Title

Tensor shape search for efficient compression of tensorized data and neural networks

Authors

Solgi, Ryan

He, Zichang

Liang, William Jiahua

et al.

Publication Date

2023-12-01

DOI

10.1016/j.asoc.2023.110987

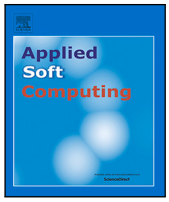
Peer reviewed

Applied Soft Computing Journal 149 (2023) 110987

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc



Tensor shape search for efficient compression of tensorized data and neural networks

Ryan Solgi [a,b,*], Zichang He [a], William Jiahua Liang [c], Zheng Zhang [a], Hugo A. Loaiciga [b]

[a] Department of Electrical and Computer Engineering, University of California Santa Barbara, Santa Barbara, CA, USA
[b] Department of Geography, University of California Santa Barbara, Santa Barbara, CA, USA
[c] School of Engineering, Applied Science, University of Pennsylvania, Philadelphia, PA, USA

ARTICLE INFO

Keywords:
Data compression
Tensor train decomposition
Tensor compression
Genetic algorithm
Tensorized neural networks

ABSTRACT

Compressing big data and model parameters via tensor decomposition such as the tensor train (TT) format has gained great success in recent years. The application of tensor compression methods requires the data be high dimensional. However, not all the real-world data primarily are high-dimensional, and sometimes reshaping is necessary before the application of tensor compression methods. Meantime, reordering and reshaping data may affect the efficiency of the compression. This work utilizes tensor reshaping to improve the efficiency of tensor compression using the TT format. An optimization model is proposed that maximizes the space-saving of tensor compression with respect to the shape of a given tensor while the compression error is bounded. The study is narrowed down to the TT decomposition and the TT-SVD algorithm is linked with a genetic algorithm (GA) to find an optimal tensor shape. The proposed method is applied to compress RGB images and a neural network to exemplify its capability. The results of the proposed tensor shape search using the GA are also compared with a purely random search. The results demonstrate that the proposed tensor shape search method significantly improves the space-saving and compression ratio of the data compression and enhances the efficiency of tensorized neural networks using the TT decomposition.

1. Introduction

Processing high-dimensional data is a necessity across various disciplines. The tensor decomposition methods have been proposed for high dimensional data analysis [1,2]. 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 [3–9]. Different methods have been successfully applied to decompose a higher-order tensor to low-dimensional parameters, including the CANDECOMP/PARAFAC (CP) decomposition [10], the Tucker decomposition [11], and the tensor train (TT) decomposition [12]. Tensor decomposition was also extended to a more general form called tensor networks, which leads to various decomposition formats [13]. 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 [14–16].

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 reshaped to be presented as a tensor

* Corresponding author at: Department of Electrical and Computer Engineering, University of California Santa Barbara, Santa Barbara, CA, USA.
E-mail addresses: Solgi@ucsb.edu (R. Solgi), zichanghe@ucsb.edu (Z. He), wjhliang@seas.upenn.edu (W.J. Liang), zhengzhang@ece.ucsb.edu (Z. Zhang), hugo@geog.ucsb.edu (H.A. Loaiciga).

https://doi.org/10.1016/j.asoc.2023.110987
Received 16 June 2023; Received in revised form 17 October 2023; Accepted 26 October 2023
Available online 30 October 2023

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 shape for tensor decomposition, studies on this domain remain very sparse. This empirical study attempts to answer the aforementioned question.

This study investigates the effect of the tensor shape on tensor compression and proposes an optimization model that maximizes the space saving with respect to the shape 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. Related works

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 [3]. Applications of tensor decomposition for data mining were reviewed in previous studies [17]. Memory efficient Tucker (MET) decomposition was proposed for data mining of sparse multi-way data [18]. Tensor decomposition was used for text mining [19]. A tensor decomposition-based machine learning approach was developed and applied for health data mining [20]. Furthermore, tensor decomposition and representation have been applied for uncertainty quantification [4–6,21], and high dimensional data recovery [14, 22,23] and imaging [24,25], quantum simulation and computation simulation [26–29], to name a few.

Tensor decomposition has been recently shown to be promising for model parameters compression in machine learning [30]. For instance, tensor decomposition has been applied for running compressed convolutional neural networks (CNNs) on mobile devices [31]. 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 [31]. 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 [32]. Compression of fully connected neural networks using tensor decomposition was studied by [33]. In another study, tensor decomposition was applied to study the generalizability of neural networks [34]. Tensor ring network was proposed by [35] in which neural networks were compressed using the Tensor Ring decomposition.

Among the tensor decomposition formats, tensor train [12] 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 [36], hyperspectral imaging [24], neural architecture search [37], deep learning model compression [38–40], quantum dynamics simulation [29], and quantum computation simulation [27, 28].

2.2. Tensor decomposition and hyperparameter tuning

In many real-world applications deciding about some hyperparameters (such as tensor ranks and tensor shapes) of tensor decomposition can be challenging. There have been some recent studies that addressed the tensor rank determination problem. The recent works [15,16] determined the tensor ranks automatically in neural network training, enabling on-device training of neural networks with limited computing resources [7]. A tensor regression method was proposed for automatic rank determination and applied for uncertainty quantification [21]. Bayesian tensor decomposition was also applied to automatic rank determination for tensor completion and dimension reduction [14,23]. However, the study of the effect of the shape on tensor decomposition has rarely been reported in the literature. In a previous study [41] 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.

2.3. 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) [42]. The early version of the genetic algorithm (GA) was presented by [43]. 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 [44–47]. A wide range of evolutionary algorithms, including GAs and their applications in engineering domains, have been studied in the literature [48]. Particularly, [49] applied an evolutionary algorithm to find optimal hyperparameters of the singular value decomposition for the neural network compression.

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 [50]. The CP decomposition was applied to reduce the dimensionality of solutions to solve high-dimensional optimization problems with evolutionary algorithms [51]. A study formulated the CP decomposition of non-negative tensors as a stochastic problem and solved it using an evolutionary algorithm [52]. Also, an evolutionary search was applied to determine an optimum tensor network topology [13]. To the best of these authors' knowledge the present study is the first endeavor that applies an evolutionary tensor shape search with a tensor decomposition to optimize data and neural networks compression.

3. Background

Throughout this manuscript 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. $\mathcal{A}[i_1, \ldots, i_d]$ refers to the element $i_1, \ldots, i_d$ of the tensor $\mathcal{A}$.

3.1. Tensor shape 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

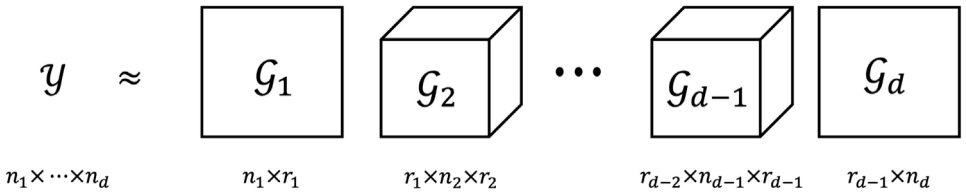


Fig. 1. A schematic of the TT format.

shape of a tensor determines the order and the number of elements of each dimension. Throughout the manuscript, $\theta = (I_1, I_2, \dots, I_k)$ specifies the shape of a tensor, where $I_j \in \mathbb{N}$ is the size of dimension j and k 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 $\mathcal{Y} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$. Cardinality of tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_k}$ is defined as $|\mathcal{X}| = I_1 \times \cdots \times I_k$. Reshaping a tensor may change its cardinality if the cardinality of $\mathcal{Y}$, $|\mathcal{Y}|$, is greater than that of $\mathcal{X}$ ($|\mathcal{Y}| > |\mathcal{X}|$), then dummy elements (e.g., zeros) are entered.

Throughout the manuscript two different functions are applied for reshaping: (1) $\text{reshape}(\mathcal{X}, \theta)$ is used when reshaping does not change the cardinality, and (2) $\Phi(\mathcal{X}, \theta)$ is used to denote reshaping a given tensor $\mathcal{X}$ to a new shape θ if reshaping may change the cardinality. 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 cardinality is larger than that of the original tensor.

3.2. Tensor train (TT) decomposition

In the tensor train (TT) format [12] a d-way tensor $\mathcal{Y} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is approximated with a set of d cores $\bar{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_d\}$ where $\mathcal{G}_j \in \mathbb{R}^{r_{j-1} \times n_j \times r_j}$, r_j's for $j = 1, \dots, d-1$ are the ranks, $r_0 = r_d = 1$, and each element of $\mathcal{Y}$ is approximated by Eq. (1).

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

Fig. 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.

Algorithm 1 TT-SVD

Require: d-way tensor $\mathcal{Y}$, error bound ϵ.
1: $\sigma = \frac{\epsilon}{d-1} \|\mathcal{Y}\|_F$
2: $r_0 = 1$
3: $r_d = 1$
4: $\mathbf{W} = \text{reshape}(\mathcal{Y}, (n_1, \frac{|\mathcal{Y}|}{n_1}))$
5: **for** $j = 1$ to $j = d - 1$ **do**
6: $\quad \mathbf{W} = \text{reshape}(\mathbf{W}, (r_{j-1} n_j, \frac{|\mathbf{W}|}{r_{j-1} n_j}))$
7: $\quad$ Compute σ-truncated SVD: $\mathbf{W} = \mathbf{U}\mathbf{S}\mathbf{V}^T + \mathbf{E}$, where $\|\mathbf{E}\|_F \leq \sigma$
8: $\quad r_j =$ the rank of matrix $\mathbf{W}$ based on σ-truncated SVD
9: $\quad \mathcal{G}_j = \text{reshape}(\mathbf{U}, (r_{j-1}, n_j, r_j))$
10: $\quad \mathbf{W} = \mathbf{S}\mathbf{V}^T$
11: **end for**
12: $\mathcal{G}_d = \text{reshape}(\mathbf{W}, (r_{d-1}, n_d, r_d))$
13: Return $\bar{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_d\}$

This work applies the proposed tensor shape search to the TT-SVD. However, it is possible to extend this framework to other tensor decomposition methods such as the CP decomposition, the Tucker decomposition, and generally to the tensor networks.

4. Problem statement

The current study proposes a search algorithm to find a shape 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 $\mathcal{Y} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ in the explicit original format requires $\prod_{j=1}^{d} n_j$ elements to be stored. However, the TT format requires $\sum_{j=1}^{d} r_{j-1} \times n_j \times r_j$ parameters to be stored. We can use the TT factors as an estimation of the original tensor by applying Eq. (1). 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 [33]. Therefore, the application of the TT format on 1D (i.e., vectors) and 2D (i.e., matrices) arrays requires reshaping 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 shape 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 $\mathcal{Y}_\theta \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ as shown below.

$$\mathcal{Y}_\theta = \Phi(\mathcal{X}, \theta) \tag{2}$$

where $\theta = (n_1, n_2, \dots, n_d)$ refers to the new shape. Function $\Phi(\mathcal{X}, \theta)$ reshapes the given tensor $\mathcal{X}$ to the new shape θ and enter zero values (dummy elements) if $|\mathcal{Y}_\theta| > |\mathcal{X}|$ to fill the rest of the reshaped tensor. Next, $\mathcal{Y}_\theta$ is approximated using the TT decomposition where $\hat{\mathcal{Y}}_\theta \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is the approximation of $\mathcal{Y}_\theta$ using algorithm 1. Note that there exist a bijection between $\mathcal{X}$ and $\hat{\mathcal{Y}}_\theta$ that allows elements of $\hat{\mathcal{X}}$ to be accessed directly from $\hat{\mathcal{Y}}_\theta$ as shown below:

$$\hat{\mathcal{X}} = \Phi^{-1}(\hat{\mathcal{Y}}_\theta) \tag{3}$$

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 $\theta = (n_1, n_2, \dots, n_d)$, space-saving of the TT format using the shape θ is defined as shown below.

$$C(\theta) = 1 - \frac{\sum_{j=1}^{d} r_{j-1} \times n_j \times r_j}{\prod_{j=1}^{k} I_j} \tag{4}$$

where n_j refers to the size of dimension j of reshaped tensor $\hat{\mathcal{Y}}_\theta \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and r_j refers to the ranks of TT decomposition of reshaped

tensor $\mathcal{Y}_\theta$. 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.

$$R(\theta) = (1 - C(\theta))^{-1} \tag{5}$$

where $R(\theta)$ refers to the compression ratio of the shape θ.

5. Methodology: Tensor shape optimization

This study proposes a tensor shape 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 $\mathcal{Y}_\theta \in \mathbb{R}^{n_1 \times \cdots \times n_d}$. Instead of $\mathcal{X}$, $\mathcal{Y}_\theta$ is decomposed and its factors are stored. To retrieve $\hat{\mathcal{X}}$, first $\hat{\mathcal{Y}}_\theta$ is reconstructed using factors of $\mathcal{Y}_\theta$ and elements of $\hat{\mathcal{X}}$ can be accessed directly from bijection between $\mathcal{Y}_\theta$ and $\hat{\mathcal{X}}$ by Eq. (3). This work proposes an optimization model to maximize the space saving by the TT decomposition with respect to the tensor shape θ.

Given d (the order of $\mathcal{Y}$), $\theta = (n_1, n_2, \ldots, n_d)$ is a possible shape; and let S be the space made of all possible θ's such that $n_i \in \mathbb{N}$ and $l \le n_i \le u$ for $i = 1, 2, \ldots, d$ 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 given an error bound ϵ as defined below.

$$\max_{\forall \theta \in \Theta} C(\theta) = 1 - \frac{|\bar{\mathcal{G}}_\theta|}{|\mathcal{X}|}$$

subject to

$$\begin{aligned} \bar{\mathcal{G}}_\theta &= f(\mathcal{Y}_\theta, \epsilon) \\ \mathcal{Y}_\theta &= \Phi(\mathcal{X}, \theta) \\ \Theta &= \{\theta | \theta \in S, |\mathcal{Y}_\theta| \ge |\mathcal{X}|\} \\ S &= \{\theta = (n_1, n_2, \ldots, n_d) | n_i \in \mathbb{N}, l \le n_i \le u\} \end{aligned} \tag{6}$$

where $\Theta \subset S$ and the sub-space Θ refers to the feasible domain of the decision space S. $f(\mathcal{Y}_\theta, \epsilon)$ generates the factors of $\mathcal{Y}_\theta$, $\bar{\mathcal{G}}_\theta$, using the TT-SVD algorithm based on the error bound ϵ. $\Phi(\mathcal{X}, \theta)$ resizes the given tensor $\mathcal{X}$ to the shape θ and enter zero values (dummy elements) if $|\mathcal{Y}_\theta| > |\mathcal{X}|$ to fill the rest of the reshaped tensor. The upper limit of the $C(\theta)$ is 1. When $0 < C(\theta) < 1$ the cardinality of the factors is less than that of the data, but when $C(\theta) \le 0$ the memory requirement is inflated, and there is no data compression.

Any shape that results in a tensor $\mathcal{Y}_\theta$ whose cardinality is smaller than the cardinality of the original given data $\mathcal{X}$ ($|\mathcal{Y}_\theta| < |\mathcal{X}|$) is infeasible because some data is missed. Furthermore, the resized tensor is filled with dummy elements (e.g., zeros) when a possible θ results in a tensor whose cardinality is greater than that of the original data. Any shape which results in an unnecessarily large cardinality is undesirable because it makes the compression less efficient. The objective function defined in Eq. (6) maximizes the space saving considering the effect of the added dummy elements. Therefore, the objective function guides the search toward a shape whose cardinality is the closest to that of the data. The definition of the feasible subspace Θ prevents shapes that have a cardinality smaller than that of the original tensor.

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

$$E(\theta) = \frac{\|\mathcal{X} - \hat{\mathcal{X}}\|_F}{\|\mathcal{X}\|_F}, \text{ with } \hat{\mathcal{X}} = \Phi^{-1}(\hat{\mathcal{Y}}_\theta) \text{ and } \hat{\mathcal{Y}}_\theta = \Psi(\bar{\mathcal{G}}_\theta) \tag{7}$$

where $\Phi(\cdot)^{-1}$ resizes the tensor to the original shape and removes dummy elements if there are any, $\Psi(\cdot)$ generates the approximation tensor $\hat{\mathcal{Y}}_\theta$ from the factors, and $\bar{\mathcal{G}}_\theta$ refers to the decomposed factors. Since the added dummy elements are zero, then $\|\mathcal{X}\|_F = \|\mathcal{Y}\|_F$ and $\|\mathcal{X} - \hat{\mathcal{X}}\|_F \le \|\mathcal{Y} - \hat{\mathcal{Y}}\|_F$. Also, the TT-SVD guarantees that $\frac{\|\mathcal{Y} - \hat{\mathcal{Y}}\|_F}{\|\mathcal{Y}\|_F} \le \epsilon$. Therefore, if the TT-SVD (described in Algorithm 1) is applied for the decomposition of the reshaped tensor, $E(\theta) \le \epsilon$ and it is not required to consider the error bound as a constraint in the optimization model.

6. Genetic algorithm for tensor shape search

The proposed optimization model (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 [53]. 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.

6.1. Initialization

The GA starts with generating a set of random shapes (solutions) $\mathcal{I} = \{\theta_1, \theta_2, \ldots, \theta_m\}$ as an initial population. The initial population is generated by applying a discrete uniform distribution [specifically, $\mathbf{unif}(l, u)$] on each variable $(n_i, i = 1, 2, \ldots, d)$ of $\theta_j = (n_1, n_2, \ldots, n_d)$ for $j = 1, 2, \ldots, m$. Next for each shape θ_j, the TT-SVD is called, and the space saving $C(\theta_j)$ is calculated by Eq. (6).

6.2. Selection

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

$$\Pi(\theta_j) = \frac{C(\theta_j)}{\sum_{j=1}^{m} C(\theta_j)}, \; j = 1, 2, \ldots, m \tag{8}$$

where $\Pi(\theta_j)$ is the selection probability of shape θ_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.

6.3. Reproduction

During the reproduction process the crossover operator is applied first. Based on the crossover operator two shapes like $\theta = (n_1, \ldots, n_d)$ and $\theta' = (n'_1, \ldots, n'_d)$ 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.

$$\theta^{\text{new}} = (n_1, \ldots, n_c, n'_{c+1}, \ldots, n'_d) \tag{9}$$

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 $\theta = (n_1, \ldots, n_i, \ldots, n_d)$ is a newly generated shape by the crossover, the muted shape is $\theta^{\text{new}} = (n_1, \ldots, n''_i, \ldots, n_d)$ 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.

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 [48]. [54] presented the stochastic convergence of the elitist GA.

Algorithm 2 The genetic algorithm for the tensor shape 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

7. Random shape 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 shape search algorithm

Require: T, d, l, $|\mathcal{X}|$
  $C(\boldsymbol{\theta}_r) = 0$
  for $t = 1$ to T **do**
    $u = \lceil \frac{|\mathcal{X}|}{l^{d-1}} \rceil$
    for $j = 1$ to $d - 1$ **do**
      $n_j = RandInit(l, u)$
      $u = \lceil \frac{l \times u}{n_j} \rceil$
    end for
    $n_d = \lceil \frac{|\mathcal{X}|}{\prod_{j=1}^{d-1} n_j} \rceil$
    $\theta = (n_1, n_2, \cdots, n_d)$
    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 cardinality of the data, $|\mathcal{X}|$, where a possible shape is defined as $\theta = (n_1, n_2, \ldots, n_d)$, there are $d-1$ degrees of freedom, and the last dimension is determined such that the cardinality of the randomly generated shape is immediately greater than or equal to the cardinality 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.

Table 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

8. Experimental results

The proposed tensor shape search using the TT-SVD algorithm is applied to decompose some arbitrary RGB images from the Microsoft common objects in context (COCO) data set [55] depicted in Fig. 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 also shows the original shape (height, width, depth) of the data arrays of the images below them.

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. 3 shows the convergence curve of the GA runs for the studied images with $\epsilon = 0.1$. Tables 1–3 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 1–3, $\boldsymbol{\theta}^*$ refers to the optimal shape found by the GA, and $\boldsymbol{\theta}_o$ refers to the original shape of the images.

It is seen in Tables 1–3 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

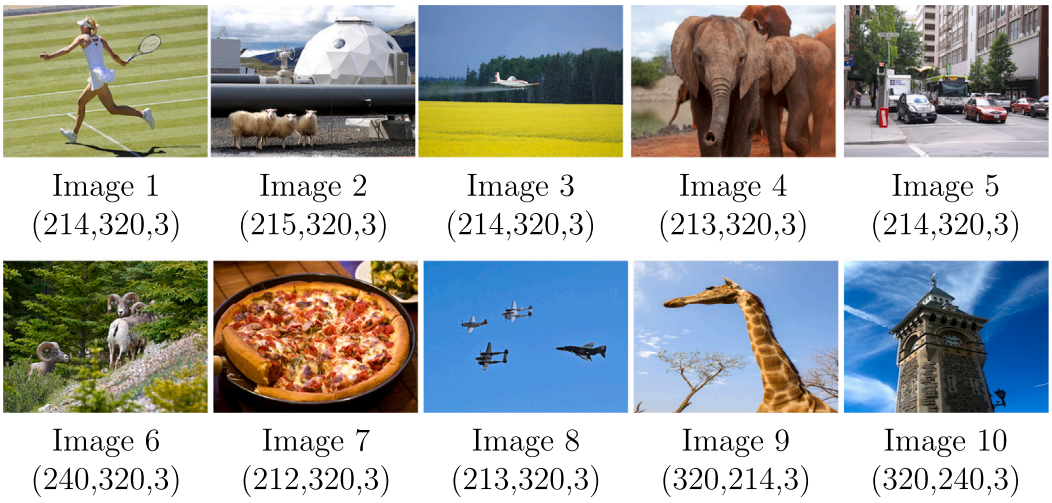


Fig. 2. The arbitrary selected images form the COCO data set (the images are not depicted to their correct scale and the numbers written in parenthesis (height, width, depth) refer to the original shape, θ_o, of the image's data array).

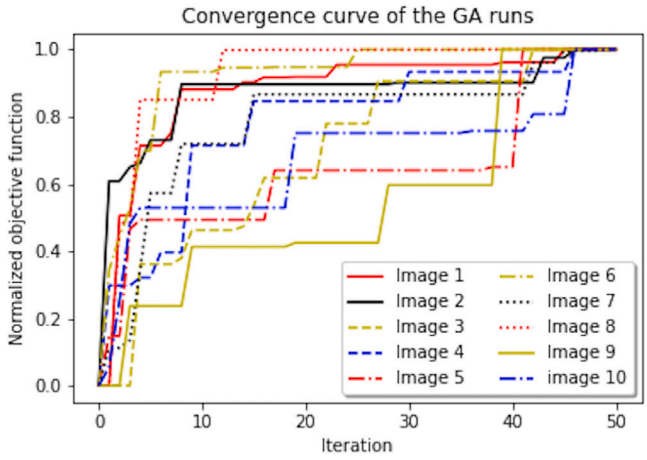


Fig. 3. The convergence curve of the GA runs.

Table 2
The result of the compression of the studied images with their original shape (θ_o) and the optimal shape (θ^*) for $\epsilon = 0.1$.

Image	$C(\theta_o)$%	$E(\theta_o)$	Optimal shape (θ^*)	$C(\theta^*)$%	$E(\theta^*)$
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

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 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

Table 3
The result of the compression of the studied images with their original shape (θ_o) and the optimal shape (θ^*) for $\epsilon = 0.2$.

Image	$C(\theta_o)$%	$E(\theta_o)$	Optimal shape (θ^*)	$C(\theta^*)$%	$E(\theta^*)$
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

(referring to the difference between columns 2 and 5 of Tables 1–3). We can conclude that the compression results of the optimal shapes were significantly improved in comparison with that of the original shapes.

Table 4 lists the ratio between the compression ratio of the optimal shape, $R(\theta^*)$, and the compression ratio of the original shape, $R(\theta_o)$ for different error bounds from $\epsilon = 0.01$ up to $\epsilon = 0.2$. In other words, Table 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, $R(\theta)$, is defined in Eq. (5). In Table 4, it is seen that by increasing the error bound, on average, $R(\theta^*)/R(\theta_o)$ increases while the variance also increases. This is visualized in Fig. 4, which depicts the minimum, mean, maximum, and variance of $R(\theta^*)/R(\theta_o)$ for all images versus the error bound. According to Table 4 and Fig. 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 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 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

Table 4
$R(\theta^*)/R(\theta_o)$ 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

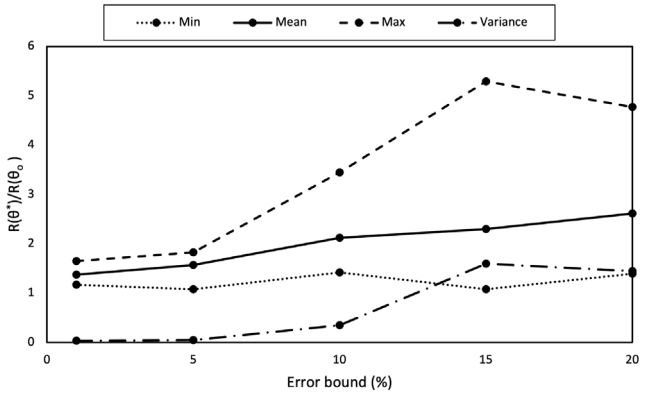


Fig. 4. Comparison of $R(\theta^*)/R(\theta_o)$ for different error bounds including $\epsilon = 0.01$, $\epsilon = 0.05$, $\epsilon = 0.10$, $\epsilon = 0.15$, and $\epsilon = 0.20$.

Table 5
The space saving of the best shapes found by the random search, $C(\theta_r)\%$, 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

Table 6
Comparing the results of the GA and the RS including the differences in space savings (%) and the ratio between compression ratios.

Image	$C(\theta^*) - C(\theta_r)\%$			$R(\theta^*)/R(\theta_r)$		
	$\epsilon = 0.05$	$\epsilon = 0.10$	$\epsilon = 0.20$	$\epsilon = 0.05$	$\epsilon = 0.10$	$\epsilon = 0.20$
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

shown in Fig. 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.

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. (1). 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. Fig. 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 Fig. 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 1–3, yet, the actual error of the optimal shape was mostly higher than that of the original shape. This difference is visible in Fig. 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.

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 1000, 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 5. Table 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 6 it is seen that on average the GA improved the space saving by 2.66%, 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 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 s, respectively. The average wall time of the RS for error bounds 0.05, 0.1, and 0.2 were about 30, 23, and 17 s, respectively. The compression efficiency of the GA was higher even though the RS was slightly faster.

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

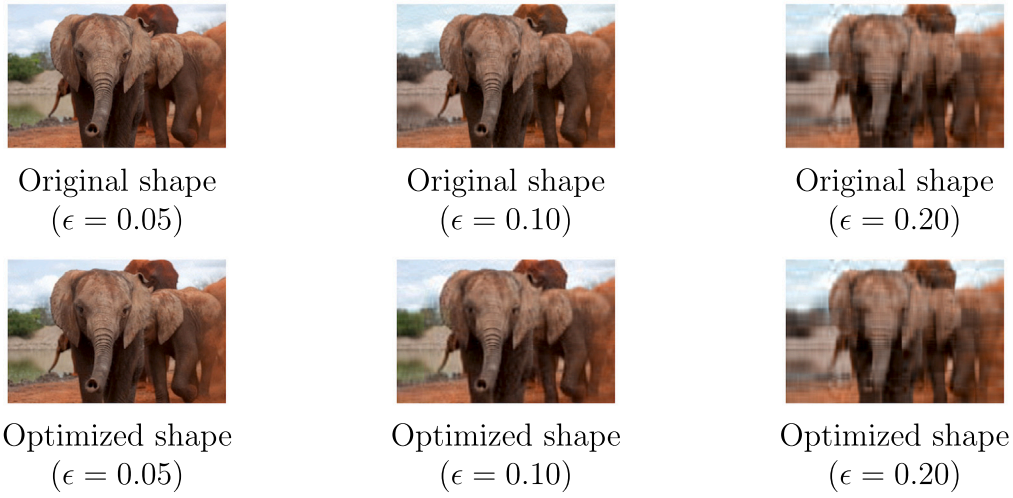


Fig. 5. Restoration of image 4 from the compressed data using the optimal shape found by the GA and the original shape.

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 [56]. 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 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. 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, 1000 randomly trial solutions were examined.

Table 7 lists the results for the network accuracy before and after compression. Table 8 lists the optimum shape found by the GA and RS for compressing the first dense layer. It is seen in Table 7 that using the proposed shape search by the GA the network can be compressed up to about 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 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 8.

The weights of the first layer are compressed for which the maximum, average, and minimum space saving of 1000 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 1000 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 1000 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 7 and 8 correspond to the largest space savings.

Table 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	401,408
GA ($\epsilon = 1.0$)	95.92	88.62	1353 (297×)
RS ($\epsilon = 1.0$)	95.41	88.10	4425 (91×)
GA ($\epsilon = 0.8$)	98.13	89.30	36,170 (11×)
RS ($\epsilon = 0.8$)	97.65	88.60	97,916 (4×)

Table 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)

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 s, respectively. On the same processor, the wall time of the RS for error bounds 1 and 0.8 were about 31 and 43 s, 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 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.

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 shape 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 shape 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 shape 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 shape 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 limits the application of tensor shape 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 including the Tucker and CP decomposition, and improving the efficiency of the optimization algorithm are the subjects of future studies.

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 shape 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 shape 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.

CRediT authorship contribution statement

Ryan Solgi: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing. **Zichang He:** Conceptualization, Writing – original draft, Writing – review & editing. **William Jiahua Liang:** Visualization. **Zheng Zhang:** Conceptualization, Writing – review & editing. **Hugo A. Loaiciga:** Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Zheng Zhang, Zichang He reports was provided by National Science Foundation.

Data availability

This study applied the publicly available COCO [55] and MNIST [56] data sets.

Acknowledgments

This work is partially supported by NSF, USA Grants CCF-1817037 and CCF-1763699, and the Department of Geography, University of California Santa Barbara, USA.

References

[1] J. Jang, U. Kang, Fast and memory-efficient tucker decomposition for answering diverse time range queries, in: ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2021, 2021.
[2] S. Zhou, N.X. Vinh, J. Bailey, Y. Jia, I. Davidson, Accelerating online CP decompositions for higher order tensors, in: ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2016, 2016.
[3] T.G. Kolda, B.W. Bader, A fast learning algorithm for deep belief nets, SIAM Rev. 51 (3) (2009) 455–500.
[4] Z. Zhang, X. Yang, I.V. Oseledets, G.E. Karniadakis, L. Daniel, Enabling high-dimensional hierarchical uncertainty quantification by ANOVA and tensor-train decomposition, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 34 (1) (2015) 63–76.
[5] Z. Zhang, W.T. Weng, L. Daniel, Big-data tensor recovery for high-dimensional uncertainty quantification of process variations, IEEE Trans. Compon. Packag. Manuf. Technol. 7 (5) (2017) 687–697.
[6] Z. Zhang, K. Batselier, H. Liu, L. Daniel, N. Wong, Tensor computation: a new framework for high-dimensional problems in EDA, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 36 (4) (2017) 521–536.
[7] K. Zhang, C. Hawkins, X. Zhang, C. Hao, Z. Zhang, On-FPGA training with ultra memory reduction: A low-precision tensor method, 2021, arXiv preprint arXiv:2104.03420.
[8] C. Dai, X. Liu, Z. Li, C. Mu-Yen, A tucker decomposition based knowledge distillation for intelligent edge applications, Appl. Soft Comput. 101 (2021).
[9] D. Peddireddy, V. Bansal, V. Aggarwal, Classical simulation of variational quantum classifiers using tensor rings, Appl. Soft Comput. 141 (2023).
[10] R. Bro, Parafac. Tutorial and applications, Intell. Lab. Syst. 38 (2) (1997) 149–171.
[11] L.R. Tucker, Some mathematical notes on three-mode factor analysis, Psychometrika 31 (3) (1966) 279–311.
[12] V. Oseledets, Tensor train decomposition, SIAM J. Sci. Comput. (SISC) 33 (5) (2011) 2295–2317.
[13] C. Li, Z. Sun, Evolutionary topology search for tensor network decomposition, in: Proc. International Conference on Machine Learning, Vol. 119, 2020, pp. 5947–5957.
[14] Q. Zhao, L. Zhang, A. Cichocki, Bayesian CP factorization of incomplete tensors with automatic rank determination, IEEE Trans. Pattern Anal. Mach. Intell. 37 (9) (2015) 1751–1763.
[15] C. Hawkins, X. Liu, Z. Zhang, Towards compact neural networks via end-to-end training: A Bayesian tensor approach with automatic rank determination, 2020, arXiv preprint arXiv:2010.08689.
[16] C. Hawkins, Z. Zhang, Bayesian tensorized neural networks with automatic rank selection, Neurocomputing 453 (2021) 172–180.
[17] M. Mørup, Applications of tensor (multiway array) factorizations and decompositions in data mining, WIRES Data Min. Knowl. Discov. 1 (2011) 24–40.
[18] T.G. Kolda, J. Sun, Scalable tensor decompositions for multi-aspect data mining, in: IEEE International Conference on Data Mining (ICDM), 2008, pp. 363–372.
[19] E. Sobhani, P. Comon, M. Babaie-Zadeh, Data mining with tensor decompositions, in: GRETSI 2019 - XXVIIème Colloque Francophone De Traitement Du Signal Et Des Images, 2019.
[20] J. Fang, Tightly integrated genomic and epigenomic data mining using tensor decomposition, Bioinformatics 35 (2019) 112–118.
[21] Z. He, Z. Zhang, High-dimensional uncertainty quantification via tensor regression with rank determination and adaptive sampling, IEEE Trans. Compon. Packag. Manuf. Technol. 11 (9) (2021) 1317–1328.

[22] P. Rai, Y. Wang, S. Guo, G. Chen, D. Dunson, L. Carin, Scalable Bayesian low-rank decomposition of incomplete multiway tensors, in: Proceedings of the 31st International Conference on Machine Learning, Vol. 32, 2014, pp. 1800–1808, (2).
[23] Q. Zhao, L. Zhang, A. Cichocki, Bayesian sparse Tucker models for dimension reduction and tensor completion, 2015, arXiv:1505.02343.
[24] R. Dian, S. Li, L. Fang, Learning a low tensor-train rank representation for hyperspectral image super-resolution, IEEE Trans. Neural Netw. Learn. Syst. 30 (9) (2019) 2672–2683.
[25] Z. He, B. Zhao, 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), IEEE, 2022, pp. 3024–3028.
[26] C. Ibrahim, D. Lykov, Z. He, Y. Alexeev, I. Safro, Constructing optimal contraction trees for tensor network quantum circuit simulation, in: 2022 IEEE High Performance Extreme Computing Conference (HPEC), IEEE, 2022, pp. 1–8.
[27] J. Biamonte, V. Bergholm, Tensor networks in a nutshell, 2017, arXiv preprint arXiv:1708.00006.
[28] J. Dborin, F. Barratt, V. Wimalaweera, L. Wright, A. Green, Matrix product state pre-training for quantum machine learning, Quantum Sci. Technol. (2022).
[29] M.B. Soley, P. Bergold, A.A. Gorodetsky, V.S. Batista, Functional tensor-train Chebyshev method for multidimensional quantum dynamics simulations, J. Chem. Theory Comput. 18 (1) (2021) 25–36.
[30] M. Gabor, R. Zduneck, Compressing convolutional neural networks with hierarchical tucker-2 decomposition, Appl. Soft Comput. 132 (2023).
[31] Y.D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, D. Shin, Compression of deep convolutional neural networks for fast and low power mobile applications, 2015, arXiv:1511.06530.
[32] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, V. Lempitsky, Speeding-up convolutional neural networks using fine-tuned CP-decomposition, 2015, arXiv:1412.6553.
[33] A. Nikov, D. Podoprikhin, A. Osokin, D. Vetrov, Tensorizing neural networks, 2015, arXiv:1509.06569.
[34] J. Li, Y. Sun, J. Su, T. Suzuki, F. Huang, Understanding generalization in deep learning via tensor methods, in: International Conference on Artificial Intelligence and Statistics, 2020, pp. 504–515.
[35] W. Wang, Y.E.B. Sun, W. W., Wide compression: tensor ring nets, 2018, arXiv:1802.09052.
[36] H. Chen, F. Ahmad, S. Vorobyov, F. Porikli, Tensor decompositions in wireless communications and MIMO radar, IEEE J. Sel. Top. Sign. Proces. 15 (3) (2021) 438–453.
[37] J. Su, J. Li, X. Liu, T. Ranadive, C. Coley, T.-C. Tuan, F. Huang, Compact neural architecture designs by tensor representations, Front. Artif. Intell. 5 (2022).
[38] C. Yin, B. Acun, C.-J. Wu, X. Liu, TT-rec: Tensor train compression for deep learning recommendation models, Proc. Mach. Learn. Syst. 3 (2021) 448–462.
[39] Y. Yang, D. Krompass, V. Tresp, Tensor-train recurrent neural networks for video classification, in: International Conference on Machine Learning, PMLR, 2017, pp. 3891–3900.
[40] A. Obukhov, M. Rakhuba, A. Liniger, Z. Huang, S. Georgoulis, D. Dai, L. Van Gool, Spectral tensor train parameterization of deep learning layers, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2021, pp. 3547–3555.
[41] R. Solgi, H.A. Loaiciga, Z. Zhang, Evolutionary tensor train decomposition for hyper-spectral remote sensing images, in: IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium, 2022.
[42] R. Solgi, H.A. Loaiciga, Bee-inspired metaheuristics for global optimization: a performance comparison, Artif. Intell. Rev. (2021).
[43] J.H. Holland, Adaptations in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.
[44] G. Acampora, A. Chiatto, A. Vitiello, Genetic algorithms as classical optimizer for the quantum approximate optimization algorithm, Appl. Soft Comput. 142 (2023).
[45] M. Wang, A.A. Heidari, H. Chen, A multi-objective evolutionary algorithm with decomposition and the information feedback for high-dimensional medical data, Appl. Soft Comput. 136 (2023).
[46] C. Xing, W. Gong, S. Li, Adaptive archive-based multifactorial evolutionary algorithm for constrained multitasking optimization, Appl. Soft Comput. 143 (2023).
[47] M. Solgi, O. Bozorg-Haddad, H.A. Loaiciga, The enhanced honey-bee mating optimization algorithm for water resources optimization, Water Resour. Manag. 31 (2016) 885—901.
[48] O. Bozorg-Haddad, M. Solgi, H.A. Loaiciga, Meta-Heuristic and Evolutionary Algorithms for Engineering Optimization, Wiley, 2017.
[49] J. Huang, W. Sun, L. Huang, Deep neural networks compression learning based on multiobjective evolutionary algorithms, Neurocomputing 378 (2020) 260–269.
[50] A. Marzullo, C. Stamile, G. Terracina, F. Calimeri, S. Van Huffel, A tensor-based mutation operator for neuroevolution of augmenting topologies (NEAT), in: 2017 IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 681–687.
[51] Q. Wang, L. Zhang, S. Wei, B. Li, Tensor decomposition-based alternate sub-population evolution for large-scale many-objective optimization, Inform. Sci. 569 (2021) 376–399.
[52] S. Laura, C. Prissette, S. Maire, N. Thirion-Moreau, A parallel strategy for an evolutionary stochastic algorithm: application to the CP decomposition of nonnegative N-th order tensors, in: 28th European Signal Processing Conference (EUSIPCO), 2021, pp. 1956–1960.
[53] J. Hastad, Tensor rank is NP-complete, J. Algorithms 11 (4) (1990) 644–654.
[54] R.R. Sharapov, A.V. Lapshin, Convergence of genetic algorithms, Pattern Recognit. Image Anal. 16 (2006) 392–397.
[55] T.Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C.L. Zitnick, A. Dollar, Microsoft COCO: common objects in context, 2015, arXiv:1405.0312.
[56] L. Deng, The mnist database of handwritten digit images for machine learning research, IEEE Signal Process. Mag. 29 (6) (2012) 141–142.