1) \cdots g(j_N, k_N)}, \tag{3.33}$$

where $\mathcal{X} \in \mathbb{R}^{d_1 \times \cdots \times d_N}$ *is an input tensor,*

$$g(j_n, k_n) \triangleq j_n b_n + k_n, \tag{3.34}$$

is a re-indexing function; and j_n, k_n *are as defined in* (3.8). *The equality also holds for any valid offsets to the input's indices. That is,*

$$\mathcal{W}_{i_1 \cdots i_N} \mathcal{X}_{i_1 + o_1, \cdots, i_N + o_N} = \mathcal{A}_{j_1 \cdots j_N} \mathcal{B}_{k_1 \cdots k_N} \mathcal{X}_{g(j_1, k_1) + o_1, \cdots, g(j_N, k_N) + o_N}, \tag{3.35}$$

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

Applying Lemma 2 to the summation in (3.32) yields

$$\mathcal{Y}_{f d_1 \cdots d_N} = \sum_{r, c^{(a)}, c^{(b)}, \delta^{(a)}, \delta^{(b)}} \mathcal{A}_{f^{(a)} c^{(a)} \delta_1^{(a)} \cdots \delta_N^{(a)}} \mathcal{B}_{f^{(b)} c^{(b)} \delta_1^{(b)} \cdots \delta_N^{(b)}} \mathcal{X}_{g(c_1, c_2), g(\delta_1^{(a)}, \delta_1^{(b)}) + d_1, \cdots, g(\delta_N^{(a)}, \delta_N^{(b)}) + d_N}, \tag{3.36}$$

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:

$$\mathcal{Y}_{f d_1 \cdots d_N} = \sum_{r, c^{(a)}, \delta^{(a)}} \mathcal{A}_{f^{(a)} c^{(a)} \delta_1^{(a)} \cdots \delta_N^{(a)}} \sum_{c^{(b)}, \delta^{(b)}} \mathcal{B}_{f^{(b)} c^{(b)} \delta_1^{(b)} \cdots \delta_N^{(b)}} \mathcal{X}_{g(c_1, c_2), g(\delta_1^{(a)}, \delta_1^{(b)}) + d_1, \cdots, g(\delta_N^{(a)}, \delta_N^{(b)}) + d_N}, \tag{3.37}$$

Overall, (3.37) can be carried out efficiently in tensor form using Algorithm 1. Effectively, the input is collapsed in two stages instead of one as in a standard multidimensional convolution operation, and can be carried out efficiently using two consectuive N-dimensional convolution operations with intermediate reshapings.

Algorithm 1: KroneckerConvNd

Input:

$\mathcal{A} \in \mathbb{R}^{R\times f^{(a)}\times c^{(a)}\times d_1^{(a)}\times\cdots\times d_N^{(a)}}, \quad \mathcal{B} \in \mathbb{R}^{R\times f^{(b)}\times c^{(b)}\times d_1^{(b)}\times\cdots\times d_N^{(b)}},$
$\mathcal{X} \in \mathbb{R}^{B\times C\times D_1\times\cdots\times D_N}$

Output: $\mathcal{X} \in \mathbb{R}^{B\times F\times D_1\times\cdots\times D_N}$

$\mathcal{X} \leftarrow \textsc{Reshape}(\mathcal{X})$ // $\mathbb{R}^{Bc^{(a)}\times c^{(b)}\times D_1\times\cdots D_N}$

$\mathcal{X} \leftarrow \textsc{ConNd}(\mathcal{B}, \mathcal{X})$ // $\mathbb{R}^{Bc^{(a)}\times f^{(b)}\times D_1\times\cdots D_N}$

$\mathcal{X} \leftarrow \textsc{Reshape}(\mathcal{X})$ // $\mathbb{R}^{Bf^{(b)}\times c^{(a)}\times D_1\times\cdots D_N}$

$\mathcal{X} \leftarrow \textsc{ConvNd}(\mathcal{A}, \mathcal{X}, \text{groups} = R)$ // $\mathbb{R}^{Bf^{(b)}\times f^{(a)}\times D_1\times\cdots D_N}$

$\mathcal{X} \leftarrow \textsc{Reshape}(\mathcal{X})$ // $\mathbb{R}^{B\times F\times D_1\times\cdots\times D_N}$

return $\mathcal{X}$

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

3.4 Complexity of GKPD Factorized Convolutions

The GKPD decompostion of a tensor is not unique. Different choices of Kronecker factor dimensions will lead to different reductions in memory and number of operations. Specifically, for an N-dimensional convolution performed using a tensor represented with $\widehat{R}$ Kronecker products of factors $\mathcal{A} \in \mathbb{R}^{f^{(a)}\times c^{(a)}\times d_1^{(a)}\times\cdots\times d_N^{(a)}}$ and $\mathcal{B} \in \mathbb{R}^{f^{(b)}\times c^{(b)}\times d_1^{(b)}\times\cdots\times d_N^{(b)}}$ the memory reduction is

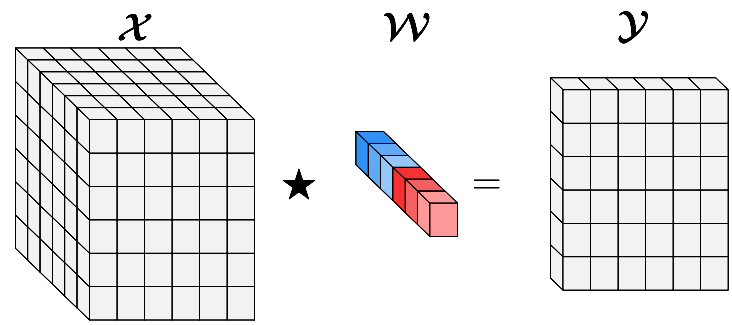
$$\text{CR} = \frac{f^{(a)}c^{(a)}f^{(b)}c^{(b)}\prod_{n=1}^{N} d_n^{(a)}d_n^{(b)}}{\widehat{R}\left(f^{(a)}c^{(a)}\prod_{n=1}^{N} d_n^{(a)} + f^{(b)}c^{(b)}\prod_{n=1} d_n^{(b)}\right)}, \tag{3.38}$$

whereas the reduction in FLOPs is

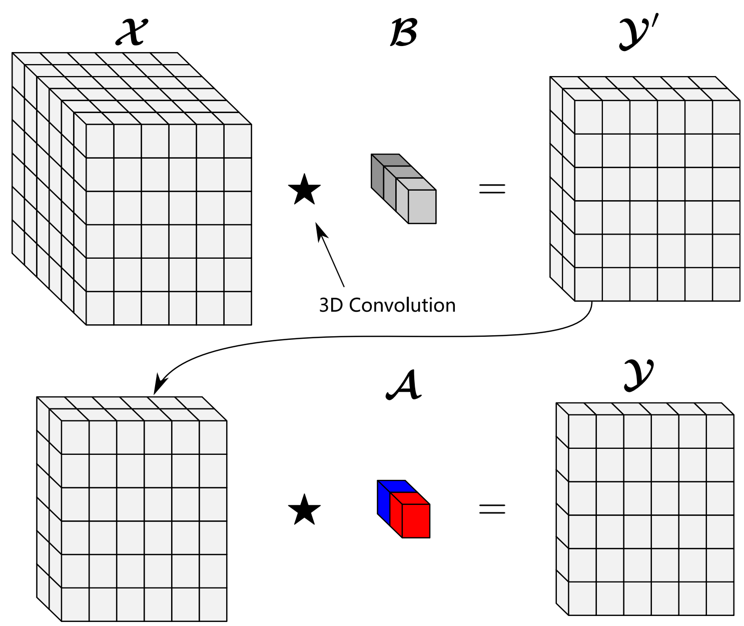
$$\mathrm{FR} = \frac{f^{(a)}c^{(a)}f^{(b)}c^{(b)}\prod_{n=1}^{N} d_n^{(a)}d_n^{(b)}}{\widehat{R}\left(f^{(b)}\cdot f^{(a)}c^{(a)}\prod_{n=1}^{N} d_n^{(a)} + c^{(a)}\cdot f^{(b)}c^{(b)}\prod_{n=1}^{N} d_n^{(b)}\right)}. \tag{3.39}$$

For the special case of a 2-dimensional convolution using separable 3×3 filters, and $\widehat{R} = 1$ the reduction in FLOPs becomes

$$\frac{3f^{(a)}c^{(b)}}{f^{(a)} + c^{(b)}}. \tag{3.40}$$



(a) Conv2d



(b) KroneckerConv2d

Figure 3.2: Illustration of KroneckerConvNd algorithm (for the 2-Dimensional setting, using a Kronecker rank of one). Although (a) and (b) result in identical outputs, the latter is more efficient in terms of memory and FLOPs.

Chapter 4

Applying GKPD to DNN models

In this chapter, we highlight two approaches to selecting the shapes of the Kronecker decomposition of convolution tensors when applying GKPD to DNN models.

Modern deep learning architectures in the computer vision domain often contain many convolution tensors of varying shapes. As the GKPD decomposition is not unique, some thought must be given to *configuration selection* in applying GKPD to model compression. For clarity, we define what is meant by a configuration below.

Definition 2. *(Configuration) Given a tensor $\mathcal{W} \in \mathbb{R}^{w_1 \times \cdots \times w_d}$ and its approximation using $\widehat{R}$ Kronecker factors*

$$\mathcal{A} \in \mathbb{R}^{a_1 \times \cdots \times a_d} \quad \textit{and} \quad \mathcal{B} \in \mathbb{R}^{b_1 \times \cdots \times b_d}, \tag{4.1}$$

such that $a_i \cdot b_i = w_i$ obtained from the solution to the Multidimensional Nearest Kronecker Product problem (3.11); the collection $(\widehat{R}, \mathbf{d}_{\mathcal{A}}, \mathbf{d}_{\mathcal{B}})$ is referred to as a configuration.

In our work we experimented with two configuration selection methods. The first method selected configurations that minimized a reconstruction error. The second was developed out of practical considerations and made selections from a smaller subset of configurations that the lowest runtime latency. In the coming sections, we describe these methods in mode detail.

4.1 Minimum Reconstruction Error Configuration

Given a tensor $\mathcal{W} \in \mathbb{R}^{w_1 \times \cdots \times w_d}$ from a pre-trained network, we find the optimal configuration by solving

$$\begin{aligned} \min_{\substack{\mathcal{A}_r, \mathcal{B}_r, \\ \mathbf{d}_{\mathcal{A}}, \mathbf{d}_{\mathcal{B}}, \widehat{R}}} \quad & \left\| \mathcal{W} - \sum_{r=1}^{\widehat{R}} \mathcal{A}_r \otimes \mathcal{B}_r \right\|_F^2 \\ \text{s.t.} \quad & \widehat{R} a_1 \cdots a_d b_1 \cdots b_d \leq c. \end{aligned} \tag{4.2}$$

In other words, we select the configuration by minimizing over the Kronecker factors' values as well as their dimensions and the choice of rank. Though its discrete nature complicates this problem, in practice it is easy to find its solution. Specifically, (4.2) can be solved by performing the minimization in two stages

$$\begin{aligned} \min_{\mathbf{d}_{\mathcal{A}}, \mathbf{d}_{\mathcal{B}}} \; \min_{\mathcal{A}_r, \mathcal{B}_r, \widehat{R}} \quad & \left\| \mathcal{W} - \sum_{r=1}^{\widehat{R}} \mathcal{A}_r \otimes \mathcal{B}_r \right\|_F^2, \\ \text{s.t.} \quad & \widehat{R} a_1 \cdots a_d b_1 \cdots b_d \leq c. \end{aligned} \tag{4.3}$$

In other words, (4.2) is solved in two stages by enumerating through all possible configurations that satisfy the constraint and solving the inner minimization at each iteration. As the objective function is a non-increasing function of $\widehat{R}$ (i.e., using a larger rank leads to better reconstruction in (3.15)), it suffices to select the largest value for $\widehat{R}$ that does not violate the constraint at each iteration.

4.2 Minimum Latency Configuration

Though the GKPD factorization of a convolution layer's weight tensor reduces its number of FLOPs, in practice it can increase its runtime. Therefore, we also explored configurations that were of no detriment to latency. Specifically, for a given desired compression rate $\mathrm{CR} \in (0, 1)$ we select the optimal configuration by solving

$$\begin{aligned} \min_{\mathbf{d}_{\mathcal{A}}, \mathbf{d}_{\mathcal{B}}, \widehat{R}} \quad & \| \widehat{R} a_1 \cdots a_d b_1 \cdots b_d - p_{ORIG} \, \mathrm{CR} \|, \\ \text{s.t.} \quad & l_{GKPD} \leq l_{ORIGINAL}, \end{aligned} \tag{4.4}$$

where p_{ORIG} is the number of parameters prior to compression and $l_{GKPD}, l_{ORIGINAL}$ are the runtime latencies of the compressed layer and the original layer respectively.

The solution to (4.4) is obtained by generating the runtimes for all configurations for a given tensor, and selecting a configuration that best matches the desired compression rate from a the subset of configurations that do not increase runtime beyond that of the original uncompressed layer.

Chapter 5

Experimental Results

This section presents our experimental results for the application of GKPD to various computer vision domains, such as image classification and action recognition. Here we focus on comparing GKPD to state-of-the-art decomposition methods on the aforementioned tasks, though comparisons to other compression approaches such as knowledge distillation and pruning are also reported.

5.1 Image Classification

In this section, we provide model compression experimental results for image classification tasks using a variety of popular CNN architectures such as ResNet [1] and SEResNet which benefits from the squeeze-and-excitation blocks [18]. We also choose to apply our compression method on MobileNetV2 [17] 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 Appendix B.

Table 5.1 shows the top-1 accuracy on the CIFAR-10 [54] 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

Model	Params (M)	FLOPs (M)	Top-1 (%)
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 5.1: 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.

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 5.2 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

Model	Params (M)	CR	Top-1 (%)
KroneckerResNet18	0.48	23.27×	92.62
KroneckerSeResNet50	0.93	23.01×	93.66
	0.29	73.79×	91.85
KroneckerMobileNetV2	0.73	3.15×	93.80
	0.29	7.90×	93.01
	0.18	12.78×	91.48

Table 5.2: Top-1 accuracy measured on CIFAR-10 highly compressed ResNet18 [1], MobileNetV2 [17] and SEResNet [18].

8 × 16 × 32 × 64 × 128 ×

Figure 5.1: Image reconstruction results when using GKPD (top) and TT (bottom) for image compression. GKPD leads to a lower reconstruction error due to its ability to exploit local redundancy in multiple dimensions simultaneously.

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×.

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

5.1.1 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 [10], Tensor-Train [13], and Tensor-Ring [14], 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 [14].

The model compression results obtained using different decomposition methods aiming for a 5× compression rate are shown in Table 5.3. 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 capacity to exploit local redundancy in multiple dimensions simultaneously,

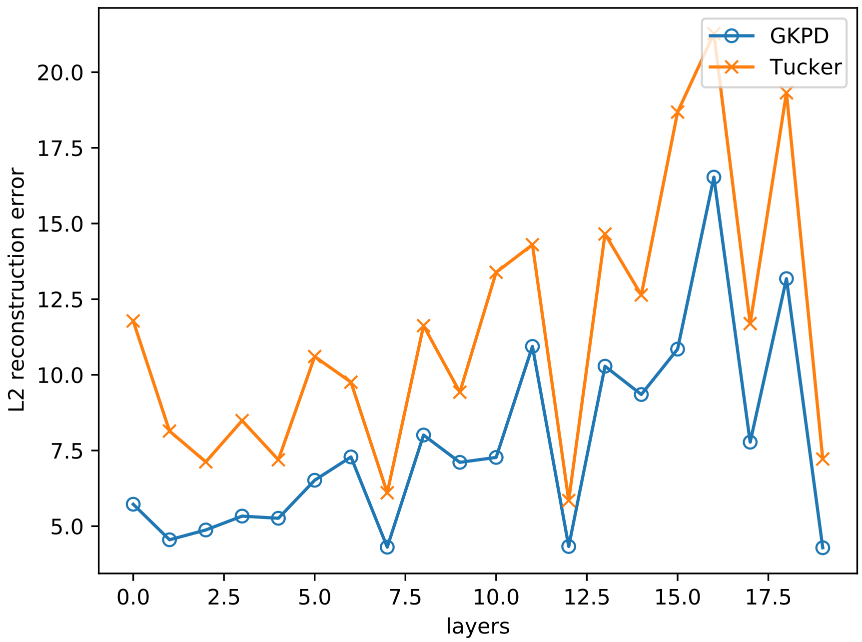


Figure 5.2: Reconstruction error between convolution tensors in a ResNet18 model pretrained on ImageNet and compressed representations at a 4× compression rate, for each layer in the network (x-axis denotes layer number). GKPD always yields a lower reconstruction error than Tucker decomposition

as shown in Figure 5.1, as well as its ability to better reconstruct weight tensors in a pretrained network, as illustrated in Figure 5.2.

5.1.2 Comparison with Other Compression Methods

We compare our proposed model compression method with two state-of-the-art KD-based compression methods [42, 43]. These methods are known to be very effective on relatively smaller networks, such as ResNet26. Thus, we perform our compression method on ResNet26 architecture in these experiments. Table 5.4 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 ∼2× and ∼5×, respectively, compared to

Model	Params (M)	CR	Top-1 (%)
Resnet32	0.46	$1\times$	92.55
TuckerResNet32	0.09	$5\times$	87.7
TensorTrainResNet32	0.096	$4.8\times$	88.3
TensorRingResNet32	0.09	$5\times$	90.6
KroneckerResNet32	0.09	$5\times$	**91.52**

Table 5.3: Top-1 Accuracy on CIFAR-10 of compressed ResNet32 using various decomposition approaches.

using the KD-based model compression methods.

5.1.3 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 [16]. Table 5.5

Model	Params (M)	Compr.	Top-1 (%)
ResNet26	0.37	$1\times$	92.94
TA [42]	0.17	$2.13\times$	91.23
DB [43]	0.17	$2.13\times$	90.34
KroneckerResNet26	0.14	$2.69\times$	**93.16**
TA [42]	0.075	$4.88\times$	88.0
DB [43]	0.075	$4.88\times$	87.32
KroneckerResNet26	0.069	$5.29\times$	**91.28**

Table 5.4: Top-1 accuracy measured on CIFAR-10 for the baseline model ResNet26 and its compressed versions obtained using the KD-based methods; [42], [43], and the proposed GKPD method.

Model	Params (M)	Compr.	Top-1 (%)
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 5.5: Top-1 accuracy measured on ImageNet for the baseline model ResNet50 and its compressed versions obtained using ThiNet [55], FSNet [47], and the proposed GKPD method.

lists the top-1 accuracy for ResNet50 baseline and its compressed variation. We achieve a compression rate of 2.13× 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 [55] and FSNet [47]. ThiNet and FSNet are based on pruning and filter sharing techniques, respectively. They both reportedly, lead to a good accuracy on large datasets. Table 5.5 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 ∼2× compression rate. Note that the performance obtained using our method is based on a random initialization, while the compression achieved with ThiNet benefits from a pretrained model. These results indicate that the proposed GKPD can lead to a high performance even if a pretrained model is not available.

Model	$\widehat{R}$	Params (M)	FLOPs (M)	Top-1 (%)
ResNet18	-	11.17	0.58	95.05
KroneckerResNet18	4	1.41	0.17	92.96
	8	1.42	0.16	93.74
	16	1.44	0.26	94.30
	32	1.51	0.32	94.58

Table 5.6: 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.

5.1.4 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 (3.11). In Table 5.6 we study the effect of using a larger $\widehat{R}$ in Kronecker networks while keeping the overall number of parameters and FLOPs constant. We find that using a larger $\widehat{R}$ does indeed improve performance.

5.2 Action Recognition

In this section we compare the proposed GKPD to other decomposition methods on compression of an I3D-ResNet50, an action recognition model, on the HMDB-51 dataset. As HMDB-51 is a relatively small dataset, it is common practice to pretrain on a large scale datset prior to finetuning on the HMDB-51 dataset for optimal results. Therefore, in our compression experiments we first pretrain I3D-ResNet50 on the Kinetics-400 dataset, then fine-tune on the HMDB-51 dataset. Finally, we compress the model by decomposing its convolution tensors and perform another stage of fine-tuning.

In this set of experiments, we opt to use the configuration selection strategy detailed in Section 4.2. This leads to compressed action recognition models that do not sacrifice latency, as reported in Table 5.7. Most notably, GKPD with the latency-focused configuration strategy leads to efficient model compression with a runtime of 48 ms in comparison to decomposition methods such as Tucker (246 ms) and TT (194 ms), on a single CPU. However, TT outperforms the other decomposition methods, including our proposed GKPD in classification accuracy.

Model	Params (M)	CR	CPU (ms)	Top-1	Top-5
I3D-ResNet50 (baseline)	46.27	1.00	45	63.73	88.63
Tucker-I3D-ResNet50	39.30	1.18	246	50.00	81.31
TT-I3D-ResNet50	29.14	1.59	194	**61.18**	**86.27**
Kronecker-I3D-ResNet50	30.34	1.52	**48**	59.15	84.90

Table 5.7: Action classification accuracy of compressed I3d-ResNet50 on HMDB-51.

5.3 Conclusion & Future Work

In this thesis, 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 from a Kronecker product representation is an area for future work.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[2] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, June 2022.

[3] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference*, pages 87.1–87.12, September 2016.

[4] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *IEEE Conference on Computer Vision and Pattern Recognition*, July 2017.

[5] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 6202–6211, 2019.

[6] Chun-Fu Chen, Rameswar Panda, Kandan Ramakrishnan, Rogerio Feris, John Cohn, Aude Oliva, and Quanfu Fan. Deep analysis of cnn-based spatio-temporal representations for action recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition*, June 2021.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 91–99, 2015.

[8] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[9] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

[10] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *International Conference on Learning Representations*, 2016.

[11] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *International Conference on Learning Representations*, 2015.

[12] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[13] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and FC layers alike. In *NIPS Workshop on Learning with Tensors: Why Now and How?*, 2016.

[14] Wenqi Wang, Yifan Sun, Wenlin Wang, and Vaneet Aggarwal. Wide compression: Tensor ring nets. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 9329–9338, 2018.

[15] Urmish Thakker, Jesse Beu, Dibakar Gope, Chu Zhou, Igor Fedorov, Ganesh Dasika, and Matthew Mattina. Compressing rnns for iot devices by 15-38x using Kronecker products. *arXiv preprint arXiv:1906.02876*, 2019.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1106–1114, 2012.

[17] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[19] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[20] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1251–1258, 2017.

[21] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[22] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. In *IEEE Conference on Computer Vision and Pattern Recognition*, October 2019.

[23] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[24] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

[25] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2019.

[26] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[27] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3123–3131, 2015.

[28] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *IEEE European Conference on Computer Vision*, pages 525–542, 2016.

[29] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[30] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[31] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[32] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[33] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *ArXiv*, abs/2103.13630, 2022.

[34] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015.

[35] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

[36] Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *IEEE European Conference on Computer Vision*, pages 662–677. Springer, 2016.

[37] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. 29:2074–2082, 2016.

[38] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[39] Aydin Buluc and John R. Gilbert. Challenges and advances in parallel sparse matrix-matrix multiplication. In *Proceedings of the 2008 37th International Conference on Parallel Processing*, ICPP '08, page 503–510, USA, 2008.

[40] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[41] Cristian Buciluundefined, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 535–541, New York, NY, USA, 2006. Association for Computing Machinery.

[42] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5191–5198, 2020.

[43] Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi. Knowledge distillation with adversarial samples supporting decision boundary. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3771–3778. AAAI Press, 2019.

[44] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[45] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference*, 2014.

[46] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[47] Yingzhen Yang, Jiahui Yu, Nebojsa Jojic, Jun Huan, and Thomas S. Huang. Fsnet: Compression of deep convolutional neural networks by filter summary. In *International Conference on Learning Representations*, 2020.

[48] Shuchang Zhou, Jia-Nan Wu, Yuxin Wu, and Xinyu Zhou. Exploiting local structures with the Kronecker layer in convolutional networks. *arXiv preprint arXiv:1512.09194*, 2015.

[49] James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 2408–2417. JMLR.org, 2015.

[50] Roger Grosse and James Martens. A Kronecker-factored approximate fisher matrix for convolution layers. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48

of *Proceedings of Machine Learning Research*, pages 573–582, New York, New York, USA, 20–22 Jun 2016. PMLR.

[51] Zedong Tang, Fenlong Jiang, Maoguo Gong, Hao Li, Yue Wu, Fan Yu, Zidong Wang, and Min Wang. SKFAC: Training neural networks with faster Kronecker-factored approximate curvature. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 13479–13487, June 2021.

[52] Charles F.Van Loan. The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123(1):85–100, 2000.

[53] C. Van Loan and N. Pitsianis. *Approximation with Kronecker products*, pages 293–314. 1992.

[54] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[55] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5068–5076, 2017.

[56] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1314–1324, 2019.

[57] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[58] Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[59] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *IEEE European Conference on Computer Vision*, pages 365–382, 2018.

[60] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[61] Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, volume 30, pages 856–867, 2017.

[62] Dejiao Zhang, Haozhu Wang, Mario Figueiredo, and Laura Balzano. Learning to share: Simultaneous parameter tying and sparsification in deep learning. In *International Conference on Learning Representations*, 2018.

[63] Jiashi Li, Qi Qi, Jingyu Wang, Ce Ge, Yujian Li, Zhangzhang Yue, and Haifeng Sun. Oicsr: Out-in-channel sparsity regularization for compact deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7046–7055, 2019.

[64] Koen Goetschalckx, Bert Moons, Patrick Wambacq, and Marian Verhelst. Efficiently combining svd, pruning, clustering and retraining for enhanced neural network compression. In *Proceedings of the 2nd International Workshop on Embedded and Mobile Deep Learning*, pages 1–6, 2018.

[65] Frederick Tung and Greg Mori. Deep neural network compression by in-parallel pruning-quantization. *IEEE transactions on pattern analysis and machine intelligence*, 42(3):568–579, 2018.

[66] Wenqi Wang, Vaneet Aggarwal, and Shuchin Aeron. Efficient low rank tensor ring completion. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5697–5705, 2017.

[67] Urmish Thakker, Paul N Whatmough, Zhi-Gang Liu, Matthew Mattina, and Jesse Beu. Compressing language models using doped Kronecker products. *arXiv preprint arXiv:2001.08896*, 2020.

[68] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5 MB model size. *arXiv preprint arXiv:1602.07360*, 2016.

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

[70] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

APPENDICES

Appendix A

Theorem Proofs

Lemma 1. *(Sum of squares preserving reshapings)*
Let $\mathcal{W} \in \mathbb{R}^{w_1 \times \cdots \times w_N}$, $\mathcal{A} \in \mathbb{R}^{a_1 \times \cdots \times a_N}$ *and* $\mathcal{B} \in \mathbb{R}^{b_1 \times \cdots \times b_N}$. *Then,*

$$\min_{\{\mathcal{A}_r\},\{\mathcal{B}_r\}} \left\| \mathcal{W} - \sum_{r=1}^{\widehat{R}} \mathcal{A}_r \otimes \mathcal{B}_r \right\|_{\mathrm{F}}^2 \equiv \min_{\{\mathcal{A}_r\},\{\mathcal{B}_r\}} \left\| R_W(\mathcal{W}) - \sum_{r=1}^{\widehat{R}} \mathbf{r}_a(\mathcal{A}_r)\mathbf{r}_b(\mathcal{B}_r)^\top \right\|_{\mathrm{F}}^2 \tag{3.23}$$

where,

$$\textsc{Vec} : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{d_1 \cdots d_N} \tag{3.20 revisted}$$

$$\textsc{Mat} : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{d_1 \times d_2 \cdots d_N} \tag{3.21 revisted}$$

$$\textsc{Unfold} : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{N_p \times d'_1 \times \cdots \times d'_N} \tag{3.19 revisted}$$

$$R_W : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{N_p \times d'_1 \cdots d'_N} \tag{3.12 revisted}$$

$$\mathbf{r}_a : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{d_1 \cdots d_N} \tag{3.13 revisted}$$

$$\mathbf{r}_b : \mathbb{R}^{d_1 \times \cdots \times d_N} \to \mathbb{R}^{d_1 \cdots d_N} \tag{3.14 revisted}$$

are reshaping operations with $d_i, d'_i \in \mathbb{N}$ and index mappings

$$I_{\textsc{Unfold}}(\mathbf{i}^{(a)},\, \mathbf{d}^{(\mathcal{A})},\, \mathbf{d}^{(\mathcal{B})}) \triangleq \begin{pmatrix} \left[P^{(u)}\mathbf{c}\right]_{p(\mathbf{i}^{(a)},\,\mathbf{d}^{(\mathcal{A})},\,\mathbf{d}^{(\mathcal{B})})} \\ \mathbf{i}^{(a)} \bmod \mathbf{d}^{(\mathcal{A})} \end{pmatrix}, \tag{3.24}$$

$$I_{\textsc{Vec}}(\mathbf{i}^{(a)},\, \mathbf{d}^{(\mathcal{A})}) \triangleq \left[P^{(v)}\mathbf{c}\right]_{p(\mathbf{i}^{(a)},\,\mathbf{d}^{(\mathcal{A})},\,\mathbf{d}^{(\mathcal{I}_\mathcal{A})})}, \tag{3.25}$$

$$I_{\textsc{Mat}}(\mathbf{i}^{(a)},\, \mathbf{d}^{(\mathcal{A})}) \triangleq \begin{pmatrix} \mathbf{i}^{(a)}_1 \\ I_{\textsc{Vec}}(\mathbf{i}^{(a)}_{2:},\, \mathbf{d}^{(\mathcal{A})}_{2:}) \end{pmatrix}, \tag{3.26}$$

$$\tag{3.27}$$

where $P^{(u)}$ and $P^{(v)}$ denote permutation matrices, vector $\mathbf{c}$ contains an integer enumeration starting at one and ending at $\prod_i \left[\frac{\mathbf{d}^{(\mathcal{A})}}{\mathbf{d}^{(\mathcal{B})}}\right]_i$, mod *denotes an element-wise modulo operation, and*

$$p(\mathbf{i}^{(a)},\, \mathbf{d}^{(\mathcal{A})},\, \mathbf{d}^{(\mathcal{B})}) = \operatorname{sum}\left(\left\lfloor \frac{\mathbf{i}^{(a)}}{\mathbf{d}^{(\mathcal{B})}} \right\rfloor \times \begin{pmatrix} 1 \\ L\left[\frac{\mathbf{d}^{(\mathcal{A})}}{\mathbf{d}^{(\mathcal{B})}}\right]_{2:} \end{pmatrix}\right) \tag{3.28}$$

with the multiplication, division and floor operations being element-wise.

Proof. Let $\mathcal{W}' = \textsc{Unfold}(\mathcal{W},\, \mathbf{d}^{(\mathcal{B})})$, $W = \textsc{Mat}(\mathcal{W}')$, $\mathcal{A}' = \textsc{Unfold}(\mathcal{A},\, \mathbf{d}^{(\mathcal{I}_\mathcal{A})})$, $\mathbf{a} = r_a(\mathcal{A}')$ and $\mathbf{b} = r_b(\mathcal{A}')$ such that

$$W_{\mathbf{i}^{(b)}} = \mathcal{W}'_{\mathbf{i}^{(b')}} = \mathcal{W}_{\mathbf{i}^{(a)}}, \quad \mathbf{a}_{j^{(b)}} = \mathcal{A}_{\mathbf{j}^{(a)}}, \quad \mathbf{b}_{k^{(b)}} = \mathcal{B}_{\mathbf{k}^{(a)}}. \tag{A.1}$$

with index mappings

$$\mathbf{i}^{(b')} = I_{\textsc{Unfold}}(\mathbf{i}^{(a)},\, \mathbf{d}^{(\mathcal{W})},\, \mathbf{d}^{(\mathcal{B})}) = \begin{pmatrix} \left[P^{(u)}\mathbf{c}\right]_{p(\mathbf{i}^{(a)},\,\mathbf{d}^{(\mathcal{W})},\,\mathbf{d}^{(\mathcal{B})})} \\ \mathbf{i}^{(a)} \bmod \mathbf{d}^{(\mathcal{B})} \end{pmatrix} \tag{A.2}$$

$$\mathbf{i}^{(b)} = I_{\textsc{Mat}}(\mathbf{i}^{(b')}\, \mathbf{d}^{(\mathcal{W}')},\, \mathbf{d}^{(\mathcal{B})}) = \begin{pmatrix} \left[P^{(u)}\mathbf{c}\right]_{p(\mathbf{i}^{(a)},\,\mathbf{d}^{(\mathcal{W})},\,\mathbf{d}^{(\mathcal{B})})} \\ \left[P^{(v)}\mathbf{c}\right]_{p(\mathbf{i}^{(a)} \bmod \mathbf{d}^{(\mathcal{B})},\,\mathbf{d}^{(\mathcal{B})},\,\mathbf{d}^{(\mathcal{I}_\mathcal{B})})} \end{pmatrix} \tag{A.3}$$

$$\mathbf{j}^{(b')} = I_{\textsc{Unfold}}(\mathbf{j}^{(a)},\, \mathbf{d}^{(\mathcal{A})},\, \mathbf{d}^{(\mathcal{I}_\mathcal{A})}) = \begin{pmatrix} \left[P^{(u)}\mathbf{c}\right]_{p(\mathbf{j}^{(a)},\,\mathbf{d}^{(\mathcal{A})},\,\mathbf{d}^{(\mathcal{I}_\mathcal{A})})} \\ \mathbf{j}^{(a)} \bmod \mathbf{d}^{(\mathcal{I}_\mathcal{A})} \end{pmatrix} \tag{A.4}$$

$$\mathbf{j}^{(b)} = I_{\textsc{Vec}}(\mathbf{j}^{(b')},\, \mathbf{d}^{(\mathcal{A}')}) = \left[P^{(u)}\mathbf{c}\right]_{p(\mathbf{j}^{(a)},\,\mathbf{d}^{(\mathcal{A})},\,\mathbf{d}^{(\mathcal{I}_\mathcal{A})})} \tag{A.5}$$

$$\mathbf{k}^{(b)} = I_{\textsc{Vec}}(\mathbf{k}^{(a)},\, \mathbf{d}^{(\mathcal{B})}) = \left[P^{(u)}\mathbf{c}\right]_{p(\mathbf{k}^{(a)},\,\mathbf{d}^{(\mathcal{B})},\,\mathbf{d}^{(\mathcal{I}_\mathcal{B})})}. \tag{A.6}$$

It suffices to show that

$$\mathbf{i}^{(b)} = \begin{pmatrix} j^{(b)} \\ k^{(b)} \end{pmatrix}. \tag{A.7}$$

Consider the first element

$$j^{(b)} = \left[P^{(u)}\mathbf{c}\right]_{p(\mathbf{j}^{(a)},\, \mathbf{d}^{(\mathcal{A})},\, \mathbf{d}^{(\mathcal{I}_\mathcal{A})})} \tag{A.8}$$

with index

$$\begin{aligned}
p(\mathbf{j}^{(b')},\, \mathbf{d}^{(\mathcal{A}')},\, \mathbf{d}^{(\mathcal{I}_{\mathcal{A}'})}) &= \operatorname{sum}\left(\left\lfloor \frac{\mathbf{j}^{(b')}}{\mathbf{d}^{(\mathcal{I}_{\mathcal{A}'})}} \right\rfloor \times \begin{pmatrix} 1 \\ L\left[\frac{\mathbf{d}^{(\mathcal{A}')}}{\mathbf{d}^{(\mathcal{I}_{\mathcal{A}'})}}\right]_{2:} \end{pmatrix}\right) && \text{(A.9)} \\
&= \operatorname{sum}\left(\mathbf{j}^{(b')} \times \begin{pmatrix} 1 \\ L\mathbf{d}_{2:}^{(\mathcal{A}')} \end{pmatrix}\right) && \text{(A.10)} \\
&= \operatorname{sum}\left(\begin{pmatrix} \left[P^{(u)}\mathbf{c}\right]_{p(\mathbf{j}^{(a)},\, \mathbf{d}^{(\mathcal{A})},\, \mathbf{d}^{(\mathcal{I}_\mathcal{A})})} \\ \mathbf{j}^{(a)} \bmod \mathbf{d}^{(\mathcal{I}_\mathcal{A})} \end{pmatrix} \times \begin{pmatrix} 1 \\ L\mathbf{d}_{2:}^{(\mathcal{A}')} \end{pmatrix}\right) && \text{(A.11)} \\
&= \left[P^{(u)}\mathbf{c}\right]_{p(\mathbf{j}^{(a)},\, \mathbf{d}^{(\mathcal{A})},\, \mathbf{d}^{(\mathcal{I}_\mathcal{A})})} && \text{(A.12)}
\end{aligned}$$

where (A.12) holds as $\mathbf{j}^{(a)} \bmod \mathbf{d}^{(\mathcal{I}_\mathcal{A})}$ is a zero vector. The first element in $\mathbf{i}^{(\mathcal{B})}$

$$\left[P^{(u)}\mathbf{c}\right]_{p(\mathbf{i}^{(a)},\, \mathbf{d}^{(\mathcal{W}')},\, \mathbf{d}^{(\mathcal{B})})} \tag{A.13}$$

is equivalent to (A.12) as

$$\begin{aligned}
p(\mathbf{i}^{(a)},\, \mathbf{d}^{(\mathcal{W})},\, \mathbf{d}^{(\mathcal{B})}) &= \operatorname{sum}\left(\left\lfloor \frac{\mathbf{i}^{(a)}}{\mathbf{d}^{(\mathcal{B})}} \right\rfloor \times \begin{pmatrix} 1 \\ L\left[\frac{\mathbf{d}^{(\mathcal{W})}}{\mathbf{d}^{(\mathcal{B})}}\right]_{2:} \end{pmatrix}\right) && \text{(A.14)} \\
&= \operatorname{sum}\left(j^{(a)} \times \begin{pmatrix} 1 \\ L\mathbf{d}_{2:}^{(\mathcal{A})} \end{pmatrix}\right) && \text{(A.15)} \\
&= p(\mathbf{j}^{(a)},\, \mathbf{d}^{(\mathcal{A})},\, \mathbf{d}^{(\mathcal{I}_\mathcal{A})}). && \text{(A.16)}
\end{aligned}$$

Similarly, the second element in $\mathbf{i}^{(\mathcal{B})}$

$$\left[P^{(v)}\mathbf{c}\right]_{p(\mathbf{i}^{(a)} \bmod \mathbf{d}^{(\mathcal{B})},\, \mathbf{d}^{(\mathcal{B})},\, \mathbf{d}^{(\mathcal{I}_\mathcal{B})})} \tag{A.17}$$

is equivalent to

$$\left[P^{(v)}\mathbf{c}\right]_{p(\mathbf{k}^{(a)},\,\mathbf{d}^{(\mathcal{B})},\,\mathbf{d}^{(\mathcal{I}_{\mathcal{A}})})} \tag{A.18}$$

because

$$p(\mathbf{i}^{(a)} \bmod \mathbf{d}^{(\mathcal{A})},\, \mathbf{d}^{(\mathcal{B})},\, \mathbf{d}^{(\mathcal{I}_{\mathcal{B}})}) = \mathrm{sum}\left(\left\lfloor \frac{\mathbf{i}^{(a)} \bmod \mathbf{d}^{(\mathcal{B})}}{\mathbf{d}^{(\mathcal{I}_{\mathcal{B}})}} \right\rfloor \times \begin{pmatrix} 1 \\ L\left[\frac{\mathbf{d}^{(\mathcal{B})}}{\mathbf{d}^{(\mathcal{I}_{\mathcal{B}})}}\right]_{2:} \end{pmatrix}\right) \tag{A.19}$$

$$= \mathrm{sum}\left(\mathbf{i}^{(a)} \bmod \mathbf{d}^{(\mathcal{B})} \times \begin{pmatrix} 1 \\ L\mathbf{d}^{(\mathcal{B})}_{2:} \end{pmatrix}\right) \tag{A.20}$$

$$= \mathrm{sum}\left(\mathbf{k}^{(a)} \times \begin{pmatrix} 1 \\ L\mathbf{d}^{(\mathcal{B})}_{2:} \end{pmatrix}\right) \tag{A.21}$$

$$= p(\mathbf{k}^{(a)},\, \mathbf{d}^{(\mathcal{B})},\, \mathbf{d}^{(\mathcal{I}_{\mathcal{A}})}) \tag{A.22}$$

□

Theorem 1. *(Optimality of GKPD)*
Any tensor $\mathcal{W} \in \mathbb{R}^{w_1 \times \cdots \times w_N}$ can be represented exactly as

$$\mathcal{W} = \sum_{r=1}^{R} \mathcal{A} \otimes \mathcal{B} \tag{3.29}$$

where $\mathcal{A}_r \in \mathbb{R}^{a_1 \times \cdots \times a_N}$, $\mathcal{B}_r \in \mathbb{R}^{b_1 \times \cdots \times b_N}$ and $R \in \mathbb{N}$

Proof. Using reshaping operations in eqs. (3.12)–(3.14), , we can re-write the reconstruction error

$$\varepsilon(\mathcal{A},\, \mathcal{B}) = \left\| \mathcal{W} - \sum_{r=1}^{\widehat{R}} \mathcal{A}_r \otimes \mathcal{B}_r \right\|_{\mathrm{F}}^2, \tag{A.23}$$

as

$$\varepsilon(\mathcal{A},\, \mathcal{B}) = \left\| R_W(\mathcal{W}) - \sum_{r=1}^{\widehat{R}} \mathbf{r}_a(\mathcal{A}_r)\mathbf{r}_b(\mathcal{B}_r)^\top \right\|_{\mathrm{F}}^2 \tag{A.24}$$

due to the sum-of-squares preserving propert of these reshaping operations. Minimization of the reconstruction error in (A.24)is equivalent to solving a low-rank matrix approximation problem which has a well-known SVD-based solution. Therefore,

$$\varepsilon(\mathcal{A},\, \mathcal{B}) = \left\| R_W(\mathcal{W}) - \sum_{r=1}^{\widehat{R}} \mathbf{r}_a(\mathcal{A}_r)\mathbf{r}_b(\mathcal{B}_r)^\top \right\|_{\mathrm{F}}^2 \tag{A.25}$$

$$\leq \sum_{r=\widehat{R}+1}^{R} \sigma_r^2(R_W(\mathcal{W}) \tag{A.26}$$

where R and σ_r denote is the rank and r^{th} largest singular value of matrix $R_W(\mathcal{W}$ respectively. Choosing $\widehat{R} = R$ results in a zero reconstruction error. □

Lemma 2. *(Linear Projections with Kronecker factorized tensors) Given tensor $\mathcal{W} \in \mathbb{R}^{w_1 \times \cdots \times w_N}$ and its GKPD factors $\mathcal{A} \in \mathbb{R}^{a_1 \times \cdots \times a_N}$, $\mathcal{B} \in \mathbb{R}^{b_1 \times \cdots \times b_N}$ 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 as follows:*

$$\mathcal{W}_{i_1 \cdots i_N} \mathcal{X}_{i_1 \cdots i_N} = \mathcal{A}_{j_1 \cdots j_N} \mathcal{B}_{k_1 \cdots k_N} \mathcal{X}_{g(j_1,k_1) \cdots g(j_N,k_N)}, \tag{3.33}$$

where $\mathcal{X} \in \mathbb{R}^{d_1 \times \cdots \times d_N}$ is an input tensor,

$$g(j_n, k_n) \triangleq j_n b_n + k_n, \tag{3.34}$$

is a re-indexing function; and j_n, k_n are as defined in (3.8). The equality also holds for any valid offsets to the input's indices. That is,

$$\mathcal{W}_{i_1 \cdots i_N} \mathcal{X}_{i_1 + o_1, \cdots, i_N + o_N} = \mathcal{A}_{j_1 \cdots j_N} \mathcal{B}_{k_1 \cdots k_N} \mathcal{X}_{g(j_1,k_1)+o_1, \cdots, g(j_N,k_N)+o_N}, \tag{3.35}$$

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

Proof. By definition the terms in tensor $\mathcal{W}$ are given by

$$\mathcal{W}_{i_1 \cdots i_N} \triangleq \mathcal{A}_{j_1 \cdots j_N} \mathcal{B}_{k_1 \cdots k_N} \tag{A.27}$$

where

$$j_n = \left\lfloor \frac{i_n}{b_n} \right\rfloor, \quad k_n = i_n \bmod b_n.$$

Since j_n and k_n decompose i_n into an integer quotient and a remainder term (with respect to divisor b_n), it follows that

$$g(j_n, k_n) \triangleq j_n b_n + k_n = i_n \tag{A.28}$$

Therefore,

$$\mathcal{X}_{i_1+o_1,\cdots,i_N+o_N} = \mathcal{X}_{g(j_1,k_1)+o_1,\cdots,g(j_N,k_N)+o_N}. \tag{A.29}$$

Combining (A.27) and (A.29) completes the proof. □

Appendix B

Implementation Details

CIFAR-10 experiments were conducted using a two NVIDIA V100 GPUs with the following training parameters:

- Epochs set to 200
- Batch size set to 128
- Learning rate initially set to 0.1 and reduced by a factor of 10× at iterations 100 and 150
- Optimization was done using Stochastic Gradient Descent with a weight decay of 0.0001 and momentum of 0.9
- Data augmentation was done by randomly flipping images horizontally throughout training
- Input images standardized using dataset mean and standard deviation values

ImageNet experiments were conducted using a four NVIDIA V100 GPUs with the following training parameters:

- Epochs set to 100
- Batch size set to 256
- Learning rate initially set to 0.1 and reduced by a factor of 10× at iterations 30, 60 and 90

- Optimization was done using Stochastic Gradient Descent with a weight decay of 0.0001 and momentum of 0.9
- Data augmentation was done by randomly cropping 224×224 patches from 256×256 input images followed by randomly flipping them horizontally throughout training
- Input images were standardized using dataset mean and standard deviation values