# BATUDE: Budget-Aware Neural Network Compression Based on Tucker Decomposition

**Miao Yin**[1], **Huy Phan**[1], **Xiao Zang**[1], **Siyu Liao**[2†], **Bo Yuan**[1]

[1]Department of Electrical and Computer Engineering, Rutgers University, [2]Amazon
{miao.yin, huy.phan, xiao.zang}@rutgers.edu, liasiyu@amazon.com, bo.yuan@soe.rutgers.edu

## Abstract

Model compression is very important for the efficient deployment of deep neural network (DNN) models on resource-constrained devices. Among various model compression approaches, high-order tensor decomposition is particularly attractive and useful because the decomposed model is very small and fully structured. For this category of approaches, tensor ranks are the most important hyper-parameters that directly determine the architecture and task performance of the compressed DNN models. However, as an NP-hard problem, selecting optimal tensor ranks under the desired budget is very challenging and the state-of-the-art studies suffer from unsatisfied compression performance and timing-consuming search procedures. To systematically address this fundamental problem, in this paper we propose BATUDE, a Budget-Aware TUcker DEcomposition-based compression approach that can efficiently calculate optimal tensor ranks via one-shot training. By integrating the rank selecting procedure to the DNN training process with a specified compression budget, the tensor ranks of the DNN models are learned from the data and thereby bringing very significant improvement on both compression ratio and classification accuracy for the compressed models. The experimental results on ImageNet dataset show that our method enjoys 0.33% top-5 higher accuracy with 2.52× less computational cost as compared to the uncompressed ResNet-18 model. For ResNet-50, the proposed approach enables 0.37% and 0.55% top-5 accuracy increase with 2.97× and 2.04× computational cost reduction, respectively, over the uncompressed model.

## 1 Introduction

Deep neural networks (DNNs) have been widely used in many applications, such as image classification (He et al. 2016), objective detection (Girshick 2015) and image caption (Xu et al. 2015). Despite the current unprecedented success, deploying DNNs on resource-constrained devices is challenging – DNNs have huge storage and computational demands to guarantee performance. To address this problem, many techniques like pruning (Han, Mao, and Dally 2015) and quantization (Han et al. 2015) have been proposed to compress and accelerate DNNs. Among these methods, tensor decomposition-based compression (Kim et al. 2016; Novikov et al. 2015; Garipov et al. 2016; Yang, Krompass, and Tresp 2017; Wang et al. 2018; Pan et al. 2019) is recently getting increasing attentions. By decomposing large weight matrices and tensors of DNNs to multiple small tensor cores, tensor decomposition can bring very high compression ratio with low performance degradation. Such unique compression strategy has another promising benefit – the decomposed model is very hardware friendly since the kernel computation is just a series of small and dense matrix multiplications, which are very attractive for various computing platforms, such as ASIC/FPGA and mobile devices (Deng et al. 2019a,b; Kim et al. 2016).

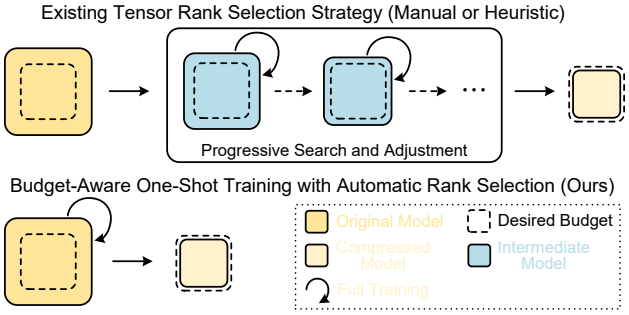


Figure 1: Comparison with existing tensor rank selection methods, where the tensor ranks are determined via a complicated search process with many times of full training.

**Tensor Rank Selection: Importance & Challenges.** To obtain a high-performance compressed DNN model under *desired budget* (e.g. model size or computational cost) via tensor decomposition approach, proper *tensor rank* selection for the decomposed tensor is particularly important and critical. To be specific, as the hyper-parameter that reflects the linear correlations in each dimension, tensor ranks directly define the architecture and overall size of the decomposed DNN model. Consequently, selecting the proper tensor ranks significantly controls and impacts DNN *compression performance*, in terms of storage and computational complexity, as well as DNN *task performance*, such as classification accuracy and prediction quality.

Despite its criticality and fundamentality, selecting the optimal rank is very challenging. Different from the rank selection in matrix decomposition, where the rank is a scalar, the rank selection in tensor decomposition needs to identify


[†]The work was done when the author was with Rutgers University.

the proper vector-format tensor ranks. As indicated in (Hillar and Lim 2013), exact determination of tensor ranks for linear tensor problem is theoretically NP-hard. Even worse, since a modern DNN model typically consists of tens or even hundreds of layers, the overall search space for determining the rank for the decomposed DNN model is extremely huge.

**Existing Work and Limitations.** To date, though tensor decomposition has emerged as important DNN compression technique, efficient tensor rank selection has not been systematically investigated. In most tensor decomposition-based DNN compression studies (Garipov et al. 2016; Yang, Krompass, and Tresp 2017; Wang et al. 2018; Pan et al. 2019), the tensor ranks for different layers are just set to equal. Such heuristic searching requires tremendous human efforts and extensive engineering – multiple attempts and fine-tuning have to be spent to find a reasonable rank setting. More importantly, considering different layers have different redundancy and sensitivity for compression, their desired tensor ranks should not be forced to be the same. Consequently, such equal rank setting strategy severely hinders the tensor decomposed DNN models to achieve the best compression ratio and accuracy.

Very recently, (Li et al. 2020) and (Gusak et al. 2019) propose to use genetic algorithm and Bayesian approach to find the proper tensor rank configuration, respectively. Compared with the conventional equal setting scheme, these two strategies show better compression ratio and classification accuracy. However these state of the arts still suffer several inherent limitations. For the genetic algorithm-based strategy, inspired by the neural architecture search, it performs time-consuming "search space sampling $\rightarrow$ evolutionary phase $\rightarrow$ rank optimization $\rightarrow$ progressive search" in an iterative way, and hence the overall searching procedure is still very expensive. For the Bayesian approach-based strategy, it is not a budget-aware solution: the tensor ranks cannot be adjusted automatically according to the target compression ratio. In other words, the model complexity determined by the selected ranks may still not satisfy the target resource budget. Another drawback of the Bayesian approach-based rank selection is that it is a layer-wise scheme – the optimal rank selection for each individual layer may not be globally optimal for the entire compressed DNN model.

**Technical Preview and Contributions.** To systematically overcome the challenge of tensor rank selection within a desired budget and obtain high-performance compressed DNN models, in this paper we propose BATUDE, a Budget-Aware TUcker DEcomposition-based compression approach, to automatically select the rank configurations for the decomposed DNN models, as illustrated in Fig. 1. Our key idea is to integrate the rank selection procedure into an one-shot training procedure, and let the model automatically learn the suitable rank setting from the data. To be specific, we first relax the original complicated NP-hard rank selection problem to a tensor nuclear norm-regularized optimization problem with the constraint of model size for DNN training. After such reformulation, during the training procedure we then develop a budget constrained alternating direction Lagrangian (BC-ADL) algorithm to solve this optimization problem via independently updating splitted variables. Upon the end of this one-shot training with iterative optimization, the suitable tensor ranks are automatically learned, and thereby bringing highly-compressed highly-accurate tensor decomposed DNN models with target compression ratio. To sum up, the contributions of this paper are summarized as follows:

- We propose an automatic tensor decomposition-based compression method, which provides high-performance compression service in a user-friendly way. Given the original model and the overall target compression budget, it can automatically select the proper tensor ranks for each layer, and output the compressed DNN model after a single run of training. Meanwhile, the high accuracy of the original model is preserved, and even increased after our compression.
- We propose to mathematically reformulate the original NP-hard rank selection problem to a tensor nuclear norm-regularized optimization problem with global rank budget constraints, thereby enabling automatic global optimal rank learning during one-shot training. To achieve that, we propose BC-ADL algorithm for solving the reformulated optimization problem via alternately updating separate variables in an iterative way. Meanwhile, the tensor ranks are adaptively learned and imposed on the DNN model.
- We conduct experiments to evaluate the performance of the proposed framework on multiple datasets. On CIFAR-100 dataset, with $2.8\times$ and $2.6\times$ compression ratio, our BATUDE brings even $1.27\%$ and $0.85\%$ accuracy increase over the original uncompressed ResNet-20 and ResNet-32 models, respectively. On ImageNet dataset, our approach brings $0.33\%$ top-5 accuracy increase with $2.52\times$ computational cost reduction as compared to the original uncompressed ResNet-18 model. For ResNet-50 our proposed approach enables $0.37\%$ and $0.55\%$ top-5 accuracy increase with $2.97\times$ and $2.04\times$ computational cost reduction, respectively, over the original uncompressed model.

## 2 Related Work

### 2.1 Model Compression

**Pruning.** Pruning, as the most popular model compression approach, realizes model size reduction via setting parts of the weight as zeros. According to different sparsity pattern after pruning, pruning approach can be categorized as unstructured pruning and structured pruning. Unstructured pruning (Han, Mao, and Dally 2015; Zhang et al. 2018) can provide high compression ratio and accuracy, but the inherent unstructured sparsity pattern prevents this approach from delivering its theoretical speedup on the practical hardware platforms. Structured pruning, such as filter (Luo, Wu, and Lin 2017; He et al. 2019; Lin et al. 2020; Sui et al. 2021) and channel (He, Zhang, and Sun 2017; Zhuang et al. 2018; Peng et al. 2019) pruning are hardware-friendly solutions. However, the purposely imposed structure pattern, on the other hand, limits the performance of pruned models in terms of compression ratio and accuracy.

**Quantization.** Quantization, as a compression approach that uses limited number of bits to represent weight and/or activation (Han, Mao, and Dally 2015; Rastegari et al. 2016; Jacob et al. 2018; Gong et al. 2019), is widely adopted in the DNN deployment on practical devices, especially for specialized DNN chips (Han et al. 2016; Chen, Emer, and Sze 2016). The number precision used in the quantization scheme typically highly depends on the resource budget and accuracy requirement. In (Rastegari et al. 2016; Courbariaux, Bengio, and David 2015), 1-bit weight, as a very aggressive quantization scheme, is proposed to realize very low storage and computational costs.

**Tensor Decomposition.** In tensor theory tensor decomposition is used for compact representation of large-scale tensor-format data objective. Starting from (Lebedev et al. 2015; Kim et al. 2016), many existing tensor decomposition approaches have been introduced to compresses DNN models by factorizing the original weight matrices/tensors into small core tensors. In (Kim et al. 2016), Tucker decomposition (Tucker 1966) is adopted for compressing and accelerating convolutional neural networks (CNNs). CP decomposition (Harshman et al. 1970), as a special format of Tucker decomposition, is also used for CNN compression (Lebedev et al. 2015). In (Yang, Krompass, and Tresp 2017; Pan et al. 2019; Yin et al. 2021a,b), tensor train (TT), tensor ring (TR) and hierarchical Tucker (HT) decomposition are used to obtain compact recurrent neural network (RNN) models for video recognition, achieving extremely high parameters reduction. Most recently, (Phan et al. 2020) proposes a low-rank Tucker-CP decomposition, which can stabilize the approximation of weight tensors of convolutional layers, to achieve high-accuracy CNN compression.

## 2.2 Tensor Rank Selection

To obtain high-performance tensor decomposition-based DNN models, proper selection of tensor ranks are very critical and important since tensor ranks directly determine the model size and architecture. Due to the inherent complexity of tensor rank selection procedure, most of the existing tensor decomposition works (Novikov et al. 2015; Garipov et al. 2016; Wang et al. 2018; Yang, Krompass, and Tresp 2017; Pan et al. 2019) adopt manual searching – using multiple attempts and fine-tuning to find a tensor rank setting. Meanwhile, in order to reduce the searching complexity, the identified tensor ranks are shared by different layers. Evidently, such heuristic strategy does not only require extensive engineering, but also causes limited compression/accuracy performance – different layers should not be forced to use the same tensor rank setting.

Some recent studies have begun to investigate the efficient tensor rank determination. In (Li et al. 2020), a genetic algorithm-based searching strategy is proposed to progressively search the tensor ranks for the TR decomposition-based DNN models. Inspired by neural architecture search, this approach performs iterative searching on the sampled model space via some crafted rules. Consequently, the overall searching procedure is still a very time-consuming process. (Gusak et al. 2019) proposes a multi-stage Tucker decomposition-based compression using Bayesian approach-based rank selection. The inherent characteristics of Bayesian approach makes this type of rank determination require additional auxiliary hyper-parameters during the searching. Meanwhile, this strategy is not budget-aware solution: the model size determined by the selected tensor ranks may not satisfy the target compression requirement. Besides, the layer-wise rank selection scheme adopted in this approach may not bring the globally optimal rank determination for the entire compressed model.

# 3 Preliminaries

**Notation.** We use boldface calligraphic script letters to high-dimensional denote tensor, e.g. $\boldsymbol{\mathcal{A}}$. A matrix and a vector are represented by boldface capital letters and boldface lowercase letters, respectively, e.g. $\boldsymbol{A}$ and $\boldsymbol{a}$. Also, non-boldface letters with indices $\mathcal{A}(i_1, \cdots, i_d)$, $A(i, j)$ and $a(i)$ denote the entry of $d$-dimensional tensor $\boldsymbol{\mathcal{A}}$, matrix $\boldsymbol{A}$ and vector $\boldsymbol{a}$, respectively.

**$\tau$-Shrinkage.** Let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ be the singular value decomposition (SVD) of matrix $\boldsymbol{A}$. We define the shrinkage operation as $\mathcal{S}_\tau(\boldsymbol{A}) = \boldsymbol{U}\boldsymbol{\Sigma}_\tau\boldsymbol{V}^T$, where

$$\Sigma_\tau(i) = \begin{cases} \sigma_i & \text{if } \sigma_i > \tau, \\ 0 & \text{otherwise,} \end{cases}$$

and $\sigma_i = \Sigma(i)$ is the $i$-th largest singular value.

**Mode-$k$ Matricization.** Given a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, the mode-$k$ matricization (also called unfolding) of $\boldsymbol{\mathcal{A}}$ is denoted as $\boldsymbol{A}_{(k)} \in \mathbb{R}^{n_k \times (n_1 \cdots n_{k-1} n_{k+1} \cdots n_d)}$. The entry $(i_1, \cdots, i_d)$ of the given tensor is mapped to the entry $(i_k, j)$ of the unfolded matrix, i.e. $\mathcal{A}(i_1, \cdots, i_d) = A_{(k)}(i_k, j)$, where

$$j = 1 + \sum_{p=1, p\neq k}^{d} (i_p - 1) J_p \quad \text{with} \quad J_p = \prod_{q=1, q\neq k}^{p-1} n_q.$$

Based on the matricization, we define the following two operations:

$$\texttt{unfold}_k(\boldsymbol{\mathcal{A}}) = \boldsymbol{A}_{(k)},$$
$$\texttt{fold}_k(\boldsymbol{A}_{(k)}) = \boldsymbol{\mathcal{A}}.$$

**Tucker-Format Convolution.** Considering a normal convolutional layer with kernel tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{S \times C \times K \times K}$, the input tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{W \times H \times C}$ is transformed into the output tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{W' \times H' \times S}$ by convolving with the kernel:

$$\mathcal{Y}(w', h', s) = \sum_{i=1}^{K} \sum_{j=1}^{K} \sum_{c=1}^{C} \mathcal{W}(s, c, i, j) \mathcal{X}(w, h, c) \quad \text{with} \quad w = w' + i - 1 \text{ and } h = h' + j - 1, \tag{1}$$

where $W' = W - K + 1$ and $H' = H - K + 1$.

In Tucker-format convolutional layer, in order to retain the spatial information, the orders associated to the kernel size are not decomposed. This decomposing approach is called Tucker-2 (Tucker 1966) decomposition, by which the kernel tensor can be represented as

$$\mathcal{W}(s, c, i, j) = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \mathcal{C}(r_1, r_2, i, j) B_1(r_1, s) B_2(r_2, c), \tag{2}$$

where $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{R_1 \times R_2 \times K \times K}$, $\boldsymbol{B}_1 \in \mathbb{R}^{R_1 \times S}$ and $\boldsymbol{B}_2 \in \mathbb{R}^{R_2 \times C}$.

With the Tucker-2 format, the original full convolution layer is decomposed into a sequence of small convolutions, thus the output tensor is obtained by:

$$\mathcal{T}_1(w, h, r_2) = \sum_{c=1}^{C} B_2(r_2, c)\mathcal{X}(w, h, c), \tag{3}$$

$$\mathcal{T}_2(w', h', r_1) = \sum_{i=1}^{K}\sum_{j=1}^{K}\sum_{r_2=1}^{R_2} \mathcal{C}(r_1, r_2, i, j)\mathcal{T}_1(w, h, r_2), \tag{4}$$

$$\mathcal{Y}(w', h', s) = \sum_{r_1=1}^{R_1} B_1(r_1, s)\mathcal{T}_2(w', h', r_1), \tag{5}$$

where $\boldsymbol{\mathcal{T}}_1 \in \mathbb{R}^{W \times H \times R_2}$ and $\boldsymbol{\mathcal{T}}_2 \in \mathbb{R}^{W' \times H' \times R_1}$. More details can be referred to (Kim et al. 2016).

**Tucker Rank.** For a tucker-format convolutional layer, only the first and second dimension (i.e. the output and input channel) are decomposed. Correspondingly, according to (Gandy, Recht, and Yamada 2011), the Tucker rank of the 4-dimensional convolutional kernel $\boldsymbol{\mathcal{W}}$ is the tuple of the ranks of mode-1 and mode-2 matricizations:

$$\text{tk-rank}(\boldsymbol{\mathcal{W}}) = [R_1, R_2], \tag{6}$$

where $R_1 = \text{rank}(\boldsymbol{W}_{(1)})$ and $R_2 = \text{rank}(\boldsymbol{W}_{(2)})$.

## 4 Problem Analysis & Formulation

Without loss of generality, we aim to compress an $L$-layer convolutional neural network (CNN) using Tucker decomposition[1]. To simplify the notation, we use $\boldsymbol{\mathcal{W}}$ without subscript to represent the weight of all layers ($\{\boldsymbol{\mathcal{W}}_i\}_{i=1}^{L}$, $\boldsymbol{\mathcal{W}}_i$ is a 4-dimensional kernel tensor) and omit the bias parameters. The loss function is denoted by $\ell(\boldsymbol{\mathcal{W}})$ (e.g. cross-entropy loss for classification task).

With such notation, our goal is to minimize the loss function and the Tucker rank of each layer simultaneously, i.e. $\min_{\boldsymbol{\mathcal{W}}} \ell(\boldsymbol{\mathcal{W}})$ and $\min_{\boldsymbol{\mathcal{W}}} \text{tk-rank}(\boldsymbol{\mathcal{W}})$, with budget constraints to satisfy the target compression ratio. Obviously, decreasing the tensor ranks would inevitably lower the capacity of the CNN model, thus making it more difficult for the loss function to reach the optimal point. Therefore, the best solution for this problem is to find a sweet point where the loss is in the acceptable range and meanwhile the tensor ranks also satisfy the compression demand. Unfortunately, since exact determination of tensor ranks for linear tensor problem is NP-hard (Hillar and Lim 2013), it is impossible to obtain the optimal tensor ranks via direct searching approaches.

To overcome this challenge, based on the fact that the nuclear norm or trace norm $\|\cdot\|_*$ is the tightest surrogate for $\text{rank}(\cdot)$ (Cai, Candès, and Shen 2010; Liu et al. 2012), we relax the second objective and approximately re-formulate the original problem as follows:

$$\begin{aligned} \min_{\boldsymbol{\mathcal{W}}} \;& \ell(\boldsymbol{\mathcal{W}}) + \|\boldsymbol{\mathcal{W}}\|_*, \\ \text{s.t.} \;& \mathcal{P}(\text{tk-rank}(\boldsymbol{\mathcal{W}})) \le \beta, \end{aligned} \tag{7}$$

where $\|\boldsymbol{\mathcal{W}}\|_*$ is the tensor nuclear norm of $\boldsymbol{\mathcal{W}}$, $\mathcal{P}(\cdot)$ returns the overall model size with the current ranks, and $\beta$, as the "budget", is the desired model size after compression.

Recall the Tucker-format convolution, where only the first mode and the second mode are decomposed to retain the spatial information, and the definition of Tucker rank Eq. (6). We define the Tucker-2 nuclear norm of the kernel $\boldsymbol{\mathcal{W}}$ is the sum of the nuclear norms of the mode-1 and mode-2 matricization:

$$\|\boldsymbol{\mathcal{W}}\|_* = \|\boldsymbol{W}_{(1)}\|_* + \|\boldsymbol{W}_{(2)}\|_*. \tag{8}$$

Hence, the formulated optimization problem (7) can be rewritten as

$$\begin{aligned} \min_{\boldsymbol{\mathcal{W}}} \;& \ell(\boldsymbol{\mathcal{W}}) + \|\boldsymbol{W}_{(1)}\|_* + \|\boldsymbol{W}_{(2)}\|_*, \\ \text{s.t.} \;& \mathcal{P}(R_1, R_2) \le \beta. \end{aligned} \tag{9}$$

In the next section, we introduce how to efficiently solve this optimization problem via BC-ADL and automatically learn the tensor ranks.

## 5 Budget-Aware Compression: Learning Tucker Ranks via One-Shot Training

As analyzed in (Liu et al. 2012), minimizing the objective that contains two entries-shared non-smooth nuclear norm terms (e.g. $\boldsymbol{W}_{(1)}$ and $\boldsymbol{W}_{(2)}$), as the problem format that Eq. (9) exactly exhibits, is very challenging. In this section, we derive a budget constrained augmented direction Lagrangian (BC-ADL) algorithm to efficiently solve the optimization problem. BC-ADL is an application of Douglas-Rachford splitting algorithm (Gabay and Mercier 1976; Eckstein and Bertsekas 1992), which can take advantage of the minimization of $f(x) + g(x)$. To that end, we first attribute separate variables to mode-1 and mode-2 matricizations of $\boldsymbol{\mathcal{W}}$, i.e. introduce auxiliary variables $\boldsymbol{\mathcal{Z}}_1$ and $\boldsymbol{\mathcal{Z}}_2$ whose shapes are identical to $\boldsymbol{\mathcal{W}}$ such that $\boldsymbol{Z}_{1(1)} = \boldsymbol{W}_{(1)}$ and $\boldsymbol{Z}_{2(2)} = \boldsymbol{W}_{(2)}$. Then, with the introduced variables, we rephrase the optimization problem (9) as

$$\begin{aligned} \min_{\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{Z}}_1, \boldsymbol{\mathcal{Z}}_2} \;& \ell(\boldsymbol{\mathcal{W}}) + \|\boldsymbol{Z}_{1(1)}\|_* + \|\boldsymbol{Z}_{2(2)}\|_*, \\ \text{s.t.} \;& \boldsymbol{\mathcal{W}} = \boldsymbol{\mathcal{Z}}_1, \boldsymbol{\mathcal{W}} = \boldsymbol{\mathcal{Z}}_2, \mathcal{P}(R_1, R_2) \le \beta. \end{aligned} \tag{10}$$

Hence, the corresponding augmented Lagrangian with dual multipliers of (10) can be defined as:

$$\begin{aligned} &\mathcal{L}_\lambda(\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{Z}}_1, \boldsymbol{\mathcal{Z}}_2, \boldsymbol{\mathcal{M}}_1, \boldsymbol{\mathcal{M}}_2) = \\ &\quad \ell(\boldsymbol{\mathcal{W}}) + \|\boldsymbol{Z}_{1(1)}\|_* + \|\boldsymbol{Z}_{2(2)}\|_* + \langle \boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}_1, \boldsymbol{\mathcal{M}}_1 \rangle \\ &+ \langle \boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}_2, \boldsymbol{\mathcal{M}}_2 \rangle + \frac{\lambda}{2}\|\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}_1\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}_2\|_F^2, \end{aligned} \tag{11}$$

where $\boldsymbol{\mathcal{M}}_1$ and $\boldsymbol{\mathcal{M}}_2$ are the dual multipliers. Next, we can optimize the problem by separately updating the original and auxiliary variables.

[1] To compress other types of DNNs, e.g., recurrent neural network (RNN), or use other tensor decomposition, e.g., tensor train or tensor ring, similar solution can also be derived under the optimization framework proposed in this paper.

### 5.1 Updating Step for $\mathcal{Z}_1, \mathcal{Z}_2$-Variable

We first describe how to perform minimization of $\mathcal{L}_\lambda$ with respect to the variables $\boldsymbol{\mathcal{Z}}_1$ and $\boldsymbol{\mathcal{Z}}_2$. Considering the terms in Eq. (11) either contain $\boldsymbol{\mathcal{Z}}_1$ or $\boldsymbol{\mathcal{Z}}_2$ and they are independent non-negative functions, thus we can minimize each $\mathcal{L}_\lambda$ with respect to $\boldsymbol{\mathcal{Z}}_1$ and $\boldsymbol{\mathcal{Z}}_2$ separately. Taking the sub-problem of $\boldsymbol{\mathcal{Z}}_1$ as a detailed example, the matrix form of $\boldsymbol{\mathcal{Z}}_1$ along mode-1 is updated by

$$\begin{aligned}\boldsymbol{Z}_{1(1)}^{k+1} =& \underset{\boldsymbol{Z}_{1(1)}}{\operatorname{argmin}} \ \|\boldsymbol{Z}_{1(1)}\|_* + \langle \boldsymbol{W}_{(1)}^k - \boldsymbol{Z}_{1(1)}, \boldsymbol{M}_{1(1)}^k \rangle \\ &+ \frac{\lambda}{2}\|\boldsymbol{W}_{(1)}^k - \boldsymbol{Z}_{1(1)}\|_F^2 \\ =& \underset{\boldsymbol{Z}_{1(1)}}{\operatorname{argmin}} \ \|\boldsymbol{Z}_{1(1)}\|_* \\ &+ \frac{\lambda}{2}\|\boldsymbol{W}_{(1)}^k - \boldsymbol{Z}_{1(1)} + \boldsymbol{M}_{1(1)}^k/\lambda\|_F^2,\end{aligned} \tag{12}$$

where $k$ is the iteration step. According to (Ma, Goldfarb, and Chen 2011), Eq. (12) has the closed-form solution $\mathcal{S}_{\tau_1}(\boldsymbol{W}_{(1)}^k + \boldsymbol{M}_{1(1)}^k/\lambda)$. Hence, $\boldsymbol{\mathcal{Z}}_1$ can be updated by

$$\boldsymbol{\mathcal{Z}}_1^{k+1} = \texttt{fold}_1(\mathcal{S}_{\tau_1}(\boldsymbol{W}_{(1)}^k + \boldsymbol{M}_{1(1)}^k/\lambda)). \tag{13}$$

For updating $\boldsymbol{\mathcal{Z}}_2$, since this procedure is independent from $\boldsymbol{\mathcal{Z}}_1$-update, $\boldsymbol{\mathcal{Z}}_2$ can be updated in parallel using the similar way as follows:

$$\boldsymbol{\mathcal{Z}}_2^{k+1} = \texttt{fold}_2(\mathcal{S}_{\tau_2}(\boldsymbol{W}_{(2)}^k + \boldsymbol{M}_{2(2)}^k/\lambda)). \tag{14}$$

In such updating steps, the automatic determination of the Tucker ranks $R_1$ and $R_2$ is performed via the shrinkage operation $\mathcal{S}(\cdot)$ in Eq. (13) and (14) with $\tau_1, \tau_2$ for the current iteration. To be specific, we first compute the singular value decomposition $\boldsymbol{U}_1 \text{diag}(\boldsymbol{\sigma}_1)\boldsymbol{V}_1^T = \boldsymbol{W}_{(1)}^k + \boldsymbol{M}_{1(1)}^k/\lambda$ and $\boldsymbol{U}_2 \text{diag}(\boldsymbol{\sigma}_2)\boldsymbol{V}_2^T = \boldsymbol{W}_{(2)}^k + \boldsymbol{M}_{2(2)}^k/\lambda$, where the singular values $\sigma_{1i}, \sigma_{2i}$ in $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2$ are sorted in a decreasing order, respectively.

Then we can obtain $\tau_1$ and $\tau_2$ by solving the following simple problem:

$$\begin{aligned}\max_{\boldsymbol{s}_1, \boldsymbol{s}_2 \in \{0,1\}} \quad & \langle \boldsymbol{\sigma}_1, \boldsymbol{s}_1 \rangle + \langle \boldsymbol{\sigma}_2, \boldsymbol{s}_2 \rangle, \\ \text{s.t.} \quad & \mathcal{P}(\|\boldsymbol{s}_1\|_0, \|\boldsymbol{s}_2\|_0) \leq \beta,\end{aligned} \tag{15}$$

where $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ are binary vectors. The lengths of $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ are identical to $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$, respectively, and the number of non-zero elements in $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ are equal to $R_1$ and $R_2$, respectively. Once $\tau_1$ and $\tau_2$ are obtained, $R_1$ and $R_2$ are also correspondingly determined. This problem can be considered as a 0-1 programming problem, which can be efficient solved via greedy algorithm provided by (Kellerer, Pferschy, and Pisinger 2004) with very low computational complexity. To be specific, we can select the largest value in $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ and toggle the corresponding zero entry to one in $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ every time. When $\mathcal{P}(\|\boldsymbol{s}_1\|_0, \|\boldsymbol{s}_2\|_0)$ reaches the budget $\beta$, the algorithm terminates. Hence, we can set $\tau_1$ and $\tau_2$ as the current selected values in $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$. Correspondingly, $R_1$ and $R_2$ can be naturally obtained by

$$R_1 = \max\{i | \sigma_{1i} > \tau_1, i = 1, 2, \cdots\}, \tag{16}$$

$$R_2 = \max\{i | \sigma_{2i} > \tau_2, i = 1, 2, \cdots\}. \tag{17}$$

Algorithm 1: Budget-Aware Training with BC-ADL

**Inputs:** Weight tensor $\boldsymbol{\mathcal{W}}$, learning rate $\alpha$, penalty parameter $\lambda$, number of epoch $T$, target budget $\beta$.
**Output:** Optimized weight tensor $\boldsymbol{\mathcal{W}}$, selected rank $R_1$ and $R_2$.
$\boldsymbol{\mathcal{Z}}_1 \leftarrow \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{Z}}_2 \leftarrow \boldsymbol{\mathcal{W}}$;
$\boldsymbol{\mathcal{M}}_1 \leftarrow \texttt{zeros\_like}(\boldsymbol{\mathcal{W}})$;
$\boldsymbol{\mathcal{M}}_2 \leftarrow \texttt{zeros\_like}(\boldsymbol{\mathcal{W}})$;
**while** $k < T$ **do**
  Obtain $\tau_1$ and $\tau_2$ by solving Eq. (15);
  Determine $R_1$ and $R_2$ using Eq. (16) and (17);
  Update $\boldsymbol{\mathcal{Z}}_1^{k+1}$ using Eq. (13);
  Update $\boldsymbol{\mathcal{Z}}_2^{k+1}$ using Eq. (14);
  Update $\boldsymbol{\mathcal{W}}^{k+1}$ using Eq. (20) with SGD;
  $\boldsymbol{\mathcal{M}}_1^{k+1} \leftarrow \boldsymbol{\mathcal{M}}_1^k + \lambda(\boldsymbol{\mathcal{W}}_1^k - \boldsymbol{\mathcal{Z}}_1^k)$;
  $\boldsymbol{\mathcal{M}}_2^{k+1} \leftarrow \boldsymbol{\mathcal{M}}_2^k + \lambda(\boldsymbol{\mathcal{W}}_2^k - \boldsymbol{\mathcal{Z}}_2^k)$;
**end while**

Upon finding the solution, the singular values that are smaller than $\tau_1$ and $\tau_2$ are truncated.

### 5.2 Updating Step for $\mathcal{W}$-Variable

After determining the tensor ranks and updating $\mathcal{Z}_1, \mathcal{Z}_2$-variable at the current iteration, we then perform update for $\boldsymbol{\mathcal{W}}$ via minimizing $\mathcal{L}_\lambda$ over $\boldsymbol{\mathcal{W}}$. As fixing all variables except $\boldsymbol{\mathcal{W}}$, the minimization of $\mathcal{L}_\lambda(\boldsymbol{\mathcal{W}})$ becomes to minimize a quadratic function:

$$\begin{aligned}\min_{\boldsymbol{\mathcal{W}}} \ & \ell(\boldsymbol{\mathcal{W}}) + \langle \boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}_1^k, \boldsymbol{\mathcal{M}}_1^k \rangle + \langle \boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}_2^k, \boldsymbol{\mathcal{M}}_2^k \rangle \\ &+ \frac{\lambda}{2}\|\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}_1^k\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}_2^k\|_F^2,\end{aligned} \tag{18}$$

which is differentiable. Hence, the gradient of $\mathcal{L}_\lambda(\boldsymbol{\mathcal{W}})$ over $\boldsymbol{\mathcal{W}}$ is given by

$$\begin{aligned}\frac{\partial \mathcal{L}_\lambda(\boldsymbol{\mathcal{W}})}{\partial \boldsymbol{\mathcal{W}}} =& \frac{\partial \ell(\boldsymbol{\mathcal{W}})}{\partial \boldsymbol{\mathcal{W}}} + \frac{\lambda}{2}(\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}_1^k + \boldsymbol{\mathcal{M}}_1^k/\lambda) \\ &+ \frac{\lambda}{2}(\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}_2^k + \boldsymbol{\mathcal{M}}_2^k/\lambda).\end{aligned} \tag{19}$$

Then, $\boldsymbol{\mathcal{W}}$ can be updated by standard stochastic gradient descent (SGD) approach as:

$$\boldsymbol{\mathcal{W}}^{k+1} = \boldsymbol{\mathcal{W}}^k - \alpha \frac{\partial \mathcal{L}_\lambda(\boldsymbol{\mathcal{W}})}{\partial \boldsymbol{\mathcal{W}}}, \tag{20}$$

where $\alpha$ is the learning rate for DNN training.

Once $\mathcal{W}$-variable is updated, another round of $\mathcal{Z}_1, \mathcal{Z}_2$-variable update will be performed based on the latest $\mathcal{W}$-variable. Such iterative update will continue till the end of training procedure. After that, the desired tensor decomposed weight tensor are well trained and obtained. Meanwhile, the finally selected tensor ranks are automatically determined as the $R_1$ and $R_2$ calculated by Eq. (16) (17) at the last iteration. To sum up, the overall procedure of the proposed training procedure with automatic rank selection is summarized in Algorithm 1.

| Model | Compression Method | Rank Selection | Top-1 (%) | # Param.↓ | Top-1 (%) | # Param.↓ |
|---|---|---|---|---|---|---|
| | | | ResNet-20 | | ResNet-32 | |
| Original (He et al. 2016) | - | - | 91.25 | 1.0× | 92.49 | 1.0× |
| Std. Tucker (Kim et al. 2016) | Tucker | Fixed | 87.41 | 2.6× | 87.70 | 5.1× |
| PSTR-M (Li et al. 2020) | Tensor Ring | Genetic Algorithm | 88.50 | 6.8× | 90.6 | 5.8× |
| PSTR-S (Li et al. 2020) | Tensor Ring | Genetic Algorithm | 90.80 | 2.5× | 91.44 | 2.7× |
| **BATUDE (Ours)** | Tucker | BC-ADL | 89.47 | **6.8**× | 91.47 | **5.8**× |
| **BATUDE (Ours)** | Tucker | BC-ADL | **91.02** | 2.6× | **92.18** | 2.8× |

Table 1: Comparison with different tensor decomposition-based approaches for ResNet-20 and ResNet-32 on CIFAR-10.

| Model | Compression Method | Rank Selection | Top-1 (%) | # Param.↓ | Top-1 (%) | # Param.↓ |
|---|---|---|---|---|---|---|
| | | | ResNet-20 | | ResNet-32 | |
| Original (He et al. 2016) | - | - | 65.4 | 1.0× | 68.10 | 1.0× |
| Std. Tucker (Kim et al. 2016) | Tucker | Fixed | 57.53 | 2.5× | 59.03 | 2.5× |
| PSTR-M (Li et al. 2020) | Tensor Ring | Genetic Algorithm | 63.62 | 4.7× | 66.77 | 5.2× |
| PSTR-S (Li et al. 2020) | Tensor Ring | Genetic Algorithm | 66.13 | 2.3× | 68.05 | 2.4× |
| **BATUDE (Ours)** | Tucker | BC-ADL | 64.91 | **4.7**× | 66.96 | **5.2**× |
| **BATUDE (Ours)** | Tucker | BC-ADL | **66.67** | 2.8× | **68.95** | 2.6× |

Table 2: Comparison with different tensor decomposition-based approaches for ResNet-20 and ResNet-32 on CIFAR-100.

# 6 Experiments

We evaluate the proposed BATUDE for various popular CNN models on CIFAR-10, CIFAR-100 and ImageNet datasets.

**Settings.** Learning rate is set as 0.1 and 0.01 for CIFAR and ImageNet dataset, respectively. The learning rate is multiplied by a factor of 0.1 every 30 epochs. Training optimizer is set as momentum SGD. For other hyper-parameters, batch size, weight decay and $\lambda$ are set as 256, 0.0001, and 0.001. Also, the compressed Tucker-format models are fine-tuned with learning rate 0.005 and 0.001 on CIFAR and ImageNet dataset, respectively.

## 6.1 Experimental Results on CIFAR-10

Table 1 shows the evaluation results on CIFAR-10 dataset, For ResNet-20 model, with the same compression ratio of 2.6×, BATUDE outperforms the standard Tucker method using the equal rank setting with 3.61% higher accuracy. Also, compared with the genetic algorithm-based tensor selection scheme (Li et al. 2020), our approach also shows significant performance improvement (almost 1% higher accuracy) with the same compression ratio of 6.8×.

For ResNet-32, our method is also superior to the state of the arts. Compared with (Li et al. 2020) the model trained by our method achieves 0.74% higher accuracy with higher compression ratio of 2.8×. Also, with the same compression ratio 5.8×, our approach brings 0.87% higher accuracy.

## 6.2 Experimental Results on CIFAR-100

Table 2 shows the evaluation results on CIFAR-100 dataset. For ResNet-20, compared with standard Tucker decomposition with equal setting, our approach brings 9.14% accuracy improvement with higher compression ratio. Compared with the state-of-the-art rank selection scheme (Li et al. 2020), BATUDE brings 0.54% higher accuracy with 0.5× higher compression ratio. This performance is even 1.27% higher than the original uncompressed model. When the targeted compression ratio increases to 4.7×, BATUDE still outperforms the state of the art with 1.29% higher accuracy.

For ResNet-32, the proposed BATUDE achieves 68.95% accuracy with compression ratio 2.6×, which enjoys 9.92% and 0.85% higher accuracy than the standard Tucker decomposition and (Li et al. 2020), respectively. Notably, our model provides 0.85% accuracy improvement compared with the original uncompressed model. Also, with the same 5.2× compression ratio, our approach brings 66.96% accuracy; while (Li et al. 2020) only has 66.77% accuracy.

## 6.3 Experimental Results on ImageNet

We also evaluate our BATUDE on ImageNet dataset for compression on ResNet-18 and ResNet-50 model. Table 3 compares the performance of our method with the other tensor decomposition approaches as well as other compression methods, such as pruning and matrix SVD. For ResNet-18, it is seen that BATUDE achieves 89.04% top-5 accuracy with 3.22× floating-point operations per second (FLOPs) reduction, while the state-of-the-art tensor decomposition-based method (Stable EPC (Phan et al. 2020)) has lower accuracy (88.93%) than ours with even lower FLOP reduction (3.09×). Moreover, for our compressed model with 2.52× FLOPs reduction, BATUDE can even achieve 0.33% top-5 accuracy increase over the original baseline model.

For ResNet-50, BATUDE also shows improved performance over the state-of-the-art approaches. With 2.97× FLOPs reduction, our compressed model achieves 93.24% top-5 accuracy, which is 0.37% higher than the uncompressed model. With 2.04× FLOPs reduction, BATUDE brings 0.55% more accuracy over the uncompressed model.

| Model | Compression Method | Budget-Aware | Top-5 (%) | FLOPs↓ | Top-5 (%) | FLOPs↓ |
|---|---|---|---|---|---|---|
| | | | ResNet-18 | | ResNet-50 | |
| Torchvision | - | - | 89.08 | 1.00× | 92.87 | 1.00× |
| TRP (Xu et al. 2020) | Low-Rank Matrix | ✗ | 86.74 | 2.60× | 92.07 | 1.80× |
| VCNNP (Zhao et al. 2019) | Channel Pruning | ✗ | - | - | 92.10 | 1.67× |
| FPGM (He et al. 2019) | Filter Pruning | ✗ | 88.53 | 1.72× | 92.63 | 1.73× |
| HRank (Lin et al. 2020) | Filter Pruning | ✗ | - | - | 92.33 | 1.78× |
| DSA (Ning et al. 2020) | Budgeted Pruning | ✓ | 88.35 | 1.72× | 92.06 | 2.00× |
| Std. Tucker (Kim et al. 2016) | Tucker | ✗ | 87.53 | 2.25× | 91.12 | 2.04× |
| MUSCO (Gusak et al. 2019) | Tucker | ✗ | 88.78 | 2.42× | - | - |
| Stable EPC (Phan et al. 2020) | Tucker-CP | ✗ | 88.93 | 3.09× | 92.16 | 2.64× |
| **BATUDE (Ours)** | Tucker | ✓ | 89.04 | **3.22**× | 93.24 | **2.97**× |
| **BATUDE (Ours)** | Tucker | ✓ | **89.41** | 2.52× | **93.42** | 2.04× |

Table 3: Comparison with multiple compression approaches for ResNet-18 and ResNet-50 on ImageNet dataset. (Li et al. 2020) does not report results on ImageNet.

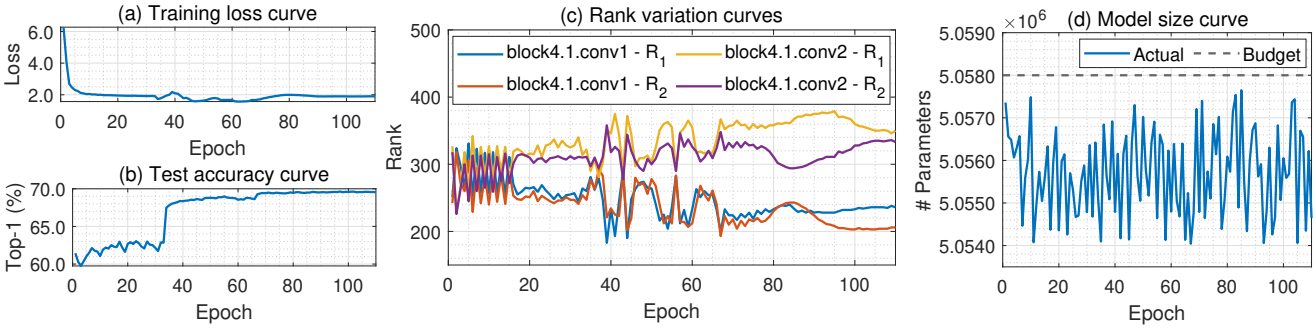


Figure 2: The change of (a) training loss and (b) Top-1 test accuracy, (c) Rank variations for two example component convolutional layer during training and (d) number of parameters in the training process for ResNet-18 on ImageNet dataset. The compression budget is set as $5.058 \times 10^6$ parameters.

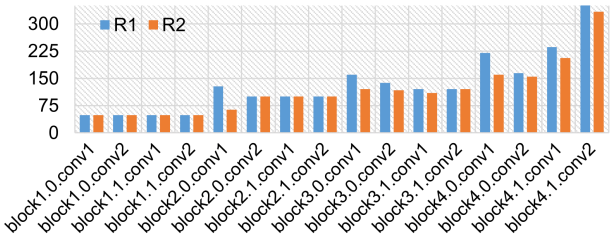


Figure 3: Final rank distribution using BATUDE for compressing ResNet-18 on ImageNet dataset.

### 6.4 Experimental Analysis & Discussion

To obtain deep understanding of the effect of the proposed BATUDE, we also analyze the training procedure for ResNet-18 compression on ImageNet dataset. Fig. 2 illiterates the curves of training loss, top-1 test accuracy and overall number of parameters with respect to training epochs. Here as shown in Fig. 2a and Fig. 2b, both training loss and top-1 accuracy converge well, thereby demonstrating that the training process with BC-ADL enjoys very good convergence characteristic – the optimization process finally reaches a good point where not only the loss function term is very small but also the tensor nuclear norm term in Eq. (7) satisfies the budget constraint. In other words, it implies that BATUDE successfully maximizes the representative ability of the compressed model in a fixed size via learning the optimal tensor ranks from data, as we target in Section 4.

In addition, Fig. 2c and Fig. 3 shows the tensor rank change for two example layers and the final rank distribution for the overall compressed model. As shown in Fig. 2c and Fig. 2d, we can observe that the tensor ranks are dynamically adjusted as the epoch increases and finally converge to constant values; meanwhile, the overall number of parameters is kept very close to the pre-set budget during the entire training process. This indicates that our BATUDE indeed can gradually impose low-rank properties on the DNN models and simultaneously satisfy the target compression demand.

## 7 Conclusion

In this paper, we propose BATUDE, a budget-aware DNN compression approach based on Tucker Decomposition with automatic tensor rank selection. By integrating the rank selection to the training procedure, we enable the decomposed DNN models to automatically learn the suitable rank from data. Evaluation on different models on various datasets show that our approach significantly outperforms the state-of-the-art DNN model compression approaches with much higher compression ratio and higher classification accuracy.

## Acknowledgements

This work was partially supported by National Science Foundation under Grant CCF-1955909.

## References

Cai, J.-F.; Candès, E. J.; and Shen, Z. 2010. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4): 1956–1982.

Chen, Y.-H.; Emer, J.; and Sze, V. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 44(3): 367–379.

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.

Deng, C.; Sun, F.; Qian, X.; Lin, J.; Wang, Z.; and Yuan, B. 2019a. TIE: energy-efficient tensor train-based inference engine for deep neural network. In *Proceedings of the 46th International Symposium on Computer Architecture*, 264–278.

Deng, C.; Yin, M.; Liu, X.-Y.; Wang, X.; and Yuan, B. 2019b. High-performance Hardware Architecture for Tensor Singular Value Decomposition. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1–6. IEEE.

Eckstein, J.; and Bertsekas, D. P. 1992. On the Douglas—Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1): 293–318.

Gabay, D.; and Mercier, B. 1976. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & mathematics with applications*, 2(1): 17–40.

Gandy, S.; Recht, B.; and Yamada, I. 2011. Tensor completion and low-n-rank tensor recovery via convex optimization. *Inverse Problems*, 27(2): 025010.

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

Girshick, R. 2015. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, 1440–1448.

Gong, R.; Liu, X.; Jiang, S.; Li, T.; Hu, P.; Lin, J.; Yu, F.; and Yan, J. 2019. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 4852–4861.

Gusak, J.; Kholiavchenko, M.; Ponomarev, E.; Markeeva, L.; Blagoveschensky, P.; Cichocki, A.; and Oseledets, I. 2019. Automated multi-stage compression of neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 0–0.

Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M. A.; and Dally, W. J. 2016. EIE: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3): 243–254.

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. 2015. Learning both weights and connections for efficient neural network. *Advances in Neural Information Processing Systems*, 28: 1135–1143.

Harshman, R. A.; et al. 1970. Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multimodal factor analysis.

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

He, Y.; Liu, P.; Wang, Z.; Hu, Z.; and Yang, Y. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4340–4349.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 1389–1397.

Hillar, C. J.; and Lim, L.-H. 2013. Most tensor problems are NP-hard. *Journal of the ACM (JACM)*, 60(6): 1–39.

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2704–2713.

Kellerer, H.; Pferschy, U.; and Pisinger, D. 2004. Multidimensional knapsack problems. In *Knapsack problems*, 235–283. Springer.

Kim, Y.-D.; 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*.

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

Li, N.; Pan, Y.; Chen, Y.; Ding, Z.; Zhao, D.; and Xu, Z. 2020. Heuristic Rank Selection with Progressively Searching Tensor Ring Network. *arXiv preprint arXiv:2009.10580*.

Lin, M.; Ji, R.; Wang, Y.; Zhang, Y.; Zhang, B.; Tian, Y.; and Shao, L. 2020. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1529–1538.

Liu, J.; Musialski, P.; Wonka, P.; and Ye, J. 2012. Tensor completion for estimating missing values in visual data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1): 208–220.

Luo, J.-H.; Wu, J.; and Lin, W. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, 5058–5066.

Ma, S.; Goldfarb, D.; and Chen, L. 2011. Fixed point and Bregman iterative methods for matrix rank minimization. *Mathematical Programming*, 128(1-2): 321–353.

Ning, X.; Zhao, T.; Li, W.; Lei, P.; Wang, Y.; and Yang, H. 2020. DSA: More Efficient Budgeted Pruning via Differentiable Sparsity Allocation. In *European Conference on Computer Vision*.

Novikov, A.; Podoprikhin, D.; Osokin, A.; and Vetrov, D. P. 2015. Tensorizing neural networks. *Advances in Neural Information Processing Systems*, 28: 442–450.

Pan, Y.; Xu, J.; Wang, M.; Ye, J.; Wang, F.; Bai, K.; and Xu, Z. 2019. Compressing recurrent neural networks with tensor ring for action recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4683–4690.

Peng, H.; Wu, J.; Chen, S.; and Huang, J. 2019. Collaborative channel pruning for deep networks. In *International Conference on Machine Learning*, 5113–5122.

Phan, A.-H.; Sobolev, K.; Sozykin, K.; Ermilov, D.; Gusak, J.; Tichavskỳ, P.; Glukhov, V.; Oseledets, I.; and Cichocki, A. 2020. Stable low-rank tensor decomposition for compression of convolutional neural network. In *European Conference on Computer Vision*, 522–539. Springer.

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. Springer.

Sui, Y.; Yin, M.; Xie, Y.; Phan, H.; Aliari Zonouz, S.; and Yuan, B. 2021. CHIP: CHannel Independence-based Pruning for Compact Neural Networks. *Advances in Neural Information Processing Systems*, 34.

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

Wang, W.; Sun, Y.; Eriksson, B.; Wang, W.; and Aggarwal, V. 2018. Wide compression: Tensor ring nets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9329–9338.

Xu, K.; Ba, J.; Kiros, R.; Cho, K.; Courville, A.; Salakhudinov, R.; Zemel, R.; and Bengio, Y. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, 2048–2057.

Xu, Y.; Li, Y.; Zhang, S.; Wen, W.; Wang, B.; Qi, Y.; Chen, Y.; Lin, W.; and Xiong, H. 2020. TRP: Trained Rank Pruning for Efficient Deep Neural Networks. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 977–983.

Yang, Y.; Krompass, D.; and Tresp, V. 2017. Tensor-Train Recurrent Neural Networks for Video Classification. In *International Conference on Machine Learning*, 3891–3900.

Yin, M.; Liao, S.; Liu, X.-Y.; Wang, X.; and Yuan, B. 2021a. Towards Extremely Compact RNNs for Video Recognition with Fully Decomposed Hierarchical Tucker Structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12085–12094.

Yin, M.; Sui, Y.; Liao, S.; and Yuan, B. 2021b. Towards Efficient Tensor Decomposition-Based DNN Model Compression With Optimization Framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10674–10683.

Zhang, T.; Ye, S.; Zhang, K.; Tang, J.; Wen, W.; Fardad, M.; and Wang, Y. 2018. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *European Conference on Computer Vision*, 184–199.

Zhao, C.; Ni, B.; Zhang, J.; Zhao, Q.; Zhang, W.; and Tian, Q. 2019. Variational convolutional neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2780–2789.

Zhuang, Z.; Tan, M.; Zhuang, B.; Liu, J.; Guo, Y.; Wu, Q.; Huang, J.; and Zhu, J. 2018. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, 875–886.