University of California
Santa Barbara

Low-Rank Tensorized Neural Networks With Tensor Geometry Optimization

A thesis submitted in partial satisfaction
of the requirements for the degree

Master of Science
in
Electrical and Computer Engineering

by

Ryan Solgi

Committee in charge:

Professor Zheng Zhang, Chair
Professor Ramtin Pedarsani
Professor Kenneth Rose

June 2024

The thesis of Ryan Solgi is approved.

Professor Ramtin Pedarsani

Professor Kenneth Rose

Professor Zheng Zhang, Committee Chair

March 2024

Low-Rank Tensorized Neural Networks With Tensor Geometry Optimization

by

Ryan Solgi

Abstract

Low-Rank Tensorized Neural Networks With Tensor Geometry Optimization

by

Ryan Solgi

Deep neural networks have demonstrated significant achievements across various fields, yet their memory and time complexities present obstacles for implementing them on resource-constrained devices. Compressing deep neural networks using tensor decomposition can decrease both memory usage and computational costs. The performance of a low-rank tensorized network depends on the choices of hyperparameters including the tensor rank and geometry. Previous studies have concentrated on identifying optimal tensor ranks. This thesis studies the effect of tensor geometry used for folding data for low-rank tensor compression. It is demonstrated that tensor geometry significantly affects compression efficiency of the tensorized data and model parameters. Consequently, a novel mathematical formulation is developed to optimize tensor geometry. The tensor geometry optimization model is adopted for efficient deployment of low-rank neural networks. The presented tensor geometry optimization model is combinatorial and thus challenging to solve. Therefore, surrogate and relaxed versions of the model are developed and various methods including integer linear programming, graph optimization, and random search algorithms are applied to solve the presented optimization model. The proposed tensor geometry optimization achieved a notable reduction in both the memory and time complexities of neural networks while maintaining accuracy. The developed methods can be applied for hardware-software co-design of artificial intelligence (AI) accelerators particularly on resource-constrained devices.

Contents

Chapter 1

Introduction

1.1 Motivation

Tensor decomposition methods have been successfully employed for compressing and accelerating various neural networks [1, 2, 3, 4, 5, 6, 7, 8, 9]. Despite the success of tensor compression methods, they pose the design challenge of determining hyper-parameters such as the tensor rank and geometry to achieve the best trade-off between efficiency and accuracy. In real-world implementations of tensor compression, determining these hyper-parameters can be challenging. There have been some recent works that studied the tensor rank determination problem for tensor compression [7, 10, 11, 12, 13]. However, the study of the effect of tensor geometry on tensor decomposition has been rarely studied in the literature [14, 15].

Tensor decomposition is exclusively applicable to data or parameters with three or more dimensions. However, this criterion may not always be met. One example is a 2-dimensional array (matrix) representing the weights of a fully connected layer in a neural network. When confronted with such layers, tensorization becomes necessary, meaning the weight matrix is folded into a higher-dimensional array. The folding step

raises questions about the geometry used for folding. Firstly, does the chosen geometry impact compression efficiency, and if so, to what extent? Secondly, how can we leverage tensor geometry to enhance the efficiency of tensor compression?

1.2 Summary of contributions

This thesis investigates the effect of tensor geometry on tensor compression efficiency. A novel tensor geometry optimization paradigm is introduced. The developed optimization paradigm maximizes the space saving of the tensor compression with respect to the tensor geometry. The optimization model is studied for both data and model compression. The tensor geometry optimization paradigm is also adopted for low-rank tensorized neural networks. Various formulations of the developed optimization model are solved using random search, integer linear programming and graph traversal algorithms. It is also demonstrated that a balanced geometry is an relaxed upper bound solution for the defined tensor geometry optimization paradigm. Furthermore, to generalize the proposed method for various neural network architectures, a unifying framework called low-rank tensorized product is introduced that accelerates the implementation of low-rank tensorized neural networks and a time complexity analysis of low-rank tensorized neural networks is conducted. The results of this study can be used for hardware-software co-design of AI accelerators, particularly on resource-constrained devices, to accelerate execution and reduce memory footprint of neural networks.

1.3 Outline

The first chapter of this thesis empirically studies the effect of the shape of a tensor in the compression of tensorized signals and neural networks. The study is narrowed

down to the tensor train decomposition. The task of finding the optimum shape for the tensor train decomposition is formulated as an optimization model that maximizes the space saving with respect to the geometry of a given tensor subject to an error bound. The capability of the proposed method is studied by compressing images and a neural network. The results demonstrates that the tensor shape has a significant effect in the compression efficiency of the tensorized data and neural networks. Therefore, the proposed optimization paradigm can be applied to utilize the shape effect for enhancing the efficiency of both data and model compression using the tensor train decomposition.

In the second chapter the tensor geometry optimization paradigm is adopted for end-to-end low-rank tensorized neural networks. The proposed optimization model is modified to address not only the compression efficiency but also the computational cost and implementation efficiency of neural networks. The proposed optimization model is addressed using different optimization methods, including integer linear programming, graph traversal, and random search algorithms. In addition to tensor train decomposition, the optimization models are also developed for Tucker decomposition. In this chapter it is demonstrated that a potential balanced geometry is an upper bound solution for the defined tensor geometry optimization paradigm. The proposed methods are studied for both dense and convolutional neural networks.

1.4 Permissions and Attributions

1. The content of chapter 1 has previously appeared in Applied Soft Computing [16].

Chapter 2

Tensor Geometry and Tensor Compression

2.1 Introduction

Processing high-dimensional data is a necessity across various disciplines. Different tensor decomposition methods have been proposed for high dimensional data analysis [17, 18]. A tensor is usually defined as a high-dimensional array (i.e., an array of three or more dimensions). For instance, red, green, blue (RGB), or hyperspectral images are examples of tensors. A tensor may also represent a model where the parameters of the model are a multi-way array. For example, the parameters of a deep neural network can be represented as a tensor. Tensor decomposition (e.g. the tensor train decomposition) refers to factorizing a tensor (i.e. a high dimensional array) to a low-rank factor space. Tensor decomposition is functional in dimensionality reduction or data and models compression.

Compressing big data via tensor decomposition has gained great success in recent years. For instance, using tensor decomposition, a massive amount of data with millions of elements can be decomposed to its factors that might be of the order of thousands

reducing the required storage space significantly. This can be used for compressing both raw data and model parameters. The tensor decomposition methods have been applied to approximate high dimensional problems in different domains, including data-mining and knowledge discovery, dimensionality reduction, scientific computation, machine learning, and signal processing [19, 20, 21, 22, 11, 23, 24]. Different methods have been successfully applied to decompose a higher-order tensor to low-dimensional parameters, including the CANDECOMP/PARAFAC (CP) decomposition [25], the Tucker decomposition [26], and the tensor train (TT) decomposition [27]. Tensor decomposition was also extended to a more general form called tensor networks, which leads to various decomposition formats [28]. In recent years Bayesian methods have also been developed for automatic rank determination in various tensor problems, including tensor completion and tensorized neural network training [29, 7, 10].

Regardless of the specific choice of a tensor decomposition method the data or the model parameters are represented as a d-way tensor prior to the decomposition. Sometimes, the given data has a high dimensional format and it is important that the original shape of the data be preserved. However, there are many cases where the original data is of a lower dimension (e.g. one-dimensional or two-dimensional arrays) and the data have to be folded (rearranged) to be presented as a tensor prior to tensor decomposition or the shape of data can be changed as long as an invertible mapping be applied. Such cases often involve a reshaping step that changes a tensor's dimension and mode size using a bijective mapping. The shape of a tensor affects the rank and, subsequently, the accuracy and compression efficiency (i.e., space saving and compression ratio) of the subsequent tensor decomposition. Consequently, one may ask what shape or mode size should be used for a given data for tensor decomposition. Despite the importance of finding an optimum geometry for tensor decomposition, studies on this domain remain very sparse. This empirical study attempts to answer the aforementioned question.

This chapter investigates the effect of the tensor geometry on tensor compression and proposes an optimization model that maximizes the space saving with respect to the shape and order of the tensor. The applied reshaping method is a bijection that allows the original data with their characteristics to be retrieved. The study is narrowed down to the TT decomposition, but the proposed technique can be extended to other tensor decomposition methods. A genetic algorithm (GA) is presented for solving the optimization problem. The proposed method is applied to compress RGB images and a neural network to study its performance. The results of the optimization model are compared with random shapes to demonstrate the effectiveness of the proposed method.

2.2 Related Works

2.2.1 Tensor Decomposition and Applications

A detailed review of tensor decomposition and its application in different applications such as data mining and knowledge discovery, signal processing, computer vision, scientific computing, and neuroscience is provided in [19]. Applications of tensor decomposition for data mining were reviewed in previous studies [30]. Memory efficient Tucker (MET) decomposition was proposed for data mining of sparse multi-way data [31]. Tensor decomposition was used for text mining [32]. A tensor decomposition-based machine learning approach was developed and applied for health data mining [33]. Furthermore, tensor decomposition and representation have been applied for uncertainty quantification [12, 20, 21, 22], and high dimensional data recovery [34, 29, 35] and imaging [36, 37], quantum simulation and computation simulation [38, 39, 40, 41], to name a few.

Tensor decomposition has been recently shown to be promising for model parameters compression in machine learning [4]. For instance, tensor decomposition has been applied

for running compressed convolutional neural networks (CNNs) on mobile devices [3]. The aforementioned study demonstrated that a significant memory storage reduction and energy usage could be achieved while compressing various CNN architectures using the Tucker decomposition [3]. Also, the CP decomposition was used to compress kernels of CNNs which resulted in a significant speedup of the run time of the studied networks with negligible drop in accuracy [2]. Compression of fully connected neural networks using tensor decomposition was studied by [1]. In another study, tensor decomposition was applied to study the generalizability of neural networks [42]. Tensor ring network was proposed by [43] in which neural networks were compressed using the Tensor Ring decomposition.

Among the tensor decomposition formats, tensor train [27] is one of the most popular ones. Due to its great power of representing high dimensional data it has been widely applied for various applications including radar data [44], hyperspectral imaging [36], neural architecture search [45], deep learning model compression [8, 6, 46], quantum dynamics simulation [41], and quantum computation simulation [39, 40].

2.2.2 Tensor Decomposition and Hyperparameter Tuning

In many real-world applications deciding about some hyper-parameters (such as tensor ranks and tensor geometry) of tensor decomposition can be challenging. There have been some recent studies that addressed the tensor rank determination problem. The recent works [7, 10] determined the tensor ranks automatically in neural network training, enabling on-device training of neural networks with limited computing resources [11]. A tensor regression method was proposed for automatic rank determination and applied for uncertainty quantification [12]. Bayesian tensor decomposition was also applied to automatic rank determination for tensor completion and dimension reduction [29, 35].

However, the study of the effect of the shape on tensor decomposition has rarely been reported in the literature. In a previous study [47] we applied an evolutionary tensor shape search for remotely sensed hyperspectral data compression. The present study generalizes the tensor shape search formulation, apply it to RGB images and neural networks, and compare the results of the GA with a random search (RS). The primarily goal is to investigate how reshaping may affect the result of tensor compression and how an optimal shape can be found if there exists one.

2.2.3 Evolutionary Algorithms

The origin of evolutionary computation dates back to the mid 1950s when it was applied in mathematical programming, machine learning, and industrial manufacturing and notably the invention of evolutionary strategies (ES), evolutionary programming (EP), and genetic algorithms (GAs) [48]. The early version of the genetic algorithm (GA) was presented by [49]. Over the past years, variations of evolutionary algorithms (i.e., GAs) have been developed and have been extensively applied to solve problems in various fields where the problems were not approachable with other optimization methods [50, 51, 52, 53]. A wide range of evolutionary algorithms, including GAs and their applications in engineering domains, have been studied in the literature [54]. Particularly, [55] applied an evolutionary algorithm to find optimal hyperparameters of the singular value decomposition for the neural network compression.

2.2.4 Evolutionary Algorithms and Tensor Decomposition

A study at the intersection of evolutionary algorithms and tensor decompositions proposed the application of tensor decomposition-based mutation to the neuroevolution of augmenting topologies (NEAT) algorithm [56]. The CP decomposition was applied to

reduce the dimensionality of solutions to solve high-dimensional optimization problems with evolutionary algorithms [57]. A study formulated the CP decomposition of non-negative tensors as a stochastic problem and solved it using an evolutionary algorithm [58]. Also, an evolutionary search was applied to determine an optimum tensor network topology [28].

2.3 Background

Throughout this chapter capital calligraphic letters (e.g., $\mathcal{A}$) are used to denote tensors, boldface capital letters (e.g., $\mathbf{A}$) are used for matrices, boldface lower case letters (e.g., $\boldsymbol{a}$) are used for vectors, and Roman (e.g., a) or Greek (e.g., α) letters are used for scalars. $[\mathcal{A}]_{ijk}$ refers to element ijk of tensor $\mathcal{A}$.

2.3.1 Tensor Geometry and Reshaping

An order-k (k-way) tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_k}$ denotes a k-dimensional data array. The order of a tensor is the number of its dimensions. The geometry of a tensor determines the order and the number of elements of each dimension. Throughout the chapter, $\boldsymbol{\theta} = (\theta_1, \theta_2, \cdots, \theta_d)$ specifies the geometry of a tensor, where $\theta_j \in \mathbb{N}$ is the size of dimension j and d is the order.

Reshaping refers to changing the order and the number of elements of each dimension. For example, a k-way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_k}$ may be reshaped to a d-way tensor like $\mathcal{Y} \in \mathbb{R}^{\theta_1 \times \cdots \times \theta_d}$. Reshaping a tensor may change its size if the size of $\mathcal{Y}$ is greater than that of $\mathcal{X}$ ($\prod_{j=1}^{d} \theta_j > \prod_{j=1}^{k} I_j$) then dummy elements (e.g., zeros) are entered to fill the gap.

Throughout this chapter two different functions are applied for reshaping: (1) reshape$(\mathcal{X}, \boldsymbol{\theta})$ is used when reshaping does not change the size, and (2) $\Phi(\mathcal{X}, \boldsymbol{\theta})$ is used to denote reshaping a given tensor $\mathcal{X}$ to a new shape $\boldsymbol{\theta}$ if reshaping may change the size. Note that

both reshaping functions are invertible mappings. This work applies a C-like index ordering for reshaping functions where the greater the axis index is, the higher the priority of reordering. Function Φ fills the reshaped tensor with zeros if its size is larger than that of the original tensor.

2.3.2 Tensor Train (TT) Decomposition

In the tensor train (TT) format [27] a d-way tensor $\mathcal{Y} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is approximated with a set of d cores $\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_d$ where $\mathcal{G}_j \in \mathbb{R}^{r_{j-1} \times n_j \times r_j}$, r_j's for $j = 1, \cdots, d-1$ are the ranks, $r_0 = r_d = 1$, and each element of $\mathcal{Y}$ is approximated by Eq.(3.3).

$$[\hat{\mathcal{Y}}]_{i_1,\cdots,i_d} = \sum_{l_0,\cdots,l_d} [\mathcal{G}_1]_{l_0,i_1,l_1} [\mathcal{G}_2]_{l_1,i_2,l_2} \cdots [\mathcal{G}_d]_{l_{d-1},i_d,l_d} \tag{2.1}$$

Figure 2.1 depicts the TT format. Given an error bound ($\epsilon = \frac{\|\mathcal{Y}-\hat{\mathcal{Y}}\|_F}{\|\mathcal{Y}\|_F}$), the core factors, $\mathcal{G}_j$'s, are computed using $(d-1)$ sequential singular value decomposition (SVD) of the auxiliary matrices formed by unfolding tensor $\mathcal{Y}$ along different axes. This decomposition process, which is called the TT-SVD is presented in Algorithm 1.

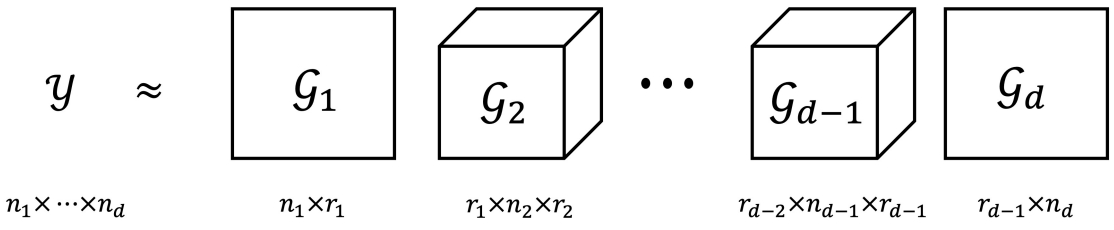

Figure 2.1: A schematic of the TT format.

This work applies the proposed tensor shape search to the TT-SVD. However, it is

Algorithm 1 TT-SVD

Require: d-way tensor $\mathcal{Y}$, error bound ϵ.

$\sigma = \frac{\epsilon}{d-1}\|\mathcal{Y}\|_F$
$r_0 = 1$
$r_d = 1$
$\mathbf{W} = \text{reshape}(\mathcal{Y}, (n_1, \frac{|\mathcal{Y}|}{n_1}))$
for $j = 1$ to $j = d-1$ **do**
    $\mathbf{W} = \text{reshape}(\mathbf{W}, (r_{j-1}n_j, \frac{|\mathbf{W}|}{r_{j-1}n_j}))$
    Compute σ-truncated SVD: $\mathbf{W} = \mathbf{U}\mathbf{S}\mathbf{V}^T + \mathbf{E}$, where $\|\mathbf{E}\|_F \leq \sigma$
    r_j = the rank of matrix $\mathbf{W}$ based on σ-truncated SVD
    $\mathcal{G}_j = \text{reshape}(\mathbf{U}, (r_{j-1}, n_j, r_j))$
    $\mathbf{W} = \mathbf{S}\mathbf{V}^T$
end for
$\mathcal{G}_d = \text{reshape}(\mathbf{W}, (r_{d-1}, n_d, r_d))$
Return $\bar{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_d\}$

possible to extend this framework to other tensor decomposition methods such as the Tucker decomposition, and generally to the tensor networks.

2.4 Problem Statement

The current study proposes a search algorithm to find a geometry that maximizes the space saving of the compression using the TT decomposition. The TT decomposition is used for big data compression and dimensionality reduction. Representing a given tensor $\mathcal{Y} \in \mathbb{R}^{\theta_1 \times \cdots \times \theta_d}$ in the explicit original format requires $\prod_{j=1}^{d} \theta_j$ elements to be stored. However, the TT format requires $\sum_{j=1}^{d} r_{j-1} \times \theta_j \times r_j$ parameters to be stored. We can use the TT factors as an estimation of the original tensor by applying Eq.(3.3). The efficiency of the compression depends on the value of the ranks of the TT format. The space saving is significant when ranks are small. In real world applications high order data usually have low ranks that make compression using TT format to be functional.

One application of the proposed method is changing the order of data to facilitate the application of tensor decomposition (i.e., the TT compression). In practice, there exist

plenty of big data that are in the form of vectors and matrices, and they are not primarily high dimensional. Applying the TT decomposition on vectors results in no compression, and applying the TT decomposition on matrices results in a plain SVD decomposition that limits compression capability [1]. Therefore, the application of the TT format on 1D (i.e., vectors) and 2D (i.e., matrices) arrays requires folding the given data to a higher dimension (i.e., 3D or more) prior to decomposition. The aforementioned bottleneck can be addressed by the proposed tensor geometry search. Besides, this study empirically demonstrates that reshaping may improve compression efficiency even without changing the order, which extends the application of the proposed method for data arrays that are already of dimension three and higher. Therefore the proposed method is formulated for a general tensor with an arbitrary dimension.

Let $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_k}$ be the original data given to be compressed using the TT decomposition and $\hat{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_k}$ is the approximation of the given $\mathcal{X}$ using the TT format. For example, $\mathcal{X}$ can be an RGB image where $k = 3$. To compress the given data first reshape the given $\mathcal{X}$ into a d-way tensor (usually $d \geq k$) like $\mathcal{Y}_{\boldsymbol{\theta}} \in \mathbb{R}^{\theta_1 \times \cdots \times \theta_d}$ as shown below.

$$\mathcal{Y}_{\boldsymbol{\theta}} = \Phi(\mathcal{X}, \boldsymbol{\theta}) \tag{2.2}$$

where $\boldsymbol{\theta} = (\theta_1, \theta_2, \cdots, \theta_d)$ refers to the new shape. Function $\Phi(\mathcal{X}, \boldsymbol{\theta})$ reshapes the given tensor $\mathcal{X}$ to the new shape $\boldsymbol{\theta}$ and enter zero values (dummy elements) if $\prod_{j=1}^{d} \theta_j > \prod_{j=1}^{k} I_j$ to fill the rest of the reshaped tensor. Next, $\mathcal{Y}_{\boldsymbol{\theta}}$ is approximated using the TT decomposition where $\hat{\mathcal{Y}}_{\boldsymbol{\theta}} \in \mathbb{R}^{\theta_1 \times \cdots \times \theta_d}$ is the approximation of $\mathcal{Y}_{\boldsymbol{\theta}}$ using the TT-SVD algorithm 1. Note that there exist a bijection between $\mathcal{X}$ and $\hat{\mathcal{Y}}_{\boldsymbol{\theta}}$ that allows elements of $\hat{\mathcal{X}}$ to be accessed directly from $\hat{\mathcal{Y}}_{\boldsymbol{\theta}}$ as shown below:

$$\hat{\mathcal{X}} = \Phi^{-1}(\hat{\mathcal{Y}}_{\boldsymbol{\theta}}) \tag{2.3}$$

where Φ^{-1} refers to the inverse of reshaping function Φ that consists of reshaping and removing the added dummy elements.

Considering a reshaping stage before applying the TT decomposition on a given data array like $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_k}$ and $\boldsymbol{\theta} = (\theta_1, \theta_2, \cdots, \theta_d)$, space-saving of the TT format using the shape $\boldsymbol{\theta}$ is defined as shown below.

$$C(\boldsymbol{\theta}) = 1 - \frac{\sum_{j=1}^{d} r_{j-1}^{(\boldsymbol{\theta})} \times \theta_j \times r_j^{(\boldsymbol{\theta})}}{\prod_{j=1}^{k} I_j} \tag{2.4}$$

where θ_j refers to the size of dimension j of reshaped tensor $\hat{\mathcal{Y}}_{\boldsymbol{\theta}} \in \mathbb{R}^{\theta_1 \times \cdots \times \theta_d}$ and $r_j^{(\boldsymbol{\theta})}$ refers to the ranks of TT decomposition of reshaped tensor $\mathcal{Y}_{\boldsymbol{\theta}}$. I_j refers to the size of dimension j of original data array $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_k}$. In other words, to calculate the space-saving the size of the factor cores resulting from the decomposition of a reshaped tensor is compared with the size of the original tensor. The ratio of the size of the original data to the size of compressed factors is defined as the compression ratio. Given the space-saving of a shape θ the compression ratio is defined as shown below.

$$R(\boldsymbol{\theta}) = (1 - C(\boldsymbol{\theta}))^{-1} \tag{2.5}$$

where $R(\boldsymbol{\theta})$ refers to the compression ratio of the shape $\boldsymbol{\theta}$.

2.5 Methodology: Tensor Geometry Optimization

This study proposes a tensor geometry search for data compression using the TT decomposition. As described above given a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_k}$, $\mathcal{X}$ can be reshaped to a tensor $\mathcal{Y}_{\boldsymbol{\theta}} \in \mathbb{R}^{\theta_1 \times \cdots \times \theta_d}$. Instead of $\mathcal{X}$, $\mathcal{Y}_{\boldsymbol{\theta}}$ is decomposed and its factors are stored. To retrieve $\hat{\mathcal{X}}$, first $\hat{\mathcal{Y}}_{\boldsymbol{\theta}}$ is reconstructed using factors of $\mathcal{Y}_{\boldsymbol{\theta}}$ and elements of $\hat{\mathcal{X}}$ can be accessed directly from bijection between $\mathcal{Y}_{\boldsymbol{\theta}}$ and $\hat{\mathcal{X}}$ by Eq.(2.3). This work proposes an optimization model to maximize the space saving by the TT decomposition with respect to the tensor shape $\boldsymbol{\theta}$.

Given d (the order of $\mathcal{Y}$), $\boldsymbol{\theta} = (\theta_1, \theta_2, \cdots, \theta_d)$ is a possible shape; and let Θ be the space made of all possible $\boldsymbol{\theta}$'s such that $\theta_i \in \mathbb{N}$ and $l \leq \theta_i \leq u$ for $i = 1, 2, .., d$ and $l, u \in \mathbb{N}$. If $l = 1$, d is the maximal order because when $n_i = 1$ dimension i becomes ineffective, practically. The proposed optimization model maximizes the space saving of the TT decomposition as defined below.

$$\min_{\forall \boldsymbol{\theta} \in \Theta} f(\boldsymbol{\theta}) = \sum_{j=1}^{d} r_{j-1}^{(\boldsymbol{\theta})} \times \theta_j \times r_j^{(\boldsymbol{\theta})}$$

subject to

$$\prod_{j=1}^{d} \theta_j \geq S \tag{2.6}$$

where $S = \prod_{j=1}^{k} I_j$ and $C(\boldsymbol{\theta}) = 1 - \frac{f(\boldsymbol{\theta})}{S}$ maximizes for the minimum of $f(\boldsymbol{\theta})$. Given an error bound ϵ, $r_j^{(\boldsymbol{\theta})}$s are derived from the TT-SVD algorithm based on shape . The upper limit of the $C(\boldsymbol{\theta})$ is 1. When $0 < C(\boldsymbol{\theta}) < 1$ the size of the factors is less than that of the data, but when $C(\boldsymbol{\theta}) \leq 0$ the memory requirement is inflated, and there is no data compression.

Any geometry that results in a tensor $\mathcal{Y}_{\boldsymbol{\theta}}$ whose size is smaller than the size of the

original given data $\mathcal{X}$ is infeasible because some data is missed. Furthermore, the resized tensor is filled with dummy elements (e.g., zeros) when a possible $\boldsymbol{\theta}$ results in a tensor whose size is greater than that of the original data. Any shape which results in an unnecessarily large size is undesirable because it makes the compression less efficient. The objective function defined in Eq.(2.6) maximizes the space saving considering the effect of the added dummy elements. Therefore, the objective function guides the search toward a shape whose size is the closest to that of the data. The definition of the feasible subspace Θ prevents shapes that have a size smaller than that of the original tensor.

Let $E(\boldsymbol{\theta})$ be the relative error measured by the Frobenius norm as follows.

$$E(\boldsymbol{\theta}) = \frac{\|\mathcal{X} - \hat{\mathcal{X}}\|_F}{\|\mathcal{X}\|_F}, \text{ with } \hat{\mathcal{X}} = \Phi^{-1}(\hat{\mathcal{Y}}_{\boldsymbol{\theta}}) \text{ and } \hat{\mathcal{Y}}_{\boldsymbol{\theta}} = T(\mathcal{G}_1^{(\boldsymbol{\theta})}, \cdots, \mathcal{G}_d^{(\boldsymbol{\theta})}) \tag{2.7}$$

where $\Phi(\cdot)^{-1}$ resizes the tensor to the original shape and removes dummy elements if there are any, $T(\cdot)$ generates the approximation tensor $\hat{\mathcal{Y}}_{\boldsymbol{\theta}}$ from its decomposed factors $\mathcal{G}_j^{(\boldsymbol{\theta})}$s.

Since the added dummy elements are zero, then $\|\mathcal{X}\|_F = \|\mathcal{Y}\|_F$ and $\|\mathcal{X} - \hat{\mathcal{X}}\|_F \leq \|\mathcal{Y} - \hat{\mathcal{Y}}\|_F$. Also, the TT-SVD guarantees that $\frac{\|\mathcal{Y}-\hat{\mathcal{Y}}\|_F}{\|\mathcal{Y}\|_F} \leq \epsilon$. Therefore, if the TT-SVD is applied for the decomposition of the reshaped tensor, $E(\boldsymbol{\theta}) \leq \epsilon$ and it is not required to consider the error bound as a constraint in the optimization model.

2.6 Genetic Algorithm for Tensor Geometry Search

The proposed optimization model (2.6) is a challenging combinatorial problem. When the data are reordered and reshaped the TT ranks of the rearranged data need to be determined for calculation of the space saving of tensor compression. Determining the ranks of a tensor is known to be NP-complete [59]. Therefore, a genetic algorithm (GA)

is applied to solve the defined optimization model and find the optimal tensor shape. The GA and evolutionary algorithms in general are usually used where the problem is combinatorial and non-convex, and the GA is an effective and common approach for solving this kind of problem. A pseudo-code of the GA for tensor shape search is presented in Algorithm 2, and its key steps are described below.

2.6.1 Initialization

The GA starts with generating a set of random shapes (solutions) $\mathcal{I} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, ..., \boldsymbol{\theta}_m\}$ as an initial population. The initial population is generated by applying a discrete uniform distribution [specifically, $\mathbf{unif}(l, u)$] on each variable $(n_i, i = 1, 2, ..., d)$ of $\boldsymbol{\theta}_j = (n_1, n_2, ..., n_d)$ for $j = 1, 2, ..., m$. Next for each shape $\boldsymbol{\theta}_j$, the TT-SVD is called, and the space saving $C(\boldsymbol{\theta}_j)$ is calculated by Eq.(2.6).

2.6.2 Selection

Proportional to the space saving of each solution, a selection probability is assigned to each shape as below.

$$\Pi(\boldsymbol{\theta}_j) = \frac{C(\boldsymbol{\theta}_j)}{\sum_{j=1}^{m} C(\boldsymbol{\theta}_j)},\ j = 1, 2, \ldots, m \tag{2.8}$$

where $\Pi(\boldsymbol{\theta}_j)$ is the selection probability of shape $\boldsymbol{\theta}_j$. In the selection process of the GA, p $(p < m)$ shapes are selected as parents. $(p - 1)$ solutions are selected based on the probability distribution Π (calculated above) with replacement such that the shapes with higher probability (Π) have more chance to be selected to enter to the parent set. If a solution is selected several times, then several copies of that exist in the parent set. An elitism operation is also applied so that the best shape of the current population (the shape with the maximum compression) is moved to the parent set with probability 1.

2.6.3 Reproduction

During the reproduction process the crossover operator is applied first. Based on the crossover operator two shapes like $\boldsymbol{\theta} = (n_1, ..., n_d)$ and $\boldsymbol{\theta}' = (n'_1, ..., n'_d)$ are randomly selected from the parent set, and a new trial shape is generated by exchanging the variables of the two selected solution as shown below.

$$\boldsymbol{\theta}^{\text{new}} = (n_1, ..., n_c, n'_{c+1}, ..., n'_d) \tag{2.9}$$

where c is the crossover point. Next, the mutation operator is applied to the newly generated solution. Based on the mutation operator, some of the dimensions (variables) of the newly generated shapes are randomly replaced by applying a discrete uniform distribution $\mathbf{unif}(l, u)$. If $\boldsymbol{\theta} = (n_1, ...n_i, ..., n_d)$ is a newly generated shape by the crossover, the muted shape is $\boldsymbol{\theta}^{\text{new}} = (n_1, ..., n''_i, ..., n_d)$ where dimension i is muted. The procedure of selecting parents and generating new solutions continues until $m - p$ new shapes are generated. The space saving of the newly generated shapes (new population) is calculated and the selection probabilities are updated.

2.6.4 Iteration and Convergence

The process of selection and reproduction repeats for T iterations. The best final shape is reported as the best (optimal) solution. There is no guarantee that the GA will find an optimal solution, but experimental results have shown the effectiveness of the GA in finding a near optimal solution [54]. [60] presented the stochastic convergence of the elitist GA.

Algorithm 2 The genetic algorithm for the tensor geometry search with the tensor train compression

Require: T, m, p
  Generate m tentative shapes
  for $j = 1$ to m **do**
    Run the TT-SVD algorithm and Calculate $C(\boldsymbol{\theta}_j)$
  end for
  $\boldsymbol{\theta}^* =$ the best shape in the current population
  for $t = 1$ to T **do**
    for $j = 1$ to m **do**
      Calculate $\Pi(\boldsymbol{\theta}_j)$
    end for
    for $j = 1$ to $p - 1$ **do**
      Select one shape using the distribution Π
      Copy the selected shape to the parent set
    end for
    Copy the best solution to the parent set
    for $j = 1$ to $m - p$ **do**
      Generate a new solution using the crossover operator
      Mute the newly generated solution using the mutation operator
      Run the TT-SVD algorithm for the new shape $\boldsymbol{\theta}_j$ and Calculate $C(\boldsymbol{\theta}_j)$
    end for
    New population = parent set + new solutions
    $\boldsymbol{b} =$ the best shape in the current population
    if $C(\boldsymbol{b}) > C(\boldsymbol{\theta}^*)$ **then**
      $\boldsymbol{\theta}^* = \boldsymbol{b}$
    end if
  end for

2.7 Random Search

In addition to the genetic algorithm (GA) that searches a near-optimal shape, a random search (RS) is also applied in this work. The best solution found by the RS, $\boldsymbol{\theta}_r$, is compared with the near optimal shape found by the GA. Hence the number of randomly generated shapes is the same as the total number of solutions examined by the GA. Algorithm 3 represents the applied random shape search.

Algorithm 3 Random search algorithm

Require: T, d, l, S
  $C(\boldsymbol{\theta}_r) = 0$
  for $t = 1$ to T **do**
    $u = \lceil \frac{S}{l^{d-1}} \rceil$
    for $j = 1$ to $d - 1$ **do**
      $n_j = RandInit(l, u)$
      $u = \lceil \frac{l \times u}{n_j} \rceil$
    end for
    $n_d = \lceil \frac{S}{\prod_{j=1}^{d-1} n_j} \rceil$
    $\theta = (n_1, n_2, \cdots, n_d)$
    Run the TT-SVD algorithm for the shape θ and Calculate $C(\theta)$
    if $C(\theta) > C(\boldsymbol{\theta}_r)$ **then**
      $\boldsymbol{\theta}_r = \theta$
    end if
  end for

In Algorithm 3, given the size of the data, S , where a possible shape is defined as $\theta = (n_1, n_2, \cdots, n_d)$, there are $d - 1$ degrees of freedom, and the last dimension is determined such that the size of the randomly generated shape is immediately greater than or equal to the size of the given data to be compressed. Meantime, to generate a random shape, the lower boundary of the size of all dimensions l is fixed, but the upper boundary, u, dynamically changes according to the previously determined dimensions. Note that the same approach described in Algorithm 3 is applied for the initialization of the GA, too.

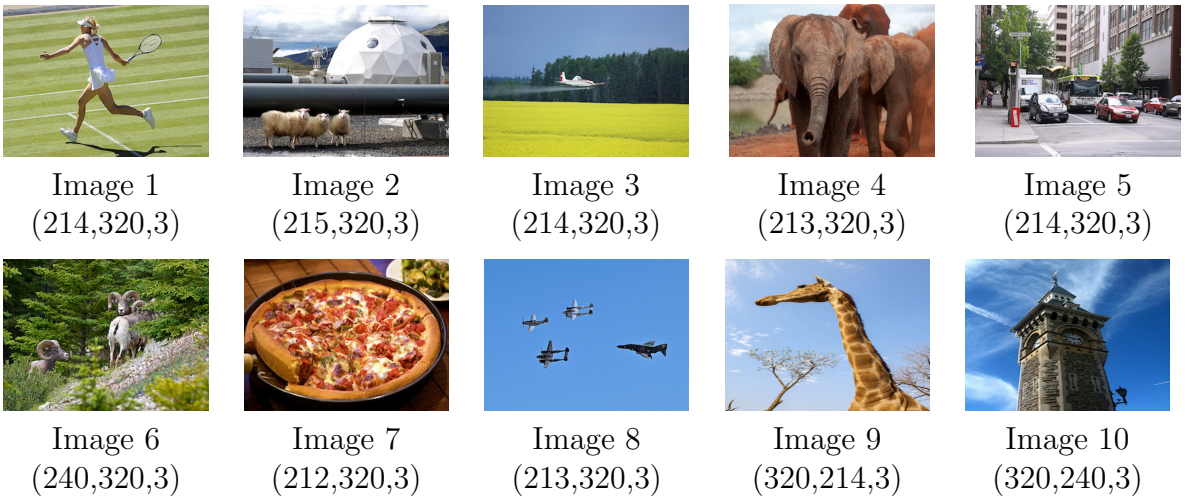

Image 1 (214,320,3) | Image 2 (215,320,3) | Image 3 (214,320,3) | Image 4 (213,320,3) | Image 5 (214,320,3)

Image 6 (240,320,3) | Image 7 (212,320,3) | Image 8 (213,320,3) | Image 9 (320,214,3) | Image 10 (320,240,3)

Figure 2.2: The arbitrary selected images form the COCO data set (the images are not depicted to their correct scale and the numbers written in parenthesis (height, width, depth) refer to the original shape, $\boldsymbol{\theta}_o$, of the image's data array).

2.8 Experimental Results

The proposed tensor geometry search using the TT-SVD algorithm is applied to decompose some arbitrary RGB images from the Microsoft common objects in context (COCO) data set [61] depicted in Fig. 2.2. Note that using the proposed method to compress the RGB images is only done for experimental purposes and for demonstrating the capability of the method for signal compression and dimensionality reduction while studying the method's performance but the application of the proposed method is beyond just compressing the RGB images. The images are resized in the experiments such that the longest dimension has 320 pixels with a fixed aspect ratio of the original image. Fig. 2.2 also shows the original shape (height, width, depth) of the data arrays of the images below them.

Table 2.1: The result of the compression of the studied images with their original shape ($\boldsymbol{\theta}_o$) and the optimal shape ($\boldsymbol{\theta}^*$) for $\epsilon = 0.05$.

Image	$C(\boldsymbol{\theta}_o)\%$	$E(\boldsymbol{\theta}_o)$	Optimal shape ($\boldsymbol{\theta}^*$)	$C(\boldsymbol{\theta}^*)\%$	$E(\boldsymbol{\theta}^*)$
1	48.45	0.0349	(1903,3,36)	67.50	0.0345
2	57.33	0.0264	(230,10,96)	72.13	0.0341
3	82.54	0.0313	(116,60,30)	88.05	0.0332
4	50.09	0.0249	(448,20,24)	75.02	0.0351
5	22.28	0.0247	(3493,2,30)	56.87	0.0345
6	-4.17	0.0248	(3200,4,18)	33.47	0.0346
7	12.46	0.0246	(3770,3,18)	40.00	0.0331
8	86.65	0.0253	(430,20,24)	92.67	0.0340
9	59.58	0.0348	(1975,5,21)	62.44	0.0340
10	65.62	0.0270	(320,10,72)	74.47	0.0343

2.8.1 Optimal Shape Versus Original Shape

The decomposition results of the reshaped data are compared with that of the original shapes. The largest number of dimensions and the lower boundary for the dimension size are set to have a fair comparison such that all the optimum shapes are of order three, similar to the original shapes (i.e., $d = 3$ and $l = 2$). For each image the GA runs for 50 iterations with a population size of 20. Note that reducing the error bound ϵ for TT decomposition reduces the space saving and compression efficiency because reducing the error increases the ranks and requires more factors to be stored. In this study the performance of the proposed method was studied using different error bounds varying from $\epsilon = 0.01$ to $\epsilon = 0.2$. Fig. 2.3 shows the convergence curve of the GA runs for the studied images with $\epsilon = 0.1$. Tables 2.1-2.3 lists the results of the compression of the studied images with their original shapes and the optimal shapes found by the GA for different error bounds including $\epsilon = 0.05$, $\epsilon = 0.1$, and $\epsilon = 0.2$, respectively. In Tables 2.1-2.3, $\boldsymbol{\theta}^*$ refers to the optimal shape found by the GA, and $\boldsymbol{\theta}_o$ refers to the original shape of the images.

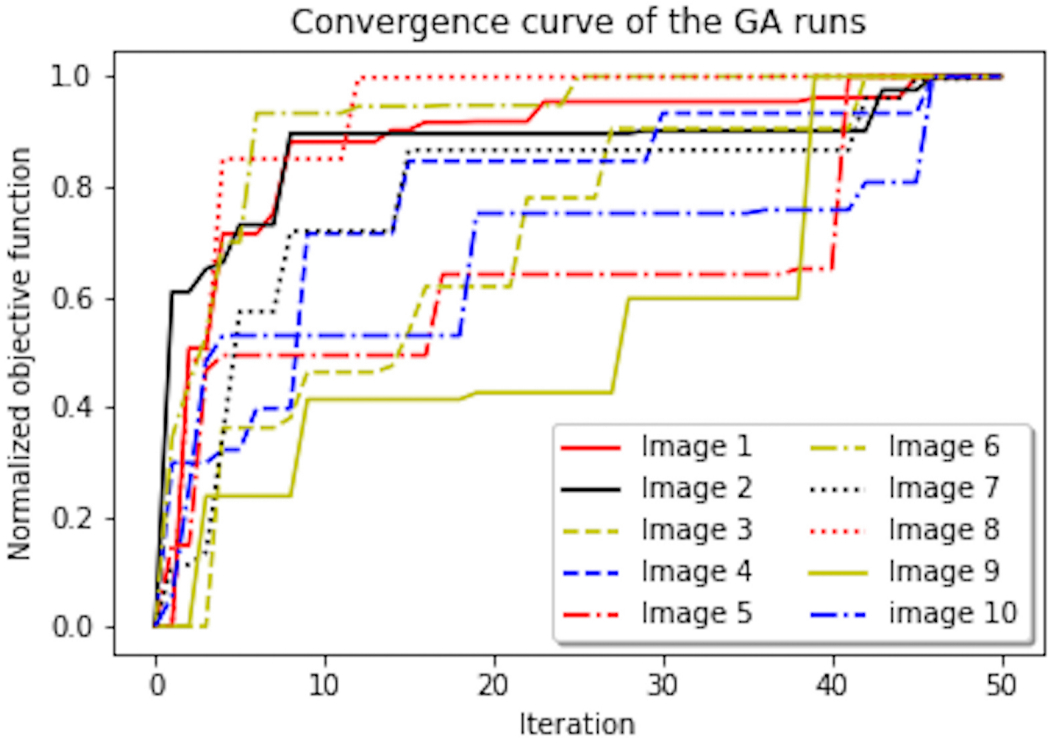

Figure 2.3: The convergence curve of the GA runs.

Table 2.2: The result of the compression of the studied images with their original shape ($\boldsymbol{\theta}_o$) and the optimal shape ($\boldsymbol{\theta}^*$) for $\epsilon = 0.1$.

Image	$C(\boldsymbol{\theta}_o)\%$	$E(\boldsymbol{\theta}_o)$	Optimal shape ($\boldsymbol{\theta}^*$)	$C(\boldsymbol{\theta}^*)\%$	$E(\boldsymbol{\theta}^*)$
1	72.98	0.0553	(222,16,60)	89.78	0.0694
2	75.14	0.0505	(437,8,60)	88.88	0.0701
3	94.18	0.0526	(428,10,48)	98.31	0.0680
4	82.89	0.0559	(107,16,120)	92.35	0.0696
5	62.58	0.0646	(471,8,60)	75.46	0.0702
6	17.70	0.0495	(1920,4,30)	58.46	0.0694
7	36.07	0.0499	(2270,3,30)	65.71	0.0697
8	97.13	0.0583	(71,320,9)	98.65	0.0695
9	79.61	0.0505	(193,51,21)	85.61	0.0694
10	80.21	0.0504	(349,12,60)	88.52	0.0685

Table 2.3: The result of the compression of the studied images with their original shape ($\boldsymbol{\theta}_o$) and the optimal shape ($\boldsymbol{\theta}^*$) for $\epsilon = 0.2$.

Image	$C(\boldsymbol{\theta}_o)\%$	$E(\boldsymbol{\theta}_o)$	Optimal shape ($\boldsymbol{\theta}^*$)	$C(\boldsymbol{\theta}^*)\%$	$E(\boldsymbol{\theta}^*)$
1	96.88	0.1370	(98,28,75)	98.32	0.1383
2	95.59	0.1149	(108,15,128)	98.15	0.1383
3	98.33	0.0999	(70,28,108)	99.65	0.1374
4	94.99	0.1032	(92,44,51)	97.15	0.1399
5	90.12	0.1202	(437,16,30)	92.91	0.1365
6	63.33	0.1145	(2575,3,30)	78.39	0.1378
7	76.14	0.1218	(433,5,96)	87.91	0.1377
8	99.48	0.1012	(161,17,75)	99.88	0.1279
9	94.54	0.0990	(81,45,57)	98.52	0.1366
10	92.71	0.1005	(214,30,36)	96.76	0.1387

It is seen in Tables 2.1-2.3 that for all images the space saving of the optimal shape ($\boldsymbol{\theta}^*$) found by the GA is superior to that of the original shape ($\boldsymbol{\theta}_o$). Also, all the errors are smaller than the error bound ϵ. The change in the error is negligible and is bounded although the error slightly increases by improving the space saving, whereas the improvement in the space saving is significant. It is also seen that the space saving of the studied images varies, and it is because the images have different ranks. Regardless of the ranks of the images the proposed method improved the space saving of all the studied images. For instance, for image 7, the space saving has increased from 12.46%, 36.07%, and 76.14% to 40.00%, 65.71%, and 87.91% for error bounds 0.05, 0.1, and 0.2, respectively. In Table 2.1 image 6 has a negative space saving when its original shape is used. That means there was no compression and the factors require more space than the original data. However, the space saving achieved by using the shape search algorithm improved from -4.17% to 33.47%. Considering all of the studied images, on average, the space saving improved by about 18.5%, 14.3%, and 4.6% for error bounds 0.05, 0.1, and 0.2, respectively (referring to the difference between columns 2 and 5 of tables 2.1-2.3).

We can conclude that the compression results of the optimal shapes were significantly improved in comparison with that of the original shapes.

Table 2.4 lists the ratio between the compression ratio of the optimal shape, $R(\boldsymbol{\theta}^*)$, and the compression ratio of the original shape, $R(\boldsymbol{\theta}_o)$ for different error bounds from $\epsilon = 0.01$ up to $\epsilon = 0.2$. In other words, Table 2.4 refers to the ratio of the size of the compressed data using the original shape to the size of the compressed data using the optimal shape. Therefore, the larger the ratio, the higher the efficiency of the optimal shape. Remember that the compression ratio, $R(\boldsymbol{\theta})$, is defined in Eq.(2.5). In Table 2.4, it is seen that by increasing the error bound, on average, $R(\boldsymbol{\theta}^*)/R(\boldsymbol{\theta}_o)$ increases while the variance also increases. This is visualized in Fig 2.4, which depicts the minimum, mean, maximum, and variance of $R(\boldsymbol{\theta}^*)/R(\boldsymbol{\theta}_o)$ for all images versus the error bound. According to Table 2.4 and Fig. 2.4, for the small error bounds variance is close to zero and the compression ratio for the optimal shape is about a factor of 1.5 greater than that of the original shape. By relaxing the error bound on average the compression ratio of the optimal shape is about 2.6 times that of the original shape. Compression is usually more challenging when the error bound is very tight because the accuracy of the data is well preserved. According to Table 2.1 the space saving for the original shape is about 47% on average for an error bound of 5% while the space saving for the optimal shape increases to about 66% on average over all studied images. It is seen in Table 2.3 that for $\epsilon = 0.2$ the TT compression using the original shape achieved a space saving of 90% on average that represents an increment of up to 94% on average using the tensor shape search (i.e., the optimal shape). The improvement in the space saving from 90% to 94% may not seem as significant as the raise in the space saving from 47 to 66 for the smaller error bound. However, studying the compression ratios as shown in Fig. 2.4 along side the space saving, it becomes more clear that the proposed tensor shape search improved the compression efficiency for both tight and loose error bounds significantly.

Table 2.4: $R(\boldsymbol{\theta}^*)/R(\boldsymbol{\theta}_o)$ for different error bounds.

.

Image	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.15$	$\epsilon = 0.2$
1	1.63	1.51	2.64	3.21	1.86
2	1.51	1.53	2.24	1.49	2.38
3	1.27	1.46	3.44	5.29	4.77
4	1.65	1.65	2.24	2.43	1.76
5	1.17	1.80	1.52	1.07	1.39
6	1.24	1.57	1.98	1.60	1.70
7	1.17	1.46	1.86	1.53	1.97
8	1.36	1.82	2.13	2.97	4.33
9	1.32	1.08	1.42	1.82	3.69
10	1.41	1.70	1.72	1.53	2.25

The original data are restored using the TT factors. Normally the retrieval of the original data only includes the multiplication of the TT core factors as specified in Eq.(3.3). Concerning the GA solution the reshaping may add dummy variables during the reshaping process. However, the reshaping is a bijection mapping that simply allows the original data to be restored. Figure 2.5 visualizes an estimation of image 4 for different error bounds for both GA's optimal shape and the original shape. It is seen in Figure 2.5 for $\epsilon = 0.05$ the restored image is almost identical to the original image. However, by relaxing the error bound, the accuracy reduces and the restored image is blurry for $\epsilon = 0.2$ for both original shape and the optimal shape. The error bound, ϵ was set to be the same for both the original shape and the optimal shape as it is reported in Tables 2.1-2.3, yet, the actual error of the optimal shape was mostly higher than that of the original shape. This difference is visible in Figure 2.5 comparing the restored images for $\epsilon = 0.2$. Therefore, improving the compression efficiency of the TT decomposition using the GA may slightly result in a lower accuracy, although the error is bounded.

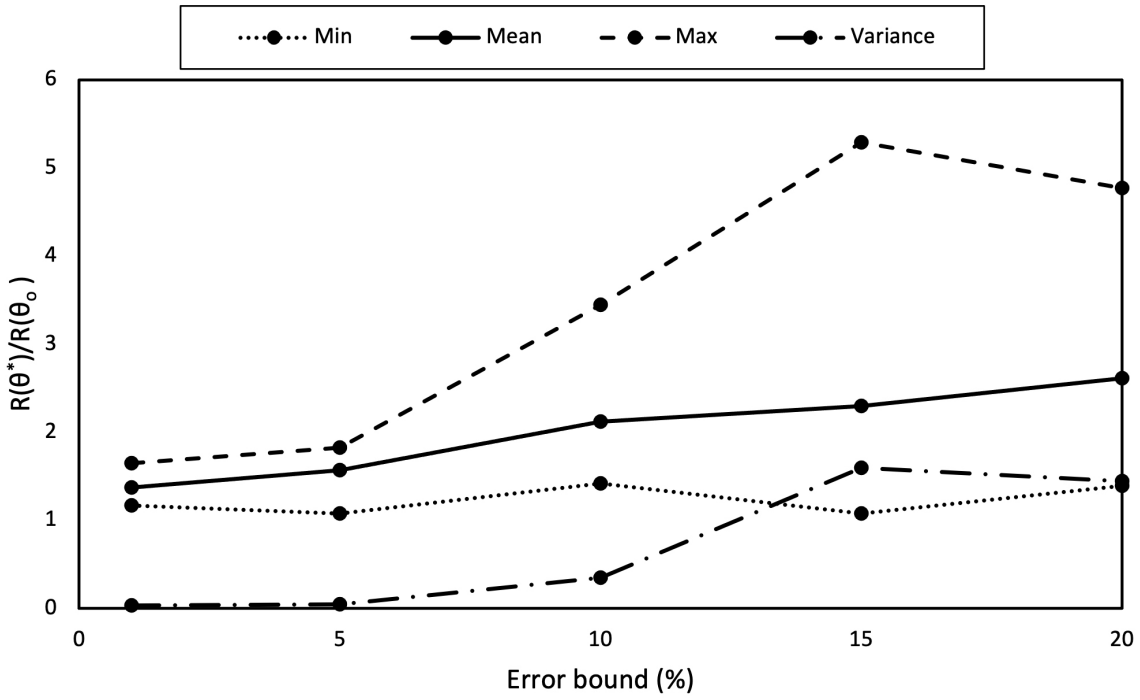

Figure 2.4: Comparison of $R(\boldsymbol{\theta}^*)/R(\boldsymbol{\theta}_o)$ for different error bounds including $\epsilon = 0.01$, $\epsilon = 0.05$, $\epsilon = 0.10$, $\epsilon = 0.15$, and $\epsilon = 0.20$.

2.8.2 Random Search Versus GA

A random search (RS) is applied in addition to the GA. The total number of randomly generated shapes was 1,000, which is equal to the total number of solutions examined by the GA. This keeps the computational cost almost the same between the two methods since the major computational burden belongs to simulating the TT decomposition and the related SVDs for each possible shape. The best solution among all the randomly generated solutions is selected and the space savings are reported in Table 2.5. Table 2.6 compares the space saving and compression ratios between the optimal shape found by the GA and the best shape found by the random search. The results demonstrate that the optimal shapes found by the GA are all superior to those found by the RS. In Table 2.6 it is seen that on average the GA improved the space saving by 2.66%,

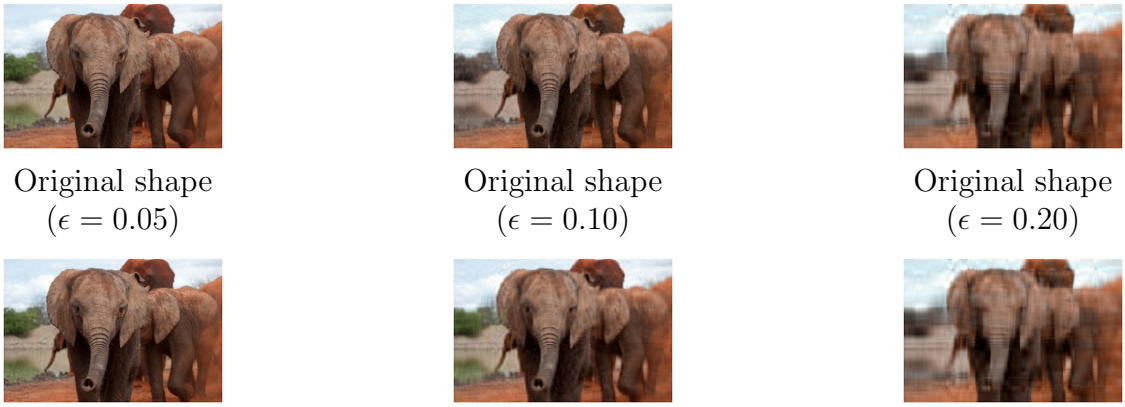

Figure 2.5: Restoration of image 4 from the compressed data using the optimal shape found by the GA and the original shape.

2.97%, and 1.19% for error bounds 0.05, 0.10, and 0.2, respectively. There are cases like image 10, where the GA's solution is 11.30 % better than that of the RS. In Table 2.6 it is also seen that the ratio between compression ratios is also always larger than 1 meaning that the shape found by the RS results in a greater cardinality of factors in comparison to that of the GA's solution. Note that the ratio between compression ratios compares the cardinality of the factors head to head regardless of the initial size of the data. Therefore, the GA performed better. However, the random search also found solutions better than the initial shape. In fact, the random search is also successful in improving the compression efficiency although the GA may provide a better solution.

The average wall time of the GA spent using a 2.3 GHz Quad-Core Intel Core i5 processor for error bounds 0.05, 0.1, and 0.2 was about 49, 35, and 25 seconds, respectively. The average wall time of the RS for error bounds 0.05, 0.1, and 0.2 were about 30, 23, and 17 seconds, respectively. The compression efficiency of the GA was higher even though the RS was slightly faster.

Table 2.5: The space saving of the best shapes found by the random search, $C(\boldsymbol{\theta}_r)\%$, for different error bounds.

Image	$\epsilon = 0.05$	$\epsilon = 0.10$	$\epsilon = 0.20$
1	66.41	87.50	97.62
2	68.24	85.82	97.41
3	86.98	97.37	98.52
4	72.77	87.66	96.10
5	55.57	72.18	91.13
6	30.51	57.76	77.97
7	39.04	64.02	85.84
8	91.19	98.07	99.65
9	62.16	82.88	96.51
10	63.17	78.80	95.00

2.8.3 Neural Network Compression

One of the common applications of tensor compression is to tensorize neural networks. Tensorizing neural networks refers to compressing parameters of a neural network using tensor decomposition that allows efficient use of the memory and computation, specially when the hardware resources are limited. In the tensorized network the tensorized factors are stored in the memory instead of storing the raw parameters. To apply tensor decomposition the parameters must be a high dimensional array (tensor), but not all neural network parameters are initially high dimensional. For example, the parameters of a fully connected layer are initially represented as a matrix or 2D array. The parameters of fully connected layers must be represented as a higher dimensional data (at least 3D) for tensor decomposition to be applicable. Here we study the performance of the tensor shape search for tensorized neural networks.

In this experiment a network is implemented to solve the MNIST data set [62]. The network consists of two dense layers. The MNIST images are 28 by 28 pixels posing 784 inputs to the network. The first dense layer has 512 neurons with rectifier linear unit (relu) activation functions and consequently has a weight matrix of size 784 by 512. The

Table 2.6: Comparing the results of the GA and the RS including the differences in space savings (%) and the ratio between compression ratios.

2*Image	$C(\boldsymbol{\theta}^*) - C(\boldsymbol{\theta}_r)\%$			$R(\boldsymbol{\theta}^*)/R(\boldsymbol{\theta}_r)$		
2-7	$\epsilon = 0.05$	$\epsilon = 0.10$	$\epsilon = 0.20$	$\epsilon = 0.05$	$\epsilon = 0.10$	$\epsilon = 0.20$
1	1.09	2.28	0.70	1.03	1.22	1.42
2	3.89	3.06	0.74	1.14	1.28	1.4
3	1.07	0.94	1.13	1.09	1.56	4.23
4	2.25	4.69	1.05	1.09	1.61	1.37
5	1.3	3.28	1.78	1.03	1.13	1.25
6	2.96	0.70	0.42	1.04	1.02	1.02
7	0.96	1.69	2.07	1.02	1.05	1.17
8	1.48	0.58	0.23	1.20	1.43	2.92
9	0.28	2.73	2.01	1.01	1.19	2.36
10	11.3	9.72	1.76	1.44	1.85	1.54
Min	0.28	0.58	0.23	1.01	1.02	1.02
Mean	2.66	2.97	1.19	1.11	1.33	1.87
Max	11.30	9.72	2.07	1.44	1.85	4.23

last layer is also a dense layer with 10 softmax units. The total number of parameters of the network is 407,050 out of which almost 99% are the weights of the first relu layer. Therefore, we only compressed the weight matrix of the first fully connected layer for compression of the network. This work applied a post-training compression technique in which the parameters of the network were initially optimized in their raw format. After initialization the parameters of the first layer were reshaped and compressed using the proposed method. The accuracy of the network might be reduced due to the error of the compression. Therefore, a retraining was applied while the parameters of the first layer were set non-trainable and only the parameters of the last dense layer (less than 1% of the total parameters) were retrained. d was set to be 4 and l was set to be 1. Therefore, the GA and random search (RS) explored shapes with dimension 3 and 4. Like the previous experiments the population size and the number of iterations of the GA were 20 and 50, respectively. For the RS, 1,000 randomly trial solutions were examined.

Table 2.7 lists the results for the network accuracy before and after compression.

Table 2.8 lists the optimum shape found by the GA and RS for compressing the first dense layer. It is seen in Table 2.7 that using the proposed shape search by the GA the network can be compressed up to 300 times while the accuracy of the network is slightly affected for $\epsilon = 1$. For a more conservative error bound, $\epsilon = 0.8$, the accuracy of the network compressed by the GA is closer to the uncompressed network and the memory requirement of the network reduces to 11 times. Comparing the RS with the GA, the GA provides a more efficient compression for both error bounds. In Table 2.8, it is seen that the RS preferred a 3D array while the GA preferred a 4D array for the reshaping. The compression efficiency depends on both shape and the resulting TT-ranks, therefore the TT ranks for each shape is also reported in Table 2.8.

The weights of the first layer are compressed for which the maximum, average, and minimum space saving of 1,000 random trial shapes examined by the RS are 99.03%, 51.96%, and -48.26%, respectively, for $\epsilon = 1.0$. Also, for $\epsilon = 0.8$ the maximum, average, and minimum space saving of 1,000 random trial shapes for the weights of the first layer are 75.73%, 4.73%, and -97.36%, respectively. For the GA, the space saving of the best found shape is 99.79% and 91.12% for $\epsilon = 1$ and $\epsilon = 0.8$, respectively. The wide range of space savings across 1,000 randomly generated shapes demonstrates the effect of the shape of the tensor on the compression efficiency and justifies the need for a shape search before transforming a 2D parameter array to a higher dimension for tensor compression. Note that the results listed in Tables 2.7 and 2.8 correspond to the largest space savings.

On a 2.6 GHz Intel Core i7 processor, the wall time of the GA for error bounds 1 and 0.8 were about 116 and 247 seconds, respectively. On the same processor, the wall time of the RS for error bounds 1 and 0.8 were about 31 and 43 seconds, respectively. Although the RS is faster, the compression efficiency of the shape found by the GA is better.

The results of the compressing the MNIST network using the proposed tensor shape

Table 2.7: The accuracy and compression of the GA and the random search (RS) for MNIST in comparison to the base uncompressed model.

Network	Train(%)	Validation(%)	#Parameters
Base (Uncompressed)	98.65	90.08	407,050
GA ($\epsilon = 1.0$)	95.92	88.62	1,353 (300×)
RS ($\epsilon = 1.0$)	95.41	88.10	4,425 (92×)
GA ($\epsilon = 0.8$)	98.13	89.30	36,170 (11.3×)
RS ($\epsilon = 0.8$)	97.65	88.60	97,916 (4.2×)

Table 2.8: The optimum shapes and the TT ranks for the compressed layer of the MNIST network.

Network	Shape	TT Ranks
GA ($\epsilon = 1.0$)	(221,303,2,3)	(1,1,2,2,1)
RS ($\epsilon = 1.0$)	(3858,53,2)	(1,1,1,1)
GA ($\epsilon = 0.8$)	(522,4,16,16)	(1,29,66,12,1)
RS ($\epsilon = 0.8$)	(506,3,300)	(1,63,134,1)

search demonstrate that the shape of the tensor significantly affects the compression efficiency. Therefore, it is necessary to explore the tensor shapes before applying tensor compression on neural networks. Also, the proposed tensor shape search using the GA successfully improved the space saving in comparison to the random search.

2.9 Discussion

Despite the success of the tensor decomposition methods such as tensor train (TT) decomposition in data compression and dimensionality reduction not all of the real-world data primarily are high-dimensional, and sometimes a reshaping is necessary prior to tensor compression. For instance, a low-dimensional (i.e., 1D or 2D) data array is required to be transformed to a higher dimension for tensor compression to be applicable. Meantime, reordering and reshaping data may affect the efficiency of the compression. This work proposed a tensor geometry optimization paradigm for data compression using

TT decomposition, and a GA was applied to solve the proposed optimization model.

The results demonstrated that the compression efficiency can be practically improved using the proposed method. The proposed tensor geometry search method significantly improved the space saving and compression ratio in comparison to the original shape of the data. Furthermore, a comparison of the GA with the pure random search revealed that the shapes found by the GA were superior to those found by the random search but the random search also may improve the compression efficiency in comparison to the original shape of the data. The proposed tensor geometry search method bounds the error, but in a head-to-head comparison between the optimal shape and the original shape, It was observed that improving the space saving of the TT decomposition using the proposed tensor geometry search may slightly increase the error. However, the gained space savings were significant while the error differences were mostly negligible.

The effect of tensor reshaping on tensor decomposition has been rarely studied in the literature. This study demonstrates the importance of the topic and justifies that further research and more attention to this topic are required. Obviously, any reformatting of a data array may affect its decomposition, and it was not the purpose of this work to show that reshaping affects the decomposition but the main objective of this work was to formulate reshaping as a practical method to improve the efficiency of tensor compression methods where such reshaping is necessary or where it is viable. Reshaping may not be feasible for some of tensor decomposition applications if the original structure of the data must be preserved. In such cases the application of the proposed method may be limited. Another limitation of the current study is solving the posed optimization model using the GA requires hierarchical SVDs for every potential shape to be conducted that is time consuming and may limit the application of tensor geometry search. The proposed methodology was only applied for the TT format. The study of the effectiveness of the proposed tensor shape search for other decomposition methods such as the Tucker de-

composition, and improving the efficiency of the optimization algorithm are the subjects of future studies.

2.10 Conclusion

This work empirically studied the possible effect of the shape of a tensor in the compression of tensorized signals and neural networks. The study was narrowed down to the TT decomposition. The task of finding the optimum shape for the tensor train (TT) decomposition was formulated as an optimization model which maximizes the space saving with respect to the geometry of a given tensor subject to an error bound. A genetic algorithm (GA) linked with the TT-SVD algorithm was presented to solve the proposed optimization model. The performance of the GA was also compared with the random search. The capability of the proposed method was exemplified by compressing RGB images and a neural network for the MNIST data set. The results demonstrated that the efficiency of tensor compression was improved using the proposed tensor geometry search method. The study demonstrated that the tensor shape had a significant effect in the compression efficiency of the tensorized data and neural networks. Therefore, the proposed optimization paradigm can be applied to utilize the shape effect for enhancing the efficiency of both data and model compression using the TT decomposition.

Chapter 3

Tensor Geometry Optimization for Low-Rank Neural Networks

3.1 Introduction

Deep neural networks have achieved success in various applications [63, 64, 65]. Meanwhile, implementing neural networks on resource-constrained edge devices (also called edge AI accelerators) has demonstrated various benefits such as improving security and data privacy, reducing latency and response time, saving bandwidth and enhancing efficiency, reliability, and scalability [66, 67, 68, 69]. The main challenge of edge AI is the substantial memory and bandwidth requirements for storing and loading network parameters, along with the time complexity associated with multiplication and accumulation (MAC) operations [70].

Various software methods and algorithms have been studied that reduce hardware cost and build compressed deep neural networks such as quantization [71, 72, 73], pruning [74, 75], knowledge distillation [76], and low-rank factorization [2, 3, 77]. Low-rank tensor factorization has achieved state-of-the-art compression performance in various neural

network architectures [43] including fully connected networks [1], convolutional neural networks (CNNs) [2, 3, 4], recurrent neural networks [5, 6], recommendation systems [7, 8] and transformers [9].

The performance of a low-rank tensorized network depends on the choices of various hyper-parameters such as the tensor rank and geometry. Although exactly determining the rank of a tensor is known to be NP-complete [59], recently Bayesian methods were used to determine proper (not necessarily exact) tensor ranks automatically in tensorized neural network training [7, 10], enabling on-device training of neural networks with limited computing resources [11]. In other fields, some penalty or cost functions were chosen carefully to determine a proper tensor rank [12, 13].

Besides tensor ranks, tensor geometry (i.e., tensor order and dimensions) can also affect significantly the performance of a low-rank tensor model [14, 15, 47, 16]. The study [14] applied tensor reshaping to achieve a square unfolded matrix in Tucker decomposition for tensor completion and data retrieval. Another study presented an adaptive dimension adjustment Tucker decomposition (ADA-Tucker) for neural network compression, which can achieve higher compression ratios than using an arbitrary tensor shape [15]. Our previous studies [47, 16] have shown that the compression performance of tensor decomposition on both image data and neural network models can be improved by a tensor shape search. However, determining the proper tensor geometry in compressed neural networks remains an open question.

Summary of Contributions. In this chapter, we present a computational framework to optimize the tensor geometry of low-rank tensorized neural networks. Our specific contributions include:

- We present a mathematical formulation to optimize the tensor geometry in low-rank tensorized neural networks. The resulting optimization problem is combinatorial

and thus challenging to solve.

- We investigate different methods to solve the formulated problem, including integer linear programming, graph traversal, and random search algorithms. We demonstrate that the most feasible balanced geometry is an upper bound solution to the defined geometry optimization model and might be sub-optimal. Consequently, we propose the geometry search-based initialization (GSI) to improve the accuracy and compression efficiency of tensorized neural networks.

- We conduct numerical experiments on fully connected and convolutional networks. We study the effect of tensor geometry on tensorized networks and apply the proposed methods to reduce the memory and computational cost of tensorized networks with the tensor train (TT) and Tucker formats.

3.2 Notations and Background

3.2.1 Notations

Throughout this manuscript, lower-case Roman or Greek letters denote scalars (e.g., a or θ). Boldface lower-case letters (e.g., $\boldsymbol{a}$ or $\boldsymbol{\theta}$) denote vectors, and boldface upper-case letters (e.g., $\mathbf{A}$) denote matrices. Upper-case Greek letters (e.g.,Θ) denotes sets. Capital calligraphic letters (e.g.,$\mathcal{A}$) denotes tensors. $[\mathcal{A}]_{i_1 i_2 \cdots i_d}$ refers to the $(i_1, i_2, \cdots, i_d)$-th element of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_d}$. The number of dimensions (i.e., order) of a tensor like $\mathcal{X}$ is presented by $dim(\mathcal{X})$. The Frobenius norm is denoted by $||.||_F$, and the 2-norm is represented by $||.||$. For a generic tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_d}$, its Frobenius norm is computed as follows:

$$\|\mathcal{A}\|_f = \sqrt{\sum_{i_1, i_2, \cdots, i_d} ([\mathcal{A}]_{i_1 i_2 \cdots i_d})^2}. \tag{3.1}$$

A frontal tensor contraction denoted by $\mathcal{C} = \mathcal{A} \times_{1,\cdots,k} \mathcal{B}$ is a speical tensor contraction that involves contracting the first k dimensions of $\mathcal{A}$ and $\mathcal{B}$ as shown below:

$$[\mathcal{C}]_{i_{k+1}\cdots i_d j_{k+1}\cdots j_h} = \sum_{t_1\cdots t_k} [\mathcal{A}]_{t_1\cdots t_k i_{k+1}\cdots i_d} [\mathcal{B}]_{t_1\cdots t_k j_{k+1}\cdots j_h},$$
$$\forall\ i_{k+1}, \cdots, i_d, j_{k+1}, \cdots, j_h, \tag{3.2}$$

where $\mathcal{C} \in \mathbb{R}^{I_{k+1}\times\cdots\times I_d \times J_{k+1}\times\cdots\times J_h}$, $\mathcal{A} \in \mathbb{R}^{I_1\times\cdots\times I_d}$, $\mathcal{B} \in \mathbb{R}^{J_1\times\cdots\times J_h}$ and $I_t = J_t$, $\forall t = 1, \cdots, k$. Meanwhile, the operator $\times_l^t$ refers to a tensor contraction operation where the l-th dimension from the left-side tensor is contracted with the t-th dimension from the right-side tensor. We use the notation $\mathcal{F}_\Psi(\mathcal{A})$ to refer to a tensor network where a tensor $\mathcal{A}$ is contracted with a set of tensors Ψ along one or more of their dimensions.

3.2.2 Tensor Decomposition

Tensor decomposition has been utilized in many science and engineering fields [30, 32, 33, 12, 20, 21, 22, 34, 29, 35, 36, 37, 38, 39, 40, 41]. A detailed review of tensor decompositions is provided in [19]. In this chapter, we focus on the tensor-train (TT) [27] and Tucker [26] decomposition methods which are widely used for neural network compression.

Definition 1 (Tensor-train (TT) format) *The TT format decomposes a d-way tensor $\mathcal{W} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ to d factors $\{\mathcal{G}_j \in \mathbb{R}^{r_{j-1}\times n_j\times r_j}\}_{j=1}^d$, where $(r_0, r_1, \cdots, r_d)$ are TT ranks and $r_0 = r_d = 1$. An approximation of $\mathcal{W}$ is given by a sequence of tensor contractions as follows [27]:*

$$\hat{\mathcal{W}} = \mathcal{G}_1 \times_3^1 \mathcal{G}_2 \times_3^1 \cdots \times_3^1 \mathcal{G}_d. \tag{3.3}$$

For a given tensor $\mathcal{W}$ and considering an error bound like $\epsilon = \frac{\|\mathcal{W}-\hat{\mathcal{W}}\|_F}{\|\mathcal{W}\|_F}$, the TT factors, $\mathcal{G}_j$'s, are derived by unfolding tensor $\mathcal{W}$ along its different dimensions, and conducting a σ-truncated singular value decomposition (SVD) for the associated unfolded matrices, where $\sigma = \frac{\epsilon}{\sqrt{d-1}}\|\mathcal{W}\|_F$. The aforementioned procedure to derive the TT factors based on an error bound is called the TT-SVD algorithm [27].

Definition 2 (Tucker format) *In the Tucker format, a d-way tensor $\mathcal{W} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is decomposed into a d-way smaller-size core factor $\mathcal{G} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_d}$ and d two-way factors $\{\mathbf{U}_i \in \mathbb{R}^{n_i \times r_i}\}_{i=1}^{d}$. An approximation of $\mathcal{W}$ is computed by a series of tensor contractions as follows [78]:*

$$\hat{\mathcal{W}} = (((\mathcal{G} \times_1^2 \mathbf{U}_1) \times_2^2 \mathbf{U}_2) \times_3^2 \cdots) \times_d^2 \mathbf{U}_d. \tag{3.4}$$

Given a tensor $\mathcal{W}$ and an error bound ϵ, the Tucker factors are computed by performing a sequence of σ-truncated SVDs on the unfolded $\mathcal{W}$ along its different dimensions where $\sigma = \frac{\epsilon}{\sqrt{d}}\|\mathcal{W}\|_F$. This is achieved using the Tucker higher-order SVD (Tucker-HOSVD) algorithm [79]. The Tucker ranks $(r_1, r_2, \cdots, r_d)$ are computed using Tucker-HOSVD and are referred to as hierarchical ranks.

3.2.3 Low-rank tensorized neural networks

A standard neural network can be written as follows:

$$Y = f(X|\{\Phi_l\}_{l=1}^{L}), \tag{3.5}$$

where X and Y are the input and output of the network, and Φ_l refers to the parameters of layer l. In this study we concentrate on fully connected and convolutional layers. Both

of these layers can be reduced to the core operation below:

$$\boldsymbol{y} = \sigma(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}), \tag{3.6}$$

where $\boldsymbol{y} \in \mathbb{R}^N$ is the layer output, σ is the activation function, $\boldsymbol{x} \in \mathbb{R}^M$ is the input, $\boldsymbol{W} \in \mathbb{R}^{M\times N}$ is the weight matrix, $\boldsymbol{b} \in \mathbb{R}^N$ is the bias. The matrix multiplication of Eq.(3.6) is usually the most expensive operation in a neural network, having a memory and time complexity of $O(MN)$.

Let $\boldsymbol{z} = \boldsymbol{W}^T\boldsymbol{x}$, where $\boldsymbol{W} \in \mathbb{R}^{M\times N}$ is the parameters to be learned and $\boldsymbol{x} \in \mathbb{R}^M$ and $\boldsymbol{z} \in \mathbb{R}^N$ are known. By folding $\boldsymbol{W}$ and $\boldsymbol{x}$ into tensors, the corresponding tensorized product results in a tensor contraction defined as follows:

$$\mathcal{Z} = \mathcal{W} \times_{1,\cdots,k} \mathcal{X} = \mathcal{F}_{\Psi(\mathcal{W})}(\mathcal{X}), \tag{3.7}$$

where $\mathcal{W} \in \mathbb{R}^{n_1\times\cdots\times n_k\times\cdots\times n_d}$, $\mathcal{X} \in \mathbb{R}^{n_1\times\cdots\times n_k}$, and $\mathcal{Z} \in \mathbb{R}^{n_{k+1}\times\cdots\times n_d}$ are the tensorized representation of $\boldsymbol{W}$, $\boldsymbol{x}$, and $\boldsymbol{z}$, respectively, and $\prod_{i=1}^{k} n_i = M$ and $\prod_{i=k+1}^{d} n_i = N$ where $1 \leq k < d$ and $d \geq 3$, and the tensor network $\mathcal{F}_{\Psi(\mathcal{W})}(\mathcal{X})$ refers to the low-rank tensorized product where by replacing $\mathcal{W}$ with its low-rank factors, $\Psi(\mathcal{W})$, $\mathcal{Z}$ is computed by a sequence of tensor contractions. First, $\mathcal{X}$ is projected into the low-rank space by contracting its dimensions with the corresponding factors. Next the output is computed from partially contracted factors. Substituting low-rank tensorized product into Eq.(3.6) we have:

$$\mathcal{Y} = \sigma(\mathcal{F}_{\Psi(\mathcal{W})}(\mathcal{X}) + \mathcal{B}), \tag{3.8}$$

where $\mathcal{B} \in \mathbb{R}^{n_{k+1}\times\cdots\times n_d}$ is the tensorized representation of $\boldsymbol{b}$.

3.3 Problem Statement

In a tensorized neural network, we need to first fold a weight $\mathbf{W}$ to a d dimensional tensor $\mathcal{W} \in \mathbb{R}^{n_1 \times n_2 \cdots \times n_d}$. The quality of the afterwards tensor decomposition and complexity of tensorized forward inference depend on the hyper-parameters $(n_1, n_2, \cdots, n_d)$ in the folding process. As a result, it is crucial to find the best tensor geometry that can optimize the memory and time complexities of a tensorized neural network.

We provide the mathematical formulation for the tensor geometry optimization in both the TT and Tucker formats.

Formulation 1 (Optimal tensor geometry for TT layers) *Let $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ be the weights of a layer and $\mathcal{W}_{\boldsymbol{\theta}} \in \mathbb{R}^{\theta_1 \times \cdots \times \theta_d}$ be the tensorized weights according to the geometry $\boldsymbol{\theta}$. Given a $d \geq 3$, a tensor shape $\boldsymbol{\theta} = (\theta_1, \cdots, \theta_d)$ is optimum when it minimizes the total number of variables in the TT factorization of $\mathcal{W}_{\boldsymbol{\theta}}$:*

$$\begin{aligned} \min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = & \sum_{j=1}^{d} r_{j-1}^{(\boldsymbol{\theta})} \times \theta_j \times r_j^{(\boldsymbol{\theta})}, \\ \text{subject to } & \prod_{j=1}^{k} \theta_j = M, \prod_{j=k+1}^{d} \theta_j = N, \\ & \theta_j \geq 2, \theta_j \in \mathbb{Z}, j = 1, \cdots, d. \end{aligned} \tag{3.9}$$

Here $r_j^{(\boldsymbol{\theta})}$ represents the TT ranks derived from decomposing $\mathcal{W}_{\boldsymbol{\theta}}$ using the TT-SVD algorithm with an error bound ϵ. Note that the TT ranks may change when we change the tensor geometry $\boldsymbol{\theta}$.

Formulation 2 (Optimal tensor geometry for Tucker layers) *Let $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ be the weights of a layer and $\mathcal{W}_{\boldsymbol{\theta}} \in \mathbb{R}^{\theta_1 \times \cdots \times \theta_d}$ be the tensorized weights according to the geometry $\boldsymbol{\theta}$. Given $d \geq 3$, a tensor shape $\boldsymbol{\theta} = (\theta_1, \cdots, \theta_d)$ is optimum when it minimizes*

the number of variables in the Tucker factorization of $\mathcal{W}_{\boldsymbol{\theta}}$:

$$\begin{aligned} \min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) &= \sum_{j=1}^{d} \theta_j \times r_j^{(\boldsymbol{\theta})} + \prod_{j=1}^{d} r_j^{(\boldsymbol{\theta})}, \\ \text{subject to } \prod_{j=1}^{k} \theta_j = M,\ &\prod_{j=k+1}^{d} \theta_j = N, \\ \theta_j \geq 2, \theta_j \in \mathbb{Z}, j = 1, \cdots, d. & \end{aligned} \tag{3.10}$$

Here $r_j^{(\boldsymbol{\theta})}$ represents the hierarchical Tucker ranks derived from decomposing $\mathcal{W}_{\boldsymbol{\theta}}$ using the Tucker-HOSVD algorithm with an error bound ϵ. The value of $r_j^{(\boldsymbol{\theta})}$ also depends on the tensor shape $\boldsymbol{\theta}$.

Previous studies on tensor geometry for neural networks [15, 16] did not address computational efficiency in their optimization model. Furthermore, inspired by matrix analysis [14], it was speculated that a more balanced shape would provide a better compression [15]. Therefore, we are interested to answer the following question:

What is the relationship between a balanced shape and the general tensor geometry optimization model?

In this work, our primary goal is to create a compressed network derived from a pre-trained teacher model. Therefore, we assume that an uncompressed pre-trained teacher model is available. This assumption is necessary to derive the hierarchical ranks in the TT and Tucker formats, for every given shape. Nevertheless, as will be discussed in the subsequent sections, certain findings and outcomes from this study are also applicable to the training of low-rank neural networks from scratch, without relying on any prior knowledge.

3.4 Alternative Problem Formulations: Surrogate and Relaxation

This section presents two alternative formulations for the tensor geometry optimization problem. First, we present an upper-bound surrogate model. This model shows the relationship of the most feasible balanced shape with the general problem statement. This surrogate model can also replace the original model when no teacher model is available and computing the tensor ranks is infeasible. Next we present a relaxed model that replaces the constraints to expand the decision space and to allow solving the tensor shape optimization more efficiently.

3.4.1 Surrogate model

The hierarchical ranks in Eq.(3.9) and Eq.(3.10) are not known unless $\mathcal{W}_{\boldsymbol{\theta}}$ is decomposed by TT and Tucker for every shape $\boldsymbol{\theta}$. This requires that $\boldsymbol{W}$ is given from a teacher model that may not be available in practice. Furthermore, computing the ranks of $\mathcal{W}_{\boldsymbol{\theta}}$ for every $\boldsymbol{\theta}$ can cause a computational burden. In the following we show that a balanced shape is an upper-bound solution to the defined optimization models.

Proposition 1 *A potential balanced shape is a relaxed upper-bound solution for the tensor geometry optimization paradigm in Formulation 1 and 2.*

Proof: First, we derive the formulation for TT decomposition.

Let $\boldsymbol{v}^{(\boldsymbol{\theta})} = (r_0^{(\boldsymbol{\theta})} r_1^{(\boldsymbol{\theta})}, r_1^{(\boldsymbol{\theta})} r_2^{(\boldsymbol{\theta})}, \ldots, r_{d-1}^{(\boldsymbol{\theta})} r_d^{(\boldsymbol{\theta})})$. The cost function of Eq.(3.9) is written as below:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \boldsymbol{v}^{(\boldsymbol{\theta})} = \|\boldsymbol{\theta}\| \|\boldsymbol{v}^{\boldsymbol{\theta}}\| \cos\alpha, \tag{3.11}$$

where α is the angle between $\boldsymbol{\theta}$ and $\boldsymbol{v}^{\boldsymbol{\theta}}$. Given the accuracy of the decomposition is bounded by ϵ, there exist a constant C such that $\|\boldsymbol{v}^{\boldsymbol{\theta}}\| \leq C$ for all $\boldsymbol{\theta}$ and $\boldsymbol{v}^{\boldsymbol{\theta}}$ which satisfies the error boundary. Therefore we can minimize the upper boundary of Eq.(3.11) given $\|\boldsymbol{\theta}\|\|\boldsymbol{v}^{\boldsymbol{\theta}}\| \cos\alpha \leq C\|\boldsymbol{\theta}\|$ where $\cos\alpha \leq 1$. In many decomposition methods the ranks are determined prior to the decomposition. Here we do not fix the ranks but we assume an upper boundary ($C \geq \max_{\boldsymbol{\theta}} \|\boldsymbol{v}^{\boldsymbol{\theta}}\|$) for the norm of the vector of ranks, $\boldsymbol{v}^{\boldsymbol{\theta}}$. It is almost a relaxation in comparison to fixing the ranks and it allows flexibility in the values of the actual ranks derived from the ϵ-bounded decomposition. The case that ranks are considered fixed is a special case of the above derivation. Meantime, since $\theta_i \in \mathbb{Z}$ for $i \in (1, 2, \cdots, d)$, we have $\|\boldsymbol{\theta}\| \leq \sum_{j=1}^{d} \boldsymbol{\theta}$. Therefore, we can write:

$$\begin{aligned} &\min \sum_{j=1}^{d} \theta_j, \\ \text{subject to } &\prod_{j=1}^{k} \theta_j = M, \prod_{j=k+1}^{d} \theta_j = N, \\ &\theta_j \geq 2, \theta_j \in \mathbb{Z}, j = 1, \cdots, d. \end{aligned} \tag{3.12}$$

Eq.(3.12) can be partitioned into the summation of two independent optimization problems that both fall in the class of minimization of the sum under product constraints [80]. According to AM-GM inequality, the potential solution is $\theta_j = \theta_i$ for $i, j \in 1, 2, \cdots, k$ and similarly for $i, j \in k+1, \cdots, d$, if the integer constraints are relaxed. If instead of two constraints we had one single constraint where $\prod_{i=1}^{d} \theta_i = t = M \times N$, then the solution would be $\theta_i = \theta_j \ \forall \ i, j \in \{1, 2, \cdots, d\}$.

For Tucker decomposition Eq.(3.12) holds. let $\boldsymbol{v}^{(\boldsymbol{\theta})} = (r_0^{(\boldsymbol{\theta})}, r_1^{(\boldsymbol{\theta})}, \cdots, r_d^{(\boldsymbol{\theta})}, \prod_{j=1}^{d} r_j^{(\boldsymbol{\theta})})$ and $\boldsymbol{\theta}^{'} = (\theta_1, \cdots, \theta_d, 1)$. Like TT, Eq.(3.11) is derived and is followed by Eq.(3.12) replacing $\boldsymbol{\theta}^{'}$ for $\boldsymbol{\theta}$ for Tucker decomposition. Given the last element of $\boldsymbol{\theta}^{'}$ is a constant,

Eq.(3.12) holds for Tucker decomposition too.

■

3.4.2 Relaxed model

The results of our previous studies demonstrated that adding dummy elements to make some tensor geometries (other than those derived from prime factorization) possible may result in a better compression and space saving [16]. So here we are interested to study a relaxed version of the tensor geometry optimization where the equity constraints are replaced by the inequality constrains. For the TT format the relaxed optimization model is presented below:

$$\begin{aligned} \min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) &= \sum_{j=1}^{d} r_{j-1}^{(\boldsymbol{\theta})} \times \theta_j \times r_j^{(\boldsymbol{\theta})}, \\ \text{subject to } \prod_{j=1}^{k} \theta_j \geq M,\ &\prod_{j=k+1}^{d} \theta_j \geq N, \\ \theta_j \geq 2, \theta_j \in \mathbb{Z}, j = 1, \cdots, d. & \end{aligned} \tag{3.13}$$

The same change can be applied for Tucker decomposition by replacing the cost function of Eq.(3.13) by the cost function of Eq.(3.10) (see Appendix A).

This relaxed formulation allows a possible shape that may result in a tensor, $\mathcal{W}_{\boldsymbol{\theta}}$, whose size is greater than that of the original weight matrix $\boldsymbol{W}$. If a shape $\boldsymbol{\theta}$ poses a tensor whose size is greater than that of the weight matrix, the resized tensor is filled with dummy elements (e.g., zeros). The cost function favors shapes with a size closest to that of the given parameters to achieve better compression. However, relaxing the constraints may lead to more efficient solutions. It is important to note that such relaxation does not

increase the cost or difficulty of implementation. If $\prod_{j=1}^{k} \theta_j \geq M$ then the input is simply padded with zeros to match the size and if the solution make $\prod_{j=k+1}^{d} \theta_j \geq N$ then the extra outputs are just simply ignored. At the first glance adding to the parameter size may seem to cause a higher computational and memory cost, but in the results section it is shown that such addition in fact may reduce the memory and computation costs due to a better (smaller) low-rank factor size.

3.5 Tensor Geometry Optimization

In this section, we present methods to solve the defined optimization models. First, we apply a graph optimization for the original problem. Next, we present a dynamic programming approach to solve the surrogate model. Finally we present a random search to solve the relaxed model.

3.5.1 Graph optimization for the original model (OM)

Solving Eq.(3.9) and Eq.(3.10) are challenging. The memory complexity of a low-rank layer depends on the ranks of the weight tensor and reordering a tensor changes its ranks and determining the ranks is an NP-hard problem[59]. In other words, the hierarchical ranks are not known unless $\mathcal{W}_{\boldsymbol{\theta}}$ is decomposed by TT and Tucker for every shape $\boldsymbol{\theta}$. On the other hand, the constraints of the model limit the feasible domain to the shapes resulted from the prime factorizations of N and M that makes the problem tractable. Therefore, we initially represent the feasible domain of the problem with acyclic directed graphs and then apply graph traversal algorithms to study the memory complexities of the factors resulted from all feasible geometries.

Let us define a tensor geometry graph (TGG) with two parameters including tensor size denoted by R and tensor order denoted by l as below:

$$
\begin{aligned}
&G_{R,l} = (V, E),\\
&V = V_0 \cup \bigcup_{i=1}^{l-1} V_i \cup V_l,\ E = \bigcup_{i=1}^{l} E_i, V_0 = \{R\}, V_l = \{1\},\\
&V_i = \bigcup_{v \in V_{i-1}} \Lambda(v), \Lambda(v) = \left\{ \prod_j p_j^{a_j} \;\middle|\; \begin{array}{l} v = \prod_j p_j^{m_j}, m_j \in \mathbb{Z}, \\ a_j \leq m_j,\ a_j \in \mathbb{Z}, \\ \prod_j p_j^{a_j} \neq \{1, v\} \end{array} \right\},\\
&E_i = \left\{ e_{v_j, v_i} \;\middle|\; \begin{array}{l} e_{v_j,v_i} = \frac{v_j}{v_i}, \\ v_j \equiv 0 \pmod{v_i}, \\ v_j \in V_{i-1},\ v_i \in V_i \end{array} \right\}.
\end{aligned} \tag{3.14}
$$

Here $R, l \in \mathbb{Z}, R > 1, l \geq 1$. Each conceivable trajectory from V_0 to V_l, $\boldsymbol{\tau} = (e_{R,v_1}, \cdots, e_{v_{l-1},1})$, represents a sequence of l edges and we have $\prod_{t=1}^{l} \tau_t = R$. We extract all such trajectories of $G_{R,l}$ using a graph traversal algorithm traversing from V_0 to V_l and the set $T(G_{R,l})$ refers to a collection of all those trajectories.

Algorithm 4 demonstrates the graph-based geometry optimization (GGO) applied to solve the original problems defined in Eq.(3.9) and Eq.(3.10). In Algorithm 4, two TGGs ($G_{M,k}$ and $G_{N,d-k}$) are developed where V_0 of $G_{M,k}$ equates M and V_0 of $G_{N,d-k}$ equates N. A depth first search (DFS) algorithm [81] is applied to extract all trajectories from the initial vertex (V_0) to the end vertex (V_l) of each $G_{M,k}$ and $G_{N,d-k}$, denoted by $T(G_{M,k})$ and $T(G_{N,d-k})$, respectively. $T(G_{M,k})$ and $T(G_{N,d-k})$ are partial solutions. All combinations of $T(G_{M,k})$ and $T(G_{N,d-k})$ constitute the feasible decision space of Eq.(3.9) or Eq.(3.10), denoted by the set Θ. For every feasible shape, $\boldsymbol{\theta} \in \Theta$, then the tensorized weight matrix, $\mathcal{W}_{\boldsymbol{\theta}}$, is decomposed and its actual hierarchical ranks are computed and so their actual memory requirements, $f(\boldsymbol{\theta})$. The shape that minimizes $f(\boldsymbol{\theta})$ is returned.

Algorithm 4 Geometry graph optimization (GGO)

Require: M, N, k, d

$\Theta \leftarrow \varnothing$
Construct two TGGs $G_{M,k}$ and $G_{N,d-k}$ ▷ (Eq.(3.14))
$T(G_{M,k}) \leftarrow DFS(G_{M,k})$
$T(G_{N,d-k}) \leftarrow DFS(G_{N,d-k})$ ▷ (DFS of [81])
for $\boldsymbol{\tau} = (x_1, \cdots, x_k) \in T(G_{M,k})$ **do**
    for $\boldsymbol{\tau}^{'} = (x_1^{'}, \cdots, x_{d-k}^{'}) \in T(G_{N,d-k})$ **do**
        $\boldsymbol{\theta} \leftarrow (x_1, \cdots, x_k, x_1^{'}, \cdots, x_{d-k}^{'})$
        $\Theta \leftarrow \Theta \cup \boldsymbol{\theta}$
    end for
end for
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}^{'} \in \Theta, f(\boldsymbol{\theta}^{'}) \leq f(\boldsymbol{\theta}), \forall \boldsymbol{\theta} \in \Theta$ ▷ (Eq.(3.9) and Eq.(3.10))
Return $\boldsymbol{\theta}^*$

3.5.2 Dynamic programming for the surrogate model (SM)

The symmetric general solution may not satisfy $\theta_i \in \mathbb{Z}$ or $\theta_j \geq 2$ constraints in Eq.(3.12). Therefor we need to find the most feasible balanced shape that minimizes Eq.(3.12) that is an integer linear programming. In this work, given we already developed the graph of the feasible space in the previous section, we apply a dynamic programming to find the optimum shape as described in Algorithm 5.

Algorithm 5 Geometry dynamic programming (GDP)

Require: $T(G_{M,k})$, $T(G_{N,d-k})$

Define $g(\boldsymbol{x}) = \min_{x_1}(x_1 + g((x_2, \cdots, x_d))), \forall \boldsymbol{x} \in \mathbb{Z}^n, g(\varnothing) = 0$
$\boldsymbol{\tau} = argmin_{\boldsymbol{x} \in T(G_{M,k})} g(\boldsymbol{x})$
$\boldsymbol{\tau}^{'} = argmin_{\boldsymbol{x} \in T(G_{N,d-k})} g(\boldsymbol{x})$
$\boldsymbol{\theta}^* \leftarrow (\tau_1, \cdots, \tau_k, \tau^{'}, \cdots, \tau_{d-k}^{'})$
Return $\boldsymbol{\theta}^*$

Algorithm 5 receives all paths of the tensor geometry graphs $G_{M,k}$ and $G_{N,d-k}$ and applies a dynamic programming to find the path that minimizes the sum of the edges along the path for each graph. Next concatenate the partial solutions to build the optimum shape. Algorithm 5 does not simulate tensor decomposition to compute ranks

and the solution for both the TT and Tucker formats are the same.

3.5.3 Random search for the relaxed model (RM)

Applying the inequality constraints makes the decision space significantly larger, therefore, we did not apply the a graph optimization for this formulation. Instead, we applied a random search algorithm inspired by a genetic algorithm [54] to approximate an optimal solution for Eq.(3.13) and Eq.(A.1) with computing exact hierarchical ranks for every trial solution. We refer to this method as random geometry search (RGS). The details of the implementation of the RGS is presented in Algorithm 6.

The RGS begins by generating P initial random solutions with the best solution is stored. Additionally, $S-1$ (where $S < P$) solutions out of the P generated solutions are randomly selected and they are also stored. For selection, a probability function π is applied that assigns a probability proportional to the cost function. To generate a new solution, from the S stored solutions, two solutions like $\boldsymbol{\theta} = (\theta_1, \cdots, \theta_k, \cdots, \theta_d)$ and $\boldsymbol{\theta}^{'} = (\theta_1^{'}, \cdots, \theta_k^{'}, \cdots, \theta_d^{'})$ are randomly selected using a uniform distribution. Next, two new trial solutions are generated using a single-point crossover at k-th element, which exchanges the variables of the two selected solutions as below:

$$\begin{aligned} \boldsymbol{\theta}_1^{new} &= (\theta_1, \cdots, \theta_k, \theta_{k+1}^{'}, \cdots, \theta_d^{'}), \\ \boldsymbol{\theta}_2^{new} &= (\theta_1^{'}, \cdots, \theta_k^{'}, \theta_{k+1}, \cdots, \theta_d). \end{aligned} \tag{3.15}$$

After new solutions are generated using Eq.(3.15), a new solution like $\boldsymbol{\theta}$ is muted as

Algorithm 6 Random geometry search (RGS)

Require: P, I, S, M, N, k, d

$\Gamma \leftarrow \varnothing$
for $j = 1$ to P **do**
  $\boldsymbol{\theta} \leftarrow RSC(M, N, k, d)$ ▷ Algorithm 7
  $\Gamma \leftarrow \Gamma \cup \boldsymbol{\theta}$
end for
$\boldsymbol{\theta}^{\star} \leftarrow \boldsymbol{\theta} \in \Gamma,\ f(\boldsymbol{\theta}) \leq f(\boldsymbol{\theta}^{'}), \forall \boldsymbol{\theta}^{'} \in \Gamma$
for $t = 1$ to I **do**
  $\Upsilon \leftarrow \{(S-1)$ randomly selected $\boldsymbol{\theta}s$ from Γ with probability distribution $\pi(\boldsymbol{\theta}) = \frac{f(\boldsymbol{\theta})}{\sum_{\boldsymbol{\theta} \in \Gamma} f(\boldsymbol{\theta})}\}$
  $\Upsilon \leftarrow \Upsilon \cup \boldsymbol{\theta}^{*}$
  $\Upsilon^{'} \leftarrow \{(P-S)$ new shapes generated by crossover and mutation$\}$
  $\Gamma \leftarrow \Upsilon \cup \Upsilon^{'}$
  $\boldsymbol{b} \leftarrow \boldsymbol{\theta}$ If $f(\boldsymbol{\theta}) \leq f(\boldsymbol{\theta}_j)$ & $\boldsymbol{\theta} \in \Gamma\ \forall j = 1, \cdots, P$
  if $f(\boldsymbol{b}) \leq f(\boldsymbol{\theta}^{\star})$ **then**
    $\boldsymbol{\theta}^{\star} \leftarrow \boldsymbol{b}$
  end if
end for
Return $\boldsymbol{\theta}^{\star}$

below:

$$\boldsymbol{\theta}^{new} = \begin{cases} (\theta_1, \cdots, \theta_k, \theta^{''}_{k+1} \cdots, \theta^{''}_d) & 0 < rand \leq 0.5 p_m \\ (\theta^{''}_1, \cdots, \theta^{''}_k, \theta_{k+1} \cdots, \theta_d) & 0.5 p_m < rand \leq p_m \\ (\theta_1, \cdots, \theta_k, \theta_{k+1} \cdots, \theta_d) & rand > p_m \end{cases}, \tag{3.16}$$

where $\boldsymbol{\theta}^{''} = (\theta^{''}_1, \cdots, \theta^{''}_k, \cdots, \theta^{''}_d)$ is a randomly generated solution by the random shape constructor (RSC) presented in Algorithm 6, $rand = Unif(0, 1)$, and p_m is a threshold in range $[0, 1]$. The process of selection and reproduction repeats for I iterations as presented in Algorithm 6. The best found solution is reported. Algorithm 7 is used for generating random solution to ensure the search stay in the feasible space throughout the random search iterations and also to avoid searching among unnecessarily large tensor sizes.

Algorithm 7 Random shape constructor (RSC)

Require: M, N, k, d

$u \leftarrow \lceil \frac{M}{2^k} \rceil$
for $j = 1$ to $j = k - 1$ **do**
    $\theta_j \leftarrow RandInit(2, u)$
    $u \leftarrow \lceil \frac{2u}{\theta_j} \rceil$
end for
$\theta_k \leftarrow \lceil \frac{M}{\prod_{j=1}^{k-1} \theta_j} \rceil$
$u \leftarrow \lceil \frac{N}{2^{(d-k)}} \rceil$
for $j = k + 1$ to $j = d - 1$ **do**
    $\theta_j \leftarrow RandInit(2, u)$
    $u \leftarrow \lceil \frac{2u}{\theta_j} \rceil$
end for
$\theta_d \leftarrow \lceil \frac{N}{\prod_{j=k+1}^{d-1} \theta_j} \rceil$
Return $(\theta_1, \cdots, \theta_k, \cdots, \theta_d)$

3.6 Experiments and Results

We have conducted several experiments to demonstrate the applicability and functionality of the developed models. The applied networks include a two-layer dense network for MNIST data set [62] and a VGG network for CIFAR-10 data set [82]. First, we studied the effect of tensor shape on low-rank compression (Experiment I 3.6.1). Then, we applied the proposed method for end-to-end tensorized MNIST network (Experiment II 3.6.2). Next, we investigated the effect of GSI on tensorized training of the MNIST network (Experiment III 3.6.3). Finally, we applied the proposed method to design end-to-end tensorized VGG network for CIFAR-10 (Experiment IV 3.6.4).

For Experiments I 3.6.1, II 3.6.2, and III 3.6.3 we used a two-layer network trained using the MNIST data set. The applied network consists of two dense layers. Since the input images are 28 by 28 pixels, the input size to the network is 784. The first layer has 512 rectifier linear unit (ReLU) cells and the second layer has 10 Softmax units. Consequently, the total number of parameters of the network is 407,050. The MNIST

network was initially trained with a a loss of 0.0376 and 0.5542, and an accuracy of 98.65% and 90.08% for train and validation sets, respectively.

For Experiment IV 3.6.4, a Visual Geometry Group (VGG) network was applied to solve CIFAR-10 dataset. The network consists of 6 convolutional layers, 2 dense layers, a maxpooling and a dropout after every two convolutional layers and a dropout layer before the last dense layer. The first two convolutional layers have 32 filters, the next two convolutional layers have 64 filters, and the last two convolutional layers have 128 filters. All convolutional layers have filter size 3 by 3 and stride 1 and padding is the same. The final layers are two dense layers. The first dense layer consists of 512 relu units and the second one has 10 softmax units. The total number of parameters of the network is 1,341,226.

Throughout the result section and the following experiments, SM-TT, OM-TT, and RM-TT refer to the surrogate model (Eq.(3.12)) solved by the GDP, the original model (Eq.(3.9)) solved by the GGO, and relaxed model (Eq.(3.13)) solved by the RGS, respectively, in the TT format. Similarly, SM-Tucker, OM-Tucker, and RM-Tucker refer to Eq.(3.12) solved by the GDP, Eq.(3.10) solved by the GGO, and relaxed model solved by the RGS, respectively, in the Tucker format.

3.6.1 Experiment I: Tensor geometry effect

Initially, we studied whether the geometry used for folding the matrix of weights into a tensor of higher dimension affects the compression and, if so, to what extent. In order to answer this question and visualize the effect of tensor reshaping on compression and space saving, all possible shapes of dimension three for the weight matrix of the first dense layer of the MNIST network 3.6 are extracted using prim factorization. Figures 3.1 and 3.2 illustrate the space saving with respect to the normalized sum of dimensions

for the TT and Tucker formats, respectively. The space saving is calculated as below:

$$Space\ saving = 1 - \frac{f(\boldsymbol{\theta})}{size(\mathcal{W})}, \tag{3.17}$$

where $\mathcal{W}$ is the tensor of weights, which is folded with $\boldsymbol{\theta}$, and $f(\boldsymbol{\theta})$ denotes the unconstrained cost functions of Eq.(3.9) and Eq.(3.10) for the TT and Tucker formats, respectively.

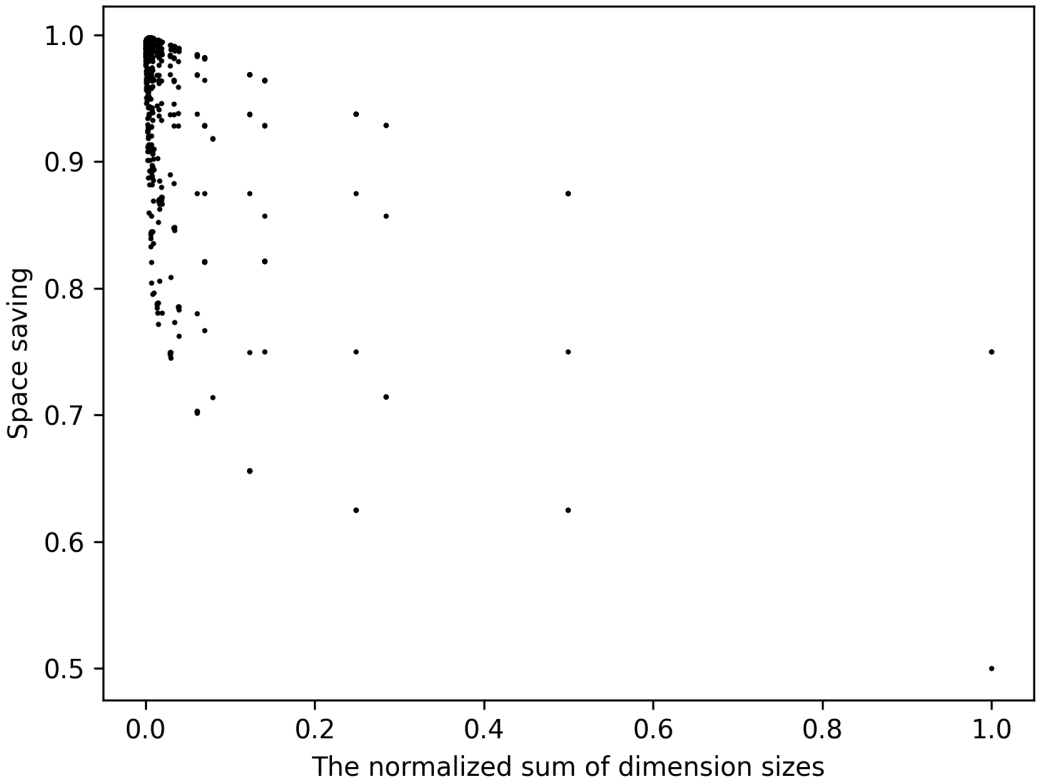

Figure 3.1: The normalized sum of dimensions is plotted against the space saving achieved through the TT decomposition for all possible shapes of dimension 3 with a size of 401,408 and $\epsilon = 1.0$ for the weights of MNIST's ReLU layer.

In Figures 3.1 and 3.2, it seen that the shape of the tensor significantly affects the

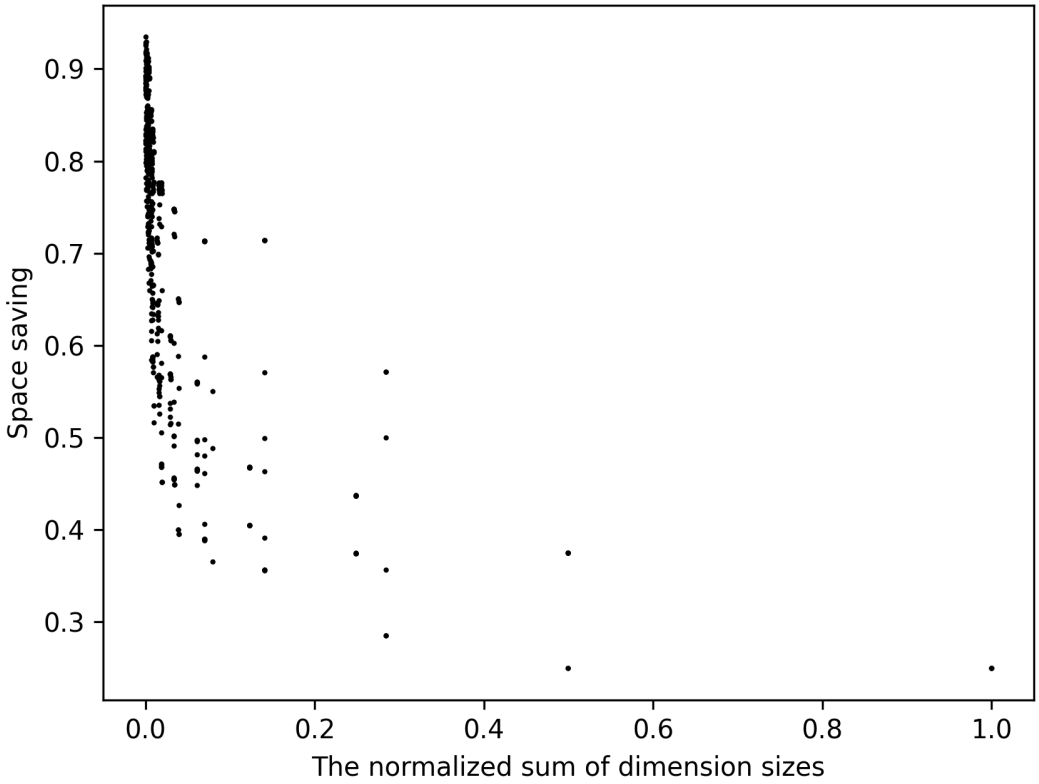

Figure 3.2: The normalized sum of dimensions is plotted against the space saving achieved through the Tucker decomposition for all possible shapes of dimension 3 with a size of 401,408 and $\epsilon = 1.0$ for the weights of MNIST's ReLU layer.

space saving. For the Tucker decomposition, the worst shape poses only less than 30% space saving while the best shapes have higher than 90% space savings for $\epsilon = 1$. Almost the same pattern can be seen for the TT decomposition. Meantime, we observe a clear pattern between the sum of dimension sizes and the space saving, such that minimizing the sum avoid shapes with low space saving and may lead to a good space saving. The observed pattern supports the surrogate model.

3.6.2 Experiment II: End-to-end tensorized MNIST

In this experiment we applied different optimization methods to find an optimum shape for tensor compression of both layers of the MNIST network and the initialization of the tensorized training of the network. For each layer, after finding an optimum shape based on an error bound, the layer is compressed accordingly. The entire network is tensorized end-to-end, and the calculated factors are applied for the initialization. The MNIST network has 407,050 parameters in total, with 401,408 are the weights of the first layer, 522 are the total number of biases, and the rest accounting for the weights of the second layer. For each layer only weights are represented with their factors and the biases are not compressed. Due to the error of the compression, the tensorized network's accuracy may drop. Therefore, after initialization of the tensorized network, the low-rank network is retrained. Given the factorization, the retraining has significantly less number of parameters in comparison to the initial training. For this experiment we set $d = 4$, and $k = 2$. In other words, the decision space of the optimization problems consists of tensors of order 4 and the first two dimensions represent the input shape and the last two dimensions represent the output shape. We used $\epsilon = 0.8$ for the TT decomposition and $\epsilon = 1$ for the Tucker decomposition. These error bounds were chosen to achieve the best balance between compression and accuracy for each decomposition method.

Table 3.1 lists the network's training and validation accuracy, the total number of parameters of the network, and the number of MACs of the network's inference for different methods. In Table 3.1 it is seen that all tensorized networks achieved almost identical validation accuracy compared to the base (uncompressed) model. It is important to note that the training accuracy can be improved to match that of the base model. However, such an improvement does not enhance the validation accuracy. Therefore we here preferred models with higher validation accuracy than those with a better training

Table 3.1: The training and validation accuracy, number of parameters, and number of MAC operations of different geometry search methods for MNIST in comparison to the base model with the TT and Tucker decomposition methods.

Method	Train	Validation	#Parameters	#MACs
Base	98.65	90.08	407,050	406,528
SM-TT	94.60	88.74	38,950	58,984
OM-TT	94.33	88.47	36,646	82,768
RM-TT	93.96	88.40	36,273	78,119
SM-Tucker	93.20	89.04	36,074	60,840
OM-Tucker	93.90	88.94	34,906	65,074
RM-Tucker	93.75	88.88	33,962	63,471

accuracy. It is also seen that the low-rank tensorized networks have about 10 times less memory requirement than the base model. The number of MACs reduced by about 5 to 10 times for the tensorized models. Comparing the SM, OM, and RM, it is clear that there is a significant difference among the methods and the optimal tensor geometries. For both the TT and Tucker formats, SM has the worst result. The OM provides an intermediate result and the RM provided the best compression. Table 3.2 lists the optimal shapes, the corresponding ranks, and space savings resulted from all methods for the first layer of the MNIST network. It is noteworthy that the solution found by the OM and RM are both far from the most feasible balanced shape. More interestingly, the solution found by the RM resulted in a tensor size slightly larger than the original weight matrix size but it leads to the most efficient compression.

The optimal solutions reduced the number of MACs required for the inference of the MNIST network as well. Generally, the lower number of MACs across all low-rank layers are primarily due to three reasons: 1- the efficient low-rank inference implemented by tensor network contraction, 2- the smaller number of low-rank factors by minimizing the number of factors, and 3- adaptation of the tensor geometry optimization model that address the constraints for tensor dimension sizes for inputs and outputs that facilitates

Table 3.2: The optimum dimension sizes, ranks, and space saving for the first layer of the MNIST network.

Method	Dimensions	Ranks	Saving (%)
Base	(784,512)	-	0
SM-TT	(28,28,16,32)	(1,6,66,23,1)	90.85
OM-TT	(112,7,4,128)	(1,17,56,73,1)	91.35
RM-TT	(112,7,6,86)	(1,17,60,56,1)	91.41
SM-Tucker	(28,28,16,32)	(10,12,12,22)	91.61
OM-Tucker	(28,28,8,64)	(10,12,6,40)	91.90
RM-Tucker	(28,28,26,20)	(10,12,18,14)	92.01

efficient implementation of low-rank layers. The inference of the low-rank tensorized networks is between 5 to 7 times faster than the base uncompressed model.

On an Intel Core i7 CPU, the inference time of the base (uncompressed) MNIST network for the training data set, is about 2 seconds. For all the low-rank tensorized networks the inference time reduces to about 1 second with Tucker models be slightly slower than the TT ones. It is important to note that our current code is written in Python and the inference time for low-rank models does not show the actual and true latency considering the overhead of Python compilation. We utilized TensorFlow for implementation of the associated tensor contractions in a low-rank layer, but the low-rank layer uses several loops to implement the tensor network that increases latency. However, this speedup shows promises for future works to optimize the codes and compilation to achieve faster inference time.

3.6.3 Experiment III: Initialization effect

In this experiment, we investigated how the proposed geometry based initialization (GSI) affects the tensorized training. We initialized the first layer of MNIST with the solution of the OM which suggested a shape (112,7,4,128) with ranks (1,17,56,73,1) for the TT and a shape (28,28,8,64) and ranks (10,12,6,40) for Tucker. We compare the

results with an arbitrary shape (4,196,256,2) and TT ranks (1,9,9,9,1) and Tucker ranks (13,13,13,13) which have been selected such that the number of low-rank factors be almost equal to (slightly larger than) those from the OM. We refer to the initialization with the arbitrary shape as random initialization (RI). Fig. 3.3 and Fig. 3.4 depict the accuracy of the network with respect to the number of epochs for TT and Tucker, respectively. Table 3.3 lists the loss and accuracy of the MNIST network initialized with the GSI and RI. In both cases, the network that initialized with the OM achieved a significantly higher accuracy. After 100 epochs the low-rank tensorized network initialized with the RI was unable to match the accuracy of the low-rank network initialized with the GSI. This example demonstrates that the proposed tensor geometry optimization not only reduces memory complexities but also has the potential to yield better accuracy for low-rank tensorized networks.

In previous studies it has been discussed that initialization of the tensorized networks is challenging and may lead to training stagnation [7]. In the presence of a teacher model, the proposed initialization can assist the low-rank network by guiding it toward the teacher model. In the absence of a teacher model, the SM algorithm can be employed to at least provide an upper bound solution, preventing unfavorable shapes that may hinder achieving good accuracy. After applying tensor shape optimization (i.e., the GSI), additional training methods, such as auto rank selection, can be employed to further optimize the ranks and compression efficiency of low-rank tensorized networks [10, 7].

3.6.4 Experiment IV: End-to-end tensorized VGG for CIFAR10

In this experiment, we applied the proposed tensor geometry optimization paradigm to design an end-to-end tensorized VGG for CIFAR10 dataset. Similar to Experiment II 3.6.2, we first trained the base (uncompressed) VGG model 3.6. Subsequently, for

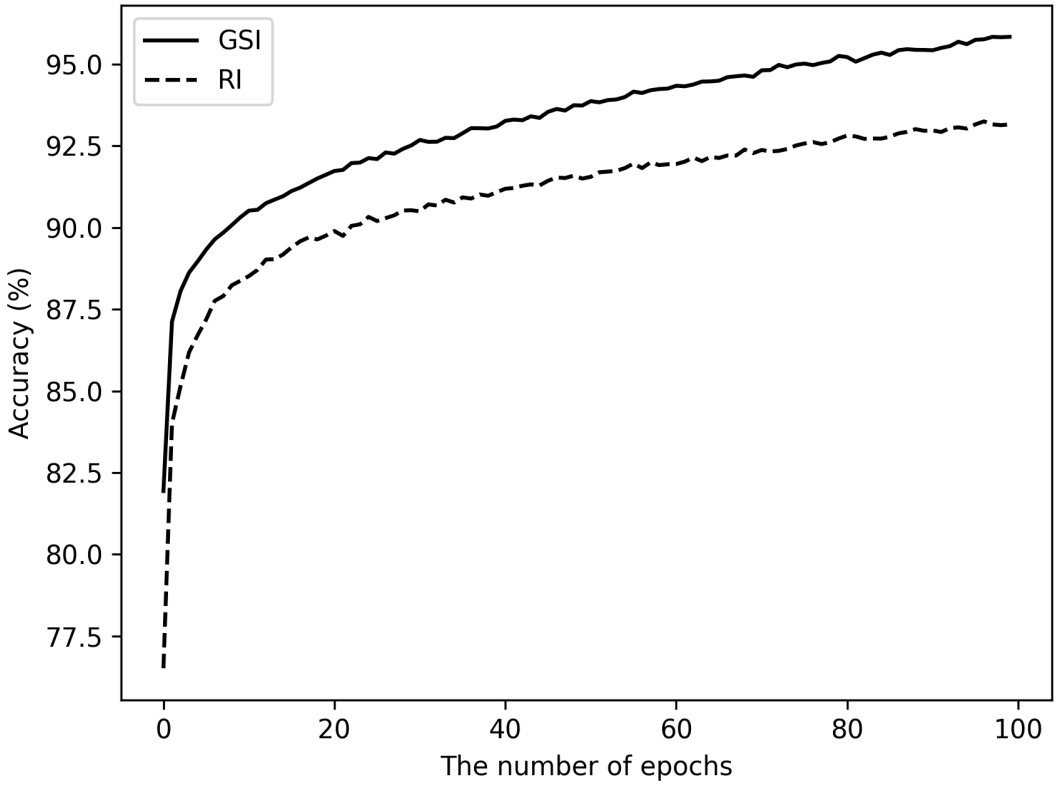

Figure 3.3: The effect of initialization on the accuracy of the MNIST network when the first layer is tensorized using the TT format.

each layer of the network, the weights are extracted and an optimal shape is found using the different optimization techniques. With the GSI technique, the tensorized network is initialized with the low-rank factors with the optimal shape. Next the low-rank tensorized network is retrained. We used $\epsilon = 0.6$ and $\epsilon = 0.8$ for compressing convolutional layers with the TT and Tucker formats, respectively. We used $\epsilon = 0.8$ and $\epsilon = 1$ for compressing dense layers with the TT and Tucker formats, respectively. These error bounds were chosen to strike the best balance between compression and accuracy for each decomposition method.

Table 3.4 presents the accuracy, number of parameters, and number of MACs for the

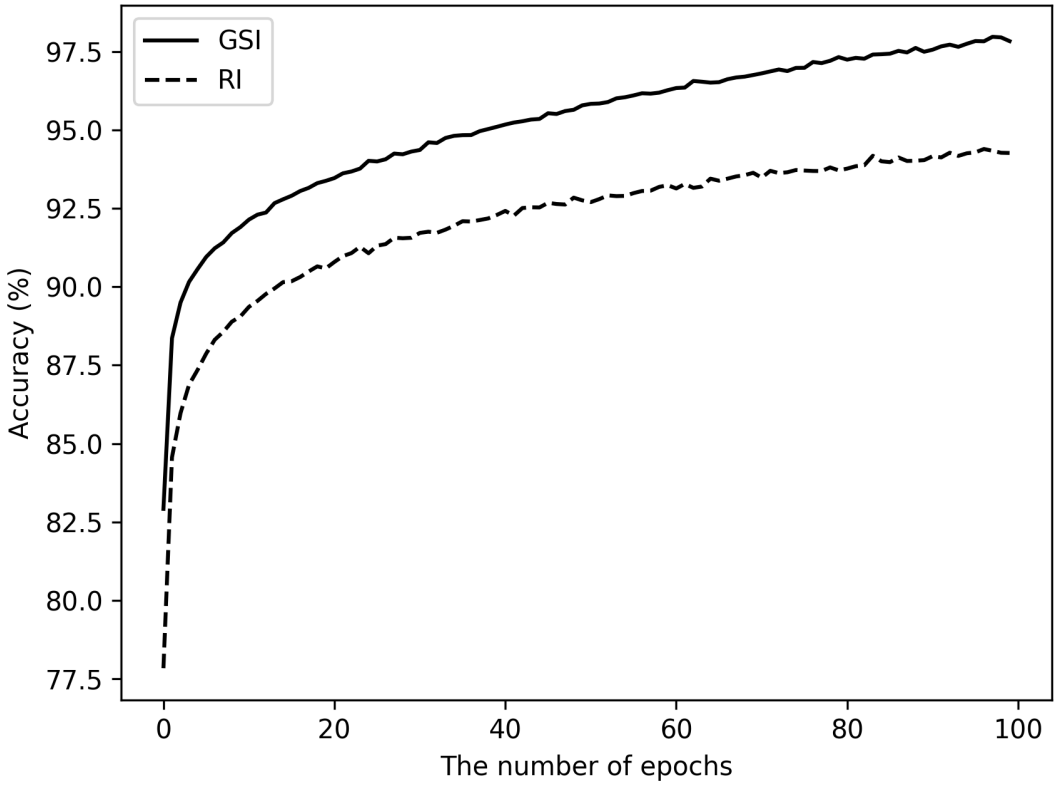

Figure 3.4: The effect of initialization on the accuracy of the MNIST network when the first layer is tensorized using the Tucker format.

base and the end-to-end tensorized networks using TT and Tucker with different optimization methods. The uncompressed model has a total number of approximately 1.341 million parameters, and one inference of the network requires about 39.3 million MACs. In Table 3.4 it is evident that the low-rank tensorized networks reduce the memory cost by about 4 times without a significant drop in accuracy. The tensor geometry optimization for TT increases the memory savings from 4 times (with SM-TT) to about 5.5 times (with RM-TT). Likewise, for the Tucker format, the memory saving has decreased from 372 thousands to 277 thousands. In a head-to-head comparison between the OM and SM, the OM reduces the memory requirements of the tensorized network by almost 20%.

Table 3.3: The loss and accuracy of the MNIST network with the first dense layer tensorized using the TT and Tucker formats using the GSI and RI methods after 100 epochs.

Method	Train (%)	Validation (%)	Loss
GSI-TT	95.84	88.17	0.112
RI-TT	93.16	86.44	0.177
GSI-Tucker	97.83	88.68	0.060
RI-Tucker	94.26	87.37	0.149

Table 3.4: The training and validation accuracy, number of parameters, and MAC operations of tensorized VGG for different geometry search methods with the TT and Tucker formats compared to the base (uncompressed) model.

Method	Train	Validation	#Parameters	#MACs
Base	98.80	83.21	1.341 M	39.3 M
SM-TT	99.77	83.43	0.335 M	30.3 M
OM-TT	99.81	82.89	0.246 M	24.4 M
RM-TT	99.97	82.97	0.246 M	22.6 M
SM-Tucker	99.90	83.58	0.372 M	38.1 M
OM-Tucker	99.96	83.35	0.290 M	19.5 M
RM-Tucker	99.87	83.22	0.277 M	18.2 M

In addition to memory savings, the low-rank tensorized networks designed with the tensor geometry optimization have lower MAC counts in comparison to the base model for both TT and Tucker. We also observe a significant difference in MAC counts across the optimization models. For the SM method (using the balanced shape) the MAC count is the worst (highest) among the three optimization methods and the best MAC count has been achieved by the RM method, which, in comparison to the base model, speeds up the inference by about 2 times. The OM reduces the time complexity of the tensorized network by approximately 50% compared to the SM solution for both TT and Tucker.

On an Intel Core i7 CPU, the inference time of the base VGG model was 39 seconds. For the low-rank networks based on TT, the inference times reduced to approximately 32 seconds with no significant differences observed between the ILS, OM, and RM. For the

low-rank networks based on the Tucker format, the inference times were about 44 seconds. It is essential to note that our current code is written in Python and the reported inference times for the low-rank models do not accurately reflect the actual and true latency due to the overhead of Python compilation. While we utilized TensorFlow for tensor contraction operations, the implemented tensor network involves several loops, leading to increased latency. The inference times for low-rank models reveal that compilation overhead, rather than actual MACs or memory operations, dominates the performance. The slower execution of the Tucker format compared to TT can be traced to the one extra call of tensor contraction operation within the the tensor network implementaton specific to the Tucker format. Our Python benchmarking using TensorFlow validates this observation. When expanding a sequence of n random tensor contractions to n+1 while keeping MACs and total elements nearly constant, a higher latency is observed for the longer sequence. This latency is only related to Python dynamic compilation and not our proposed low-rank tensorized product. Therefore, optimizing compilation and the codes of the low-rank tensorized product, particularly in a more efficient language like C++, holds promise for reducing the inference time. Improvement of the inference latency through optimization of compilation and the codes, as well as exploring parallelization on GPUs or FPGAs are suggested for future studies.

3.7 Discussion

In low-rank tensorized neural networks, it is common to choose a balanced shape for folding a weight matrix into a higher-dimensional tensor. In this study, we demonstrated that the most feasible balanced shape is, in fact, an upper bound solution to a more general tensor geometry optimization model, and the proposed SM (surrogate model) method can be used to find such a balanced shape. Although a balanced shape can be a

good choice when there is no teacher model, in the presence of a teacher model, low-rank tensorized neural networks can be designed to have lower memory requirements and time complexities by applying the presented tensor geometry optimization.

The OM (original model) searches among all feasible geometries built by prime factorization. In contrast, the RM (relaxed model) extends the search beyond prime factorization. We observed that in the studied networks, the relaxed model led to a better compression and time complexities compared to the OM. Despite the fact that the optimal shape found by the RM may appear to have a greater size, the associated low-rank factors caused a better memory saving. However, the OM might be faster than the RM, and it guarantees to find the best feasible shape. For instance, for the ReLU layer of the MNIST network, on a 2.6 GHz Intel Core i7 processor, the wall time of the RM was in average about 10 minutes. However, using the same processor, the wall time of the OM was about 0.5 minutes. Meanwhile, the RM involves randomness and it does not guarantee to find a global optimum. The choice among the SM, OM, or RM depends on the application, with each method offering its own set of advantages and disadvantages. Above all we emphasize the importance of the tensor geometry optimization and its impact on designing low-rank tensorized neural networks, irrespective of the optimization method being applied.

In this work, we observed that the designed low-rank tensorized neural networks through the tensor geometry optimization exhibited lower time complexities compared to the uncompressed models. Although we did not explicitly address minimization of time complexities in the proposed optimization models, the constraints on dimensions were set to facilitate the efficient implementation of tensor network contractions for the low-rank neural networks. The formulation of the optimization problem to minimize time complexities is consequently suggested for future studies.

3.8 Conclusion

In this study we demonstrated the effect of the tensor geometry on efficient implementation of low-rank tensorized neural networks. Hence, we proposed a novel tensor geometry optimization paradigm designed to enhance the compression efficiency of end-to-end low-rank tensorized neural networks. We implemented the proposed methods for the TT and Tucker formats to compress the MNIST and VGG networks. We have applied various optimization techniques to solve the geometry optimization problem, including graph optimization, integer linear programming and random search.

The results of our study demonstrated that the tensor geometry optimization can significantly reduce the memory requirements of low-rank tensorized neural networks. Meanwhile, the proposed methods reduced the time complexities and achieved lower number of multiplication and accumulation for the inference of the low-rank tensorized neural networks compared to uncompressed networks. We also demonstrated that initializing tensorized training with a tensor geometry-based approach resulted in a better accuracy. Furthermore, the results demonstrated that the most feasible balanced tensor shape is an upper bound solution for the defined optimization model. Therefore, the most balanced shape may be sub-optimal, and tensor geometry optimization can offer a more efficient solution for low-rank neural networks. The results of this study provide additional insight into the role of tensor geometry in tensor compression, and the proposed methods can be applied to design memory-efficient implementations of neural networks for AI accelerators, particularly on resource-constrained devices.

Appendix A

Relaxed Model For Tucker

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \sum_{j=1}^{d} \theta_j \times r_j^{(\boldsymbol{\theta})} + \prod_{j=1}^{d} r_j^{(\boldsymbol{\theta})}$$

subject to

$$\prod_{j=1}^{k} \theta_j \geq M, \prod_{j=k+1}^{d} \theta_j \geq N,$$

$$\theta_j \geq 2, \theta_j \in \mathbb{Z}, j = 1, \cdots, d. \tag{A.1}$$

Appendix B

Time complexities of low-rank neural network inference

The number of MAC operations required for tensor contraction of two arbitrary real-valued tensors $\mathcal{A} \in \mathbb{R}^{a_1 \times \cdots \times a_l}$ and $\mathcal{B} \in \mathbb{R}^{b_1 \times \cdots \times b_h}$ is equal to $\prod_{t=1}^{l} a_t \prod_{t=1}^{h} b_t$ regardless of how many axes are contracted. Therefore, the total number of MACs for one inference of the low-rank tensorized product using the TT format is as follows:

$$\sum_{i=1}^{k} \frac{M r_{i-1} r_i}{\prod_{j=1}^{i-1} n_j} + \sum_{i=k+1}^{d} (\prod_{j=k+1}^{i} n_j) r_{i-1} r_i. \tag{B.1}$$

The total number of MACs for one inference of the low-rank tensorized product using the Tucker format is computed below:

$$\sum_{i=1}^{k} \frac{M \prod_{j=1}^{i} r_i}{\prod_{j=1}^{i-1} n_j} + \prod_{i=1}^{d} r_i + \sum_{i=k+1}^{d} \frac{\prod_{j=k+1}^{i} n_j \prod_{j=k+1}^{d} r_j}{\prod_{j=k+1}^{i-1} r_j}. \tag{B.2}$$

Following Eq.(B.1) and Eq.(B.2) we can compute the asymptotic time complexities of the inference of the low-rank tensozired product for the TT and Tucker formats.

Let $r > r_i \ \forall i$ be the maximum rank and $I \geq n_i \ \forall i$ be the maximum dimensions size. For the TT format we have:

$$Mr^2 \sum_{t=0}^{k} \frac{1}{I^t} + Nr^2 \sum_{t=0}^{d-k} \frac{1}{I^t}. \tag{B.3}$$

For $I > 1$ the geometric series above converge. Therefore, the time complexity for the TT format is equal to $O(max\{M, N\}\ r^2)$. For the Tucker format we have:

$$Mr^k \sum_{t=0}^{k-1} \frac{1}{r^t I^{k-1-t}} + r^d + Nr^{(d-k)} \sum_{t=1}^{d-k} \frac{1}{r^{t-1} I^{d-k-t}}. \tag{B.4}$$

If we assume $r \leq I$, then $\sum_{t=0}^{k-1} \frac{1}{r^t I^{k-1-t}} \leq k \frac{1}{r^{(k-1)}}$ and $\sum_{t=1}^{d-k} \frac{1}{r^{t-1} I^{d-k-t}} \leq \frac{d-k}{r^{d-k-1}}$. Then the time complexity for the Tucker format is asymptotically $O(max\{M, N\}\ r + r^d)$.

A standard fully connected layer is presented by Eq.(3.5). Replacing the product with the presented low-rank tensorized product we have the low-rank tensorized fully connected layer as follows:

$$\mathcal{Y} = \sigma(\mathcal{F}_{\Psi(\mathcal{W})}(\mathcal{X}) + \mathcal{B}), \tag{B.5}$$

where $\mathcal{B}$ is the tensorized bias and $\mathcal{Y}$ is the tensorized output. If $\Psi(\mathcal{W})$ refers to the TT or Tucker factors. The number of MACs of the fully connected layer is equal to that of the low-rank product presented in Eq.(B.1) and Eq.(B.2) for TT and Tucker, respectively.

In a convolutional layer, a filter traverses the input, computing the product with the covered input entries at each step. This operation is performed iteratively across the entire input. Every product produces one entry of the output of the layer after passing

through the activation function. The number of output pixels depends on the stride and the step size of the layer. Let M be the size of the filter (including the dimensions and the covered input channels). Then the input can be represented as a vector, $\boldsymbol{x}$, of size M. Let N be the number of filters. Then the weight matrix $\boldsymbol{W}$ is an M by N matrix which is constant while the convolution window is moving over the input. Every output pixel is the product of $\boldsymbol{x}$ by $\boldsymbol{W}$. For example for a 2D convolutional layer, every pixel of the output can be calculated as below:

$$\boldsymbol{y}_{ij} = \sigma(\boldsymbol{W}^T \boldsymbol{x}_{ij} + \boldsymbol{b}) \ \forall \ i, j \ , \tag{B.6}$$

where $\boldsymbol{y}_{ij}$ is a vector of length N representing the output's channels and $\boldsymbol{x}_{ij}$ represent the flatten input window associated with the output's pixel ij. Similar to the fully connected layer, the inputs and biases are tensorized, and the weight tensor is represented with its low-rank factors as follows:

$$\mathcal{Y}_{ij} = \sigma(\mathcal{F}_{\Psi(\mathcal{W})}(\mathcal{X}_{ij}) + \mathcal{B}) \ \forall \ i, j. \tag{B.7}$$

The total number of MACs of the convoltuional layer is equal to $p \times T$ where p is the number of output's pixels and T is the MAC count of the low-rank product presented in Eq.(B.1) and Eq.(B.2). This concept can also be extended to 3D convolution.

Bibliography

[1] A. Nikov, D. Podoprikhin, A. Osokin, and D. Vetrov, *Tensorizing neural networks*, *arXiv:1509.06569* (2015).

[2] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, *Speeding-up convolutional neural networks using fine-tuned cp-decomposition*, *arXiv:1412.6553* (2015).

[3] Y. D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, *Compression of deep convolutional neural networks for fast and low power mobile applications*, *arXiv:1511.06530* (2015).

[4] M. Gabor and R. Zduneck, *Compressing convolutional neural networks with hierarchical tucker-2 decomposition*, *Applied Soft Computing* **132** (2023).

[5] Z. He, S. Gao, L. Xiao, D. Liu, H. He, and D. Barber, *Wider and deeper, cheaper and faster: Tensorized lstms for sequence learning*, *arXiv:1711.01577* (2017).

[6] Y. Yang, D. Krompass, and V. Tresp, *Tensor-train recurrent neural networks for video classification*, in *International Conference on Machine Learning*, pp. 3891–3900, PMLR, 2017.

[7] C. Hawkins, X. Liu, and Z. Zhang, *Towards compact neural networks via end-to-end training: A Bayesian tensor approach with automatic rank determination*, *arXiv preprint arXiv:2010.08689* (2020).

[8] C. Yin, B. Acun, C.-J. Wu, and X. Liu, *TT-rec: Tensor train compression for deep learning recommendation models*, *Proceedings of Machine Learning and Systems* **3** (2021) 448–462.

[9] Z. Yang, S. Choudhary, S. Kunzmann, and Z. Zhang, *Quantization-aware and tensor-compressed training of transformers for natural language understanding*, *arXiv preprint arXiv:2306.01076* (2023).

[10] C. Hawkins and Z. Zhang, *Bayesian tensorized neural networks with automatic rank selection*, *Neurocomputing* **453** (2021) 172–180.

[11] K. Zhang, C. Hawkins, X. Zhang, C. Hao, and Z. Zhang, *On-FPGA training with ultra memory reduction: A low-precision tensor method*, *arXiv preprint arXiv:2104.03420* (2021).

[12] Z. He and Z. Zhang, *High-dimensional uncertainty quantification via tensor regression with rank determination and adaptive sampling*, *IEEE Transactions on Components, Packaging and Manufacturing Technology* **11** (2021), no. 9 1317–1328.

[13] M. Ghadiri, M. Fahrbach, G. Fu, and V. Mirrokni, *Approximately optimal core shapes for tensor decompositions*, *arXiv:2302.03886v* (2023).

[14] C. Mu, B. Huang, J. Wright, and D. Goldfarb, *Square deal: Lower bounds and improved relaxations for tensor recovery*, *arXiv:1307.5870v2* (2013).

[15] Z. Zhong, F. Wei, Z. Lin, and C. Zhang, *Ada-tucker: Compressing deep neural networks via adaptive dimension adjustment tucker decomposition*, *arXiv:1906.07671* (2019).

[16] R. Solgi, Z. He, W. J. Linag, Z. Zhang, and H. A. Loaiciga, *Tensor shape search for efficient compression of tensorized data and neural networks*, *Applied Soft Computing* **149** (2023).

[17] J. Jang and U. Kang, *Fast and memory-efficient tucker decomposition for answering diverse time range queries*, *ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2021* (2021).

[18] S. Zhou, N. X. Vinh, J. Bailey, Y. Jia, and I. Davidson, *Accelerating online cp decompositions for higher order tensors*, *ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2016* (2016).

[19] T. G. Kolda and B. W. Bader, *A fast learning algorithm for deep belief nets*, *SIAM review* **51(3)** (2009) 455–500.

[20] Z. Zhang, X. Yang, I. V. Oseledets, G. E. Karniadakis, and L. Daniel, *Enabling high-dimensional hierarchical uncertainty quantification by ANOVA and tensor-train decomposition*, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems* **34(1)** (2015) 63–76.

[21] Z. Zhang, W. T. Weng, and L. Daniel, *Big-data tensor recovery for high-dimensional uncertainty quantification of process variations*, *IEEE Transactions on Components, Packaging and Manufacturing Technology* **7(5)** (2017) 687–697.

[22] Z. Zhang, K. Batselier, H. Liu, L. Daniel, and N. Wong, *Tensor computation: a new framework for high-dimensional problems in EDA*, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems* **36(4)** (2017) 521–536.

[23] C. Dai, X. Liu, Z. Li, and C. Mu-Yen, *A tucker decomposition based knowledge distillation for intelligent edge applications*, *Applied Soft Computing* **101** (2021).

[24] D. Peddireddy, V. Bansal, and V. Aggarwal, *Classical simulation of variational quantum classifiers using tensor rings*, *Applied Soft Computing* **141** (2023).

[25] R. Bro, *Parafac. tutorial and applications*, *Intelligent Laboratory Systems* **38(2)** (1997) 149–171.

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

[27] V. Oseledets, *Tensor train decomposition*, *SIAM journal on Scientific Computation (SISC)* **33(5)** (2011) 2295–2317.

[28] C. Li and Z. Sun, *Evolutionary topology search for tensor network decomposition*, *Proc. International Conference on Machine Learning* **119** (2020) 5947–5957.

[29] Q. Zhao, L. Zhang, and A. Cichocki, *Bayesian CP factorization of incomplete tensors with automatic rank determination*, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **37(9)** (2015) 1751–1763.

[30] M. Mørup, *Applications of tensor (multiway array) factorizations and decompositions in data mining*, *WIRES Data Mining and Knowledge Discovery* **1** (2011) 24–40.

[31] T. G. Kolda and J. Sun, *Scalable tensor decompositions for multi-aspect data mining*, *IEEE International Conference on Data Mining (ICDM)* (2008) 363–372.

[32] E. Sobhani, P. Comon, and M. Babaie-Zadeh, *Data mining with tensor decompositions*, *GRETSI 2019 - XXVIIème Colloque francophone de traitement du signal et des images* (2019).

[33] J. Fang, *Tightly integrated genomic and epigenomic data mining using tensor decomposition*, *Bioinformatics* **35** (2019) 112–118.

[34] P. Rai, Y. Wang, S. Guo, G. Chen, D. Dunson, and L. Carin, *Scalable Bayesian low-rank decomposition of incomplete multiway tensors*, *Proceedings of the 31st International Conference on Machine Learning* **32(2)** (2014) 1800–1808.

[35] Q. Zhao, L. Zhang, and A. Cichocki, *Bayesian sparse Tucker models for dimension reduction and tensor completion*, *arXiv:1505.02343* (2015).

[36] R. Dian, S. Li, and L. Fang, *Learning a low tensor-train rank representation for hyperspectral image super-resolution*, *IEEE transactions on neural networks and learning systems* **30** (2019), no. 9 2672–2683.

[37] Z. He, B. Zhao, and Z. Zhang, *Active sampling for accelerated mri with low-rank tensors*, in *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pp. 3024–3028, IEEE, 2022.

[38] C. Ibrahim, D. Lykov, Z. He, Y. Alexeev, and I. Safro, *Constructing optimal contraction trees for tensor network quantum circuit simulation*, in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8, IEEE, 2022.

[39] J. Biamonte and V. Bergholm, *Tensor networks in a nutshell*, *arXiv preprint arXiv:1708.00006* (2017).

[40] J. Dborin, F. Barratt, V. Wimalaweera, L. Wright, and A. Green, *Matrix product state pre-training for quantum machine learning*, *Quantum Science and Technology* (2022).

[41] M. B. Soley, P. Bergold, A. A. Gorodetsky, and V. S. Batista, *Functional tensor-train chebyshev method for multidimensional quantum dynamics simulations*, *Journal of Chemical Theory and Computation* **18** (2021), no. 1 25–36.

[42] J. Li, Y. Sun, J. Su, T. Suzuki, and F. Huang, *Understanding generalization in deep learning via tensor methods*, *International Conference on Artificial Intelligence and Statistics* (2020) 504–515.

[43] W. Wang, Y. E. B. Sun, and W. W., *Wide compression: tensor ring nets*, *arXiv:1802.09052* (2018).

[44] H. Chen, F. Ahmad, S. Vorobyov, and F. Porikli, *Tensor decompositions in wireless communications and mimo radar*, *IEEE Journal of Selected Topics in Signal Processing* **15** (2021), no. 3 438–453.

[45] J. Su, J. Li, X. Liu, T. Ranadive, C. Coley, T.-C. Tuan, and F. Huang, *Compact neural architecture designs by tensor representations*, *Frontiers in artificial intelligence* **5** (2022).

[46] A. Obukhov, M. Rakhuba, A. Liniger, Z. Huang, S. Georgoulis, D. Dai, and L. Van Gool, *Spectral tensor train parameterization of deep learning layers*, in *International Conference on Artificial Intelligence and Statistics*, pp. 3547–3555, PMLR, 2021.

[47] R. Solgi, H. A. Loaiciga, and Z. Zhang, *Evolutionary tensor train decomposition for hyper-spectral remote sensing images*, *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium* (2022).

[48] R. Solgi and H. A. Loaiciga, *Bee-inspired metaheuristics for global optimization: a performance comparison*, *Artificial Intelligence Review* (2021).

[49] J. H. Holland, *Adaptations in natural and artificial systems*, *University of Michigan Press, Ann Arbor, MI* (1975).

[50] G. Acampora, A. Chiatto, and A. Vitiello, *Genetic algorithms as classical optimizer for the quantum approximate optimization algorithm*, *Applied Soft Computing* **142** (2023).

[51] M. Wang, A. A. Heidari, and H. Chen, *A multi-objective evolutionary algorithm with decomposition and the information feedback for high-dimensional medical data*, *Applied Soft Computing* **136** (2023).

[52] C. Xing, W. Gong, and S. Li, *Adaptive archive-based multifactorial evolutionary algorithm for constrained multitasking optimization*, *Applied Soft Computing* **143** (2023).

[53] M. Solgi, O. Bozorg-Haddad, and H. A. Loaiciga, *The enhanced honey-bee mating optimization algorithm for water resources optimization*, *Water Resources Management* **31** (2016) 885—-901.

[54] O. Bozorg-Haddad, M. Solgi, and H. A. Loaiciga, *Meta-heuristic and Evolutionary Algorithms for Engineering Optimization*. Wiley, 2017.

[55] J. Huang, W. Sun, and L. Huang, *Deep neural networks compression learning based on multiobjective evolutionary algorithms*, *Neurocomputing* **378** (2020) 260–269.

[56] A. Marzullo, C. Stamile, G. Terracina, F. Calimeri, and S. Van Huffel, *A tensor-based mutation operator for neuroevolution of augmenting topologies (NEAT)*, *2017 IEEE Congress on Evolutionary Computation (CEC)* (2017) 681–687.

[57] Q. Wang, L. Zhang, S. Wei, and B. Li, *Tensor decomposition-based alternate sub-population evolution for large-scale many-objective optimization*, *Information Sciences* **569** (2021) 376–399.

[58] S. Laura, C. Prissette, S. Maire, and N. Thirion-Moreau, *A parallel strategy for an evolutionary stochastic algorithm: application to the CP decomposition of nonnegative n-th order tensors*, *28th European Signal Processing Conference (EUSIPCO)* (2021) 1956–1960.

[59] J. Hastad, *Tensor rank is np-complete.*, *Journal of Algorithms* **11(4)** (1990) 644–654.

[60] R. R. Sharapov and A. V. Lapshin, *Convergence of genetic algorithms*, *Pattern Recognition and Image Analysis* **16** (2006) 392–397.

[61] T. Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and A. Dollar, *Microsoft COCO: common objects in context*, *arXiv:1405.0312* (2015).

[62] L. Deng, *The mnist database of handwritten digit images for machine learning research*, *IEEE Signal Processing Magazine* **29(6)** (2012) 141–142.

[63] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, *arXiv:14091556* (2014).

[64] M. e. a. Naumov, *Deep learning recommendation model for personalization and recommendation systems*, *arXiv:1906.00091* (2019).

[65] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, *arXiv:1512.03385* (2015).

[66] T. Sipola, J. Alatalo, T. Kokkonen, and M. Rantonen, *Artificial intelligence in the iot era: A review of edge ai hardware and software*, in *2022 31st Conference of Open Innovations Association (FRUCT)*, pp. 320–331, 2022.

[67] S. Liu, D. S. Ha, F. Shen, and Y. Yi, *Efficient neural networks for edge devices*, *Computers Electrical Engineering* **92** (2021).

[68] R. P.P., *A review on tinyml: State-of-the-art and prospects*, *Journal of King Saud University – Computer and Information Sciences* **34** (2022) 1595–1623.

[69] H. Han and J. Seibert, *Tinyml: A systematic review and synthesis of existing research*, in *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, IEEE, 2022.

[70] M. Akhoon, S. Suandi, A. Alshahrani, M. A. H. Y. Saad, F. R. Albogamy, M. Z. B. Abdullah, and S. A. Loan, *High performance accelrators for deep neural networks: A review*, *Expert Systems* **39(1)** (2021).

[71] D. Yang, W. Yu, H. Mu, and G. Yao, *Dynamic programming assisted quantization approaches for compressing normal and robust dnn models*, in *in Proc. Asia and South Pacific Design Automation Conference*, pp. 351–357, 2021.

[72] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, *Incremental network quantization: Towards lossless cnns with low-precision weights*, *arXiv:1702.03044* (2017).

[73] S. Han, J. Pool, J. Tran, and W. J. Dally, *Learning both weights and connections for efficient neural networks*, in *NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems*, vol. 1, pp. 1135–1143, 2015.

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

[75] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, *Structuredbayesianpruningvialog-normal multiplicative noise*, in *in NIPS*, pp. 6775–6784, 2017.

[76] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, *arXiv:1503.02531* (2015).

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

[78] Z. Li, *Efficient computation of the tucker decomposition and moment tensor*, *Wake Forest University Graduate School of Arts and Sciences* (2022).

[79] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergan, *A new truncation strategy for the higher-order singular value decomposition.*, *SIAM Journal on Sceintific Computing* **34(2)** (2012).

[80] S. Sadov, *Minimization of the sum under product constraints*, *arXiv:2012.15517* (2020).

[81] D. Kozen, *Depth-First and Breadth-First Search*. Springer, New York, NY, 1992.

[82] A. Krizhevsky, *Learning multiple layers of features from tiny images*, *Technical Report* (2009).