applied sciences

MDPI

Article

Learning and Compressing: Low-Rank Matrix Factorization for Deep Neural Network Compression

Gaoyuan Cai [1,2], Juhu Li [1,2,*], Xuanxin Liu [1,2], Zhibo Chen [1,2] and Haiyan Zhang [1,2]

[1] School of Information Science and Technology, Beijing Forestry University, Beijing 100083, China
[2] Engineering Research Center for Forestry-Oriented Intelligent Information Processing of National Forestry and Grassland Administration, Beijing 100083, China
* Correspondence: lijuhu@bjfu.edu.cn

Abstract: Recently, the deep neural network (DNN) has become one of the most advanced and powerful methods used in classification tasks. However, the cost of DNN models is sometimes considerable due to the huge sets of parameters. Therefore, it is necessary to compress these models in order to reduce the parameters in weight matrices and decrease computational consumption, while maintaining the same level of accuracy. In this paper, in order to deal with the compression problem, we first combine the loss function and the compression cost function into a joint function, and optimize it as an optimization framework. Then we combine the CUR decomposition method with this joint optimization framework to obtain the low-rank approximation matrices. Finally, we narrow the gap between the weight matrices and the low-rank approximations to compress the DNN models on the image classification task. In this algorithm, we not only solve the optimal ranks by enumeration, but also obtain the compression result with low-rank characteristics iteratively. Experiments were carried out on three public datasets under classification tasks. Comparisons with baselines and current state-of-the-art results can conclude that our proposed low-rank joint optimization compression algorithm can achieve higher accuracy and compression ratios.

Keywords: deep neural network compression; low-rank matrix factorization; truncated singular value decomposition; CUR decomposition; joint optimization; optimal rank

Citation: Cai, G.; Li, J.; Liu, X.; Chen, Z.; Zhang, H. Learning and Compressing: Low-Rank Matrix Factorization for Deep Neural Network Compression. *Appl. Sci.* **2023**, *13*, 2704. https://doi.org/10.3390/app13042704

Academic Editors: Phivos Mylonas, Katia Lida Kermanidis and Manolis Maragoudakis

Received: 5 January 2023
Revised: 15 February 2023
Accepted: 16 February 2023
Published: 20 February 2023

1. Introduction

In recent years, DNNs have become prevalent in the machine learning area, applied in various fields such as computer vision (CV) [1,2], natural language processing (NLP) [3,4], and speech recognition [5,6]. However, with the increase in the accuracy performance of real-world applications, the DNN model needs more neurons, thus the increasing number of layers in DNN induce massive weight parameters, which makes the calculation resources large. Therefore, it is significant to compress the DNNs while keeping their accuracy performance when running these compressed models on resource-constrained embedded devices.

There exist many DNN compressing techniques so far, such as pruning [7–10], weight quantization [11–14], and low-rank matrix factorization (MF) [15–18].

Weight pruning is proposed to reduce the weight parameters of the DNN models while retaining original precision, including removing neurons and automatically learning the correct number of neurons and weights. However, all the pruning criteria require the manual setup of sensitivity for each layer and require the fine-tuning of the weight parameters. Weight quantization intends to compress the DNN model by re-representing the number of bits required to represent each weight parameter. In fact, the main difficulty of the weight quantization is that processing low-precision weights and excessive quantization will make the parameters of DNN model lose important content and information. The low-rank matrix factorization is to find an approximate matrix with low-rank property

to replace the origin, so as to reduce the number of matrix parameters. The objective function for the low-rank MF is to minimize the Frobenius norm between the original matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ and the low-rank approximation matrix $\mathbf{UV}$ of the following form:

$$\min_{\mathbf{U},\mathbf{V}} \|\mathbf{W} - \mathbf{UV}\|_{\mathrm{F}}^2, \tag{1}$$

where $\mathbf{U} \in \mathbb{R}^{m \times k}$ and $\mathbf{V} \in \mathbb{R}^{k \times n}$. In theory, since the weight matrices or tensors in each layer of DNN models have high dimensions, we can first decompose them into lower dimension (two-dimensional) matrices and then adopt parameter reduction. The low-rank MF supports not only trained-from-scratch models but also pretrained models. However, the implementation of the technique is generally hard because it involves very complex decomposition operations in terms of computation. In addition, the compressed model using low-rank MF needs to be re-trained and fine-tuned multiple times to finally reach excellent performance and convergence.

In this paper, we propose a DNN compression algorithm based on the low-rank MF method, which incorporates the MF method into a joint optimization function and attains the optimal rank–weight matrix in each convolutional layer with optimal rank self-selection. In the experiments on three public datasets, we use the truncated singular value decomposition (TSVD) and CUR decomposition as the low-rank matrix factorization method in this joint optimization, and conduct the comprehensive comparisons and analysis of the experimental results using the two above methods. Furthermore, the algorithm compresses to varying degrees across multiple DNN models under classification tasks, and outperforms other state-of-the-art experimental results. Specifically, the main contributions of this algorithm are as follows:

- In this work, the proposed low-rank compression algorithm combines the loss function and the compression cost function into a joint function. We optimize the joint function using the augmented Lagrangian method (ALM), separating it into two independent steps of training (learning) and compression, which can achieve the optimized solution iteratively.
- From the compression level, we take the rank as the variable and express it as a linear compression cost function related to memory and computation. In addition, we add a prescribed variable parameter to control the interaction between the compression cost function and the loss function. Consequently, the proposed function can automatically search for the optimal ranks under different compression levels during the compression process.
- From the generality level, we combine the CUR decomposition and TSVD techniques with the joint optimization framework, respectively, to carry out experiments. The experimental results using joint optimization of above two MF methods show that our algorithm is very general and can be easily applied for model compression using other low-rank MFs.

The rest of the paper is organized as follows: Section 2 presents an overview of neural network compression in MF techniques and optimal rank selection methods. Section 3 describes the algorithm in detail, including the problem formulation, complexity analysis, and additional clarifications on how this low-rank compression algorithm is performed. Section 4 first introduces experimental parameters and datasets processing, then presents experimental results and conducts extensive theoretical analysis, and finally evaluates the performance among single classic MF methods and joint optimization framework in combination with different low-rank MF methods. Section 5 summarizes the manuscript, including objectives, key findings, limitations ,and future works.

2. Related Works

2.1. Matrix Factorization Techniques

Non-negative matrix Factorization (NMF) [19] is a fundamental data analysis technique that is applied in many real world systems, such as recommender systems and audio processing systems. The purpose of NMF is to express a non-negative matrix (i.e., a matrix with non-negative entries) using the product of two non=negative matrices. This method is widely used because it is computationally fast and uses nonsubtractive linear combinations to make factorization easily interpretable in many cases. Ref. [20] exploited the random projections of the data structure in NMF to reduce memory demands without any major performance deterioration. Ref. [21] proposed a sparse deep NMF model that satisfied different sparsity requirements, and adopted the Nesterov's accelerated algorithm to accelerate the computing process during optimization. This model has competitive classification accuracy and a more intuitive interpretation of features. Ref. [22] presented an iterative updating algorithm called self-paced learning and adaptive neighbors methodology (SPLNMFAN) which incorporated the self-paced learning regularization and the adaptive graph to improve NMF solution performance. However, it is difficult to accurately restore the original matrix with NMF and it can only be used as a means of approximate restoration.

The singular value decomposition (SVD) [23–25] is among the most ubiquitous and powerful methods for data processing in the computational area. Additionally, SVD performs best in Equation (1), which means it can decompose and restore the original matrix almost without losses. In SVD, matrix can be decomposed into the product of three matrices $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V^T}$. It can be approximated by truncating the largest "k" singular values of the above three matrices, which is called the TSVD. Ref. [26] combined SVD with DNNs; the authors first decomposed each layer into its full-rank form using SVD, then directly performed training on the decomposed full-rank weights. In order to encourage low ranks to avoid gradient vanishing or exploding, they also added the orthogonality regularization to the singular vectors. Ref. [27] proposed sparse low-rank (SLR) factorization, which computed the importance of neurons, and used the lower rank to disperse the SVD matrix for the unimportant neuron, resulting in a better compression ratio.However, it requires more time for fine-tuning and retraining. In the applications of DNNs, it can reduce the number of model parameters through factorization and low-rank approximation. Although the TSVD is applied to many sizable applications, as far as the domain of origin of the factorization data is concerned, its decomposition components still lack meaning and interpretation.

Recently, there has been interest in a low-rank MF called the CUR decomposition [28–30] (also called the pseudo skeleton approximation). The CUR decomposition first selects some rows and columns in the original matrix, and then constructs three low-dimensional matrices using the above-selected rows and columns, and finally calculates the product of above three matrices to approximate the original one. Although the approximations are suboptimal [31], it preserves the original properties and makes selected columns or rows retain their original meaning. Therefore, the CUR decomposition has the natural interpretability, traceability, and explicit physical meaning. In particular, Ref. [32] proposed an efficient CUR decomposition algorithm to decompose and approximate a given matrix. Moreover, it provided error bounds of choosing $\mathbf{C}$ and $\mathbf{R}$ from the given matrix $\mathbf{W}$ based upon the probabilistic method. Ref. [17] gave another CUR decomposition algorithm using a way of columns selection, which made the error bounds $\|\mathbf{W} - \mathbf{CUR}\|_{\mathrm{F}}$ less than or equal to $(2 + \varepsilon)\|\mathbf{W}\|_F$. The bounds mean that the CUR rank r approximation is equivalent to the TSVD of the matrix $\mathbf{W}$ with rank r. Ref. [28] stated that CUR decomposition should be valued in the real-world analysis of low-dimensional data, because the matrices $\mathbf{C}$ and $\mathbf{R}$ preserved the primitive structure of the data. Even though CUR decomposition can provide an effective and interpretable decomposition method, it is impossible to select an appropriate rank automatically. This is exactly what our algorithm solves.

2.2. Optimal Rank Selection

The DNN model consists of multiple layers played with different roles. Low-rank compression for DNN weight tensors requires optimal rank selection of the weight tensors at each layer. An inappropriate rank may lead to a significant drop in DNN performance. Therefore, the appropriate rank is the crucial factor for using matrix factorization to compress neural networks. In view of the above problem, one DNN model parameter-reduction technique [33], inspired by SVD was proposed. It used three factorized matrices instead of weight matrix and applied sparsity constraint to make entries of the center diagonal matrix zero. This method avoided retraining and solved for the optimal rank selection problems of the conventional low-rank MF approaches. Ref. [34] proposed a global optimization technique based on Bayesian optimization (BayesOpt), which could find the global optimal rank. The method disregarded the scale of the dataset and the structural characteristics of the network while making a good trade-off between computational complexity and accuracy. Ref. [35] proposed a method to minimize the complexity of the network while maintaining the required accuracy. The authors defined rank selection as a combinatorial optimization problem and introduced the space-restricted parameters to reduce the search space and obtain optimal ranks. Ref. [36] proposed a novel approach that stabilizes low-rank approximations of convolutional kernels and ensured efficient compression while maintaining the high-quality performance of the neural networks. Ref. [37] combined ranks with the inference runtime and adopted alternating optimization to solve it so that the network ran as fast as possible on the device while having the best performance (e.g., classification accuracy).

The literature shows that existing rank-based DNN compression algorithms have two major drawbacks: First, a general rank-selection algorithm is lacking for general matrix factorization. Second, existing algorithms can not explicitly control the degree of compression. Each compression method may have its own unique accuracy-compression ratio. According to the above two shortcomings, this paper aims to develop a low-rank DNN compression algorithm that can solve the rank selection and freely control the degree of compression.

3. Algorithm

3.1. TSVD Technique

In a numerical solution, TSVD is reformulated as the optimization problem Equation (1) that minimizes the Frobenius norm between the given matrix $\mathbf{W}$ and the low-rank approximated matrix $\mathbf{UV}$. We assume that there is a given rank k matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ which is decomposed into three matrices: the left singular matrices $\mathbf{U} \in \mathbb{R}^{m\times m}$, the diagonal singular matrix $\mathbf{\Sigma}$ and the right singular matrices $\mathbf{V} \in \mathbb{R}^{n\times n}$, and the closed form is as follows:

$$\mathbf{W} = \mathbf{U\Sigma V^T}, \text{s.t.} \mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I_m}, \mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I_n}, \tag{2}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, $\mathbf{\Sigma} = \mathrm{diag}(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, ..., \boldsymbol{\alpha}_n)$ and the diagonal elements $\boldsymbol{\alpha}$ are the singular values ordered as $\boldsymbol{\alpha}_1 > \boldsymbol{\alpha}_2 > ... > \boldsymbol{\alpha}_\mathbf{n} \geq 0$. In many cases, the sum of the top 10% or even 1% of the singular values accounts for more than 99% of the sum of all singular values. In other words, we can approximate the description matrix $\mathbf{W}$ with the largest r singular values where $r \leq k \leq \min(m, n)$. Therefore, the approximated matrix $\mathbf{\Theta}$ can be reconstructed as follows:

$$\mathbf{W} \approx \mathbf{\Theta} = \mathbf{U}(:, 1:r)\mathbf{\Sigma}(1:r, 1:r)\mathbf{V}(:, 1, r)^T \tag{3}$$

For the given matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$, after applying Equation (3), matrices $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ are truncated with low-rank r. We assume that $\mathbf{U} = \mathbf{U}(:, \mathbf{1}:\mathbf{r})$ and $\mathbf{V} = \mathbf{\Sigma}(\mathbf{1}:\mathbf{r}, \mathbf{1}:\mathbf{r})\mathbf{V}(:, \mathbf{1}, \mathbf{r})^{\mathbf{T}}$. Therefore, we can obtain the compression ratio of the TSVD is $\mathrm{O}(\frac{mn}{mr+nr})$. Obviously, $mr + nr$ will be smaller than mn when r is significantly smaller than $\min(m, n)$.

3.2. CUR Decomposition

In this subsection, we will briefly introduce the CUR decomposition method. We summarize the overall process of CUR decomposition in Algorithm 1.

Algorithm 1 Pseudocode: CUR decomposition

input: the weight matrix $\mathbf{W}$, rank parameter r and hyperparameter c

1: **Compute** the SVD of the $\mathbf{W}$: $\mathbf{W} = \mathbf{U\Sigma V}^T$
2: **Compute** the normalized statistical leverage scores with the top r right singular vectors of $\mathbf{W}$: $\pi_j = \frac{1}{r}\sum_{\xi=1}^{r}\left(\mathbf{V}_j^{\xi}\right)^2$
3: **Compute C** by choosing the jth column ($j \in \{1,\ldots,n\}$) of W based on the probability: $p_j = \min\{1, c\pi_j\}$
4: **Compute R** by inputting $\mathbf{W}^T$, r and c, and repeating step 1–3
5: **Compute U** : $\mathbf{U}=\mathbf{C}^{+}\mathbf{W}\mathbf{R}^{+}$
6: **Compute Θ** : $\mathbf{\Theta} = \mathbf{CUR}$
7: **Return Θ**

Assume that $\mathbf{W^j}$ is the j-th column of weight matrix and can be represented as $\mathbf{W}^j = \sum_{\xi=1}^{R}\mathbf{U}_{\xi}\mathbf{\Sigma}^{\xi}\mathbf{V}_j^{\xi}$ using the SVD method (Equation (2)), where $R = rank(\mathbf{W})$ and $\mathbf{V}_j^{\xi}$ is the j-th coordinate of the ξ-th right singular vector. Thus, $\mathbf{W^j}$ can be approximate as a linear combination of the truncated top r left singular vectors and corresponding singular values, denoted as $\mathbf{W}^j \approx \sum_{\xi=1}^{r}\mathbf{U}_{\xi}\mathbf{\Sigma}^{\xi}\mathbf{V}_j^{\xi}$. In the calculation of the normalized statistical leverage scores, it is obvious to prove that $\pi_j \geq 0$ and that $\sum_{j=1}^{n}\pi_j = 1$ for all $j \in \{1,\ldots,n\}$, and thus that these scores form a probability distribution with n columns. In conclusion, we calculate the CUR decomposition in three steps. First, we choose columns from the given $m \times n$ matrix $\mathbf{W}$ and calculate the normalized statistical leverage scores using the probabilities for all $j \in \{1,\ldots,n\}$ with $c = O(\frac{r\log r}{\varepsilon^2})$. Secondly, following this procedure (Algorithm 1), we select the rows and columns of matrix $\mathbf{W}$ to construct the matrices $\mathbf{C}$ and $\mathbf{R}$, respectively. Finally, we compute the matrix $\mathbf{U} = \mathbf{C}^{+}\mathbf{W}\mathbf{R}^{+}$, where $\mathbf{C}^{+}$ and $\mathbf{R}^{+}$ is the Moore–Penrose generalized inverse of $\mathbf{C}$ and $\mathbf{R}$, respectively. Additionally, the runtime of this CUR decomposition algorithm is dominated by the column and row leverage scores' calculation, and the error analysis of this decomposition method proves that:

$$\|\mathbf{W} - \mathbf{CUR}\|_{\mathrm{F}} \leq (2+\varepsilon)\|\mathbf{W}\|_F, \tag{4}$$

which is presented clearly in [32].

In generally, after applying the CUR decomposition algorithm to the matrix $\mathbf{W}$, we can obtain three matrices: $\mathbf{C} \in \mathbb{R}^{m\times c}$, $\mathbf{U} \in \mathbb{R}^{c\times c}$, and $\mathbf{R} \in \mathbb{R}^{c\times n}$. In other words, we will obtain the compressed matrix $\mathbf{\Theta}$ by the product of the above three matrices. Suppose that we have a given weight matrix $\mathbf{W}$ with size of $m \times n$, we use CUR decomposition (Algorithm 1) to decompose it and obtain two dense layers that satisfy $\mathbf{D} = \mathbf{C} \in \mathbb{R}^{m\times c}$ and $\mathbf{E} = \mathbf{UR} \in \mathbb{R}^{c\times n}$. The total number of parameters in these two dense layers $\mathbf{D}$ and $\mathbf{E}$ are $c \times (m+n)$, and the compression ratio of CUR decomposition is $O(\frac{mn}{mc+nc})$. Obviously, $c \times (m+n)$ will be smaller than $m \times n$ when c is significantly smaller than $\min(m,n)$.

3.3. Problem Formulation

Specifically, inspired by the learning-compression algorithm [11,15,38] that formulates neural net compression in a general way via constrained optimization, we first formulate the problem of low-rank compression based on ranks auto-selection as a constrained optimization and then solve it via joint optimization, and finally show how this fits into the DNN compression. Here, we exhibit and extend this approach to joint-optimization func-

tions and iterative enumeration to solve the problem of compressing deep networks using low-rank MF. It assumed that the weight matrix $\mathbf{W}$ consists of the set $\{\mathbf{W}_1, \mathbf{W}_2, ..., \mathbf{W}_k\}$. $\mathbf{W}_k$ has the dimension of $m_k \times n_k$ and the dimension satisfies $m_{k+1} = n_k$ because the output from the previous layer is used as input of the current layer. The objective function is as follows:

$$\min_{\mathbf{W},\mathbf{\Theta},r} L(\mathbf{W}) + \lambda C(r), \text{s.t.} \mathbf{W}_k = \mathbf{\Theta}_k, \text{rank}(\mathbf{\Theta}_k) = r_k \leq R_k, k = 1, ..., K, \tag{5}$$

where $L(\mathbf{W})$ is defined as classification loss, which can be solved by stochastic gradient descent, and $C(r)$ is defined as the compression cost function, λ is a prescribed positive parameter which naturally trades off the classification loss $L(\mathbf{W})$ with the compression cost function $C(r)$ and determines the distribution of ranks over the deep net layers, $\mathbf{\Theta_k}$ is the low-rank approximation matrix of weight matrix $\mathbf{W_k}$ and the rank of $\mathbf{\Theta_k}$ is r_k. Since memory and computation satisfy a linear relationship with rank r_k in k-th layer, the compression cost function $C(r)$ uses the rank r as one of the variables. As mentioned above, function C can be defined by the ranks in each layer of the following form:

$$C(r) = C_1(r_1) + C_2(r_2) + ... + C_k(r_k), \tag{6}$$

where $C_k(r_k) = (m_k + n_k) \times r_k$. From a modeling point of view, $C(r)$ can represent several related compression costs in the context of DNN compression by appropriate choices of the coefficients rank r.

3.4. Optimization Algorithm

In this section, we mainly focus on optimizing DNN model compression based on the low-rank CUR decomposition technique. We first introduce the solution of the objective function of Equation (5), and then give the incorporate process of CUR decomposition and ranks auto-selection. The detailed solution process is shown in Algorithm 2.

Algorithm 2 Pseudocode: optimization framework

input: the weight matrices $\{\mathbf{W}_k\}$ and compression cost functions $C_k(r_k)$ in k-th layer, hyperparameter λ and μ_0

Train the weight matrices of the DNN model: $\mathbf{W} = \{\mathbf{W}_k\} \leftarrow \arg\min_{\mathbf{W}} L(\mathbf{W})$
Initialize ranks and Lagrange multipliers at each layer: $\{r_k\} = 0, \{\mathbf{M}_k\} = 0$
for $\mu = \mu_0 < \mu_1 < ... < \infty$ **do**
  for $k = 1, 2, ..., K$ **do**
    Compute $\mathbf{\Theta}_k$ from $\mathbf{W}_k$ using Algorithm 1
  end for
  Solve the learning step:$\mathbf{W} \leftarrow \underset{\mathbf{W}}{\text{argmin}}\, L(\mathbf{W}) + \frac{\mu}{2}\left\|\mathbf{W} - \mathbf{\Theta} - \frac{1}{\mu}\mathbf{M}\right\|_{\text{F}}^2$
  for $k = 1, 2, ..., K$ **do**
    Solve the compressing step:

$$\mathbf{\Theta}_k, \mathbf{r}_k \leftarrow \underset{\mathbf{\Theta}_k, r_k}{\text{argmin}}\, \lambda C_k(r_k) + \frac{\mu}{2}\left\|\mathbf{W}_k - \mathbf{\Theta}_k - \frac{1}{\mu}\mathbf{M}_k\right\|_{\text{F}}^2$$

  end for
  Update Lagrange multipliers $\mathbf{M} \leftarrow \mathbf{M} - \mu(\mathbf{W} - \mathbf{\Theta})$
end for
Return $\mathbf{W}, \mathbf{\Theta}, r$

In DNNs, we assume that the network has K layers, and W_k represents the weight matrix in the k-th layer ($k \in \{1, \ldots, K\}$). In order to solve Equation (5), we introduce the ALM to convert the constrained optimization into an unconstrained optimization.

Under iterations, we can solve Equation (5)ALM by introducing the Lagrange multiplier vector $\mathbf{M}$. Then the Equation (5) can be rewritten as follows:

$$\begin{aligned} &\min_{W,\Theta,r} L(\mathbf{W}) + \lambda C(r) - \mathbf{M}^{\mathrm{T}} \sum_{k=1}^{K} (\mathbf{W}_k - \mathbf{\Theta}_k) + \tfrac{\mu}{2} \sum_{k=1}^{K} \|\mathbf{W}_k - \mathbf{\Theta}_k\|_{\mathrm{F}}^2, \\ &\text{s.t} \quad \operatorname{rank}(\mathbf{\Theta}_k) = r_k \leq R_k, k = 1, ..., K \end{aligned} \tag{7}$$

where μ is penalty parameter and its value should be non-negative, $\mathbf{M}$ is Lagrange multiplier which has the same dimension as $\mathbf{W}$. For Equation (7), it can be alternately optimized by updating Lagrange multiplier while driving $\mu \to \infty$.

In Algorithm 2, this algorithm has two sets per layer in the k-th layer: $\{\mathbf{W}\}$ in the learning step, and $\{\mathbf{\Theta}\}$ with rank $\{r\}$ determined in each compressing step. We first initialize the weight matrix by directly computing the classification loss and then divide the solution into two separated steps: the learning step and compression step. The former step is a regular learning step (training process) of a cross-entropy loss function plus a quadratic regularization penalty, which drives the matrices $\mathbf{W}_k$ towards the low-rank approximation matrices $\mathbf{\Theta}_k$, and the form is as follows:

$$\min_{\mathbf{W}} L(\mathbf{W}) + \frac{\mu}{2} \left\| \mathbf{W} - \mathbf{\Theta} - \frac{1}{\mu}\mathbf{M} \right\|_F^2, \text{s.t.} \operatorname{rank}(\mathbf{\Theta}_k) = r_k \leq R_k, k = 1, ..., K \tag{8}$$

Based on the learning step, we can use stochastic gradient descent (SGD) to optimize Equation (8) for training on large-scale datasets and high-dimensional weight matrices. The latter step compresses the matrix in each iteration via CUR decomposition, of the following form:

$$\min_{\mathbf{\Theta},r} \lambda \mathrm{C}_k(r_k) + \frac{\mu}{2} \left\| \mathbf{W}_k - \mathbf{\Theta}_k - \frac{1}{\mu}\mathbf{M}_k \right\|_F^2, \text{s.t.} \operatorname{rank}(\mathbf{\Theta}_k) = r_k \leq R_k, k = 1, ..., K \tag{9}$$

In this step, as described in Algorithm 1, we first decompose and approximate the weight matrix, where $\Sigma = \operatorname{diag}(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, ..., \boldsymbol{\alpha}_r)$ and $\boldsymbol{\alpha}_1 > \boldsymbol{\alpha}_2 > ... > \boldsymbol{\alpha}_r$. To better obtain the optimal rank, we use the Eckhart–Young theorem [39] to convert Equation (9) into the closed iterative form so that this equation can be equivalent to:

$$\min_{r_k} \lambda \mathrm{C}_k(r_k) + \frac{\mu}{2} \sum_{i=r_{k+1}}^{R_k} \boldsymbol{\alpha}_i^2, \tag{10}$$

Therefore, according to Equation (6), Equation (10) can find the optimal rank by enumerating all values of r_{k+1} up to R_k. Once we obtain the optimal rank r_k, $\mathbf{\Theta}_k$ is computed using the low-rank CUR decomposition with r_k and $c_k = \mathrm{O}(\frac{r_k \log r_k}{\varepsilon^2})$ in k-th layer.

In general, our algorithm requires multiple iterations, and each iteration consists of the learning step and the compression step. The former step reduces the gap between the weight matrix $\mathbf{W}$ and the low-rank matrix $\mathbf{\Theta}$ by using the SGD method, and the latter step finds the optimal rank r by enumeration, and then the optimal rank r obtained before is used as the input for low-rank CUR decomposition (Algorithm 1) to compress the weight matrix $\mathbf{W}$. Repeat above operations until $\mathbf{W}$ and $\mathbf{\Theta}$ finally coincide.

3.5. Complexity Analysis

Our algorithm consists of two seperate steps: the learning step and compression step, which both coincide in the limit $\mu \to \infty$ through multiple iterations. The former step is the weights training process, which can be solved multiple times by the SGD method. Its time is usually several times that of the original DNN model training time. The latter step is a low-rank approximation of the weight matrices, which can be solved by low-rank MF methods (CUR decomposition and TSVD in this manuscript).

Assume w.l.o.g. $\mathbf{W} \in \mathbb{R}^{m \times n}$. The complexity of the learning step is $I_l \times I_s \times N \times \mathcal{O}(m \times n)$, where I_l denotes the number of iterations for learning step, I_s denotes the number of times the SGD method is used in each iteration, N denotes the size of the public dataset. The compression step is divided into solving the optimal rank by enumeration and matrix low rank approximation. For Equation (10), it finds the optimal rank by enumeration, and the complexity of computing optimal rank r_k is $\mathcal{O}(R^2)$, R denotes the size of the singular value matrix. For TSVD, it is obvious that its complexity is $\mathcal{O}(m^2 \times n + n^2 \times m)$. For the CUR decomposition (Algorithm 1), its process mainly consists of SVD, normalized statistical leverage scores calculation, column selection, and matrix operation. The complexity of the first operation is $\mathcal{O}(m^2 \times n + n^2 \times m)$, of the second operation is $\mathcal{O}(r \times n)$, of the third operation is $\mathcal{O}(c \times (m+n))$, and of the fourth operation is $\mathcal{O}(m \times c^2 + 2 \times m \times c + n \times c^2)$. Therefore, the total complexity of the CUR decomposition is $\mathcal{O}(m^2 \times n + n^2 \times m + r \times n + c \times (m+n) + m \times c^2 + 2 \times m \times c + n \times c^2$. Consequently, the complexity of compression step is $I_l \times (\mathcal{O}(R^2) + \mathcal{O}(m^2 \times n + n^2 \times m + r \times n + c \times (m+n) + m \times c^2 + 2 \times m \times c + n \times c^2))$.

In a word, the computing complexity of this algorithm (the joint optimization of CUR decomposition) is $I_l \times I_s \times N \times \mathcal{O}(r \times n) + I_l \times (\mathcal{O}(R^2) + \mathcal{O}(m^2 \times n + n^2 \times m + r \times n + c \times (m+n) + m \times c^2 + 2 \times m \times c + n \times c^2))$.

4. Experiments

All experiments are evaluated on three public datasets over classification tasks: LeNet (LeNet5 and LeNet300) on Modified National Institute of Standards and Technology (MNIST) dataset [40], Visual Geometry Group Network (VGG16) [41], and (Residual Neural Network (ResNet (20,32,56)) [42] on Canadian Institute for Advanced Research (CIFAR) datasets [43]. The experiments are started from reference nets with the same or exceeding test accuracies reported in the literature [40–42]. We use Python3.7 and NVIDIA GPU (NVIDIA TITAN RTX 24GB) to carry out all experiments, the environment is torch1.1.0.

4.1. Experimental Setup

4.1.1. Datasets and Preprocessing

- MNIST
  MNIST is a large dataset of handwritten images in grayscale format, which is commonly used to train various image-processing systems. This dataset is widely used for training and testing in the field of machine learning. There are 60,000 training images and 10,000 testing images with 28×28 pixels. Half of the training set and half of the test set come from the National Institute of Standards and Technology (NIST) training dataset, while the other halves of the training set and the test set comes from the NIST test dataset.
- CIFAR
  The CIFAR has two datasets, CIFAR-10 and CIFAR100. CIFAR-10 has 60,000 colored natural images of 10 classes with 32×32 pixels, of which the training set contains 50,000 images and the test set contains 10,000 images. The dataset is divided into five training batches and one testing batch, each with 10,000 images. The test batch contains exactly 1000 randomly selected images from each category. Similar to CIFAR-10, CIFAR-100 has 100 classes, and each class contains 600 images (500 training images and 100 testing images).
- Processing
  In this paper, we preprocess the MNIST and CIFAR datasets separately before training the DNN models. For MNIST, We normalize pixels in $[0, 1]$ at first, and then subtract their mean. For CIFAR, in addition to normalization, we randomly flip and crop the datasets to expand them as well.

4.1.2. Low-Rank Parametrization of Convolutional Layers

The convolutional layer is a four-dimensional tensor of size $n \times c \times d \times d$; that is, n filters with c channels of $d \times d$ spatial resolution. In experiments, we decompose the

convolutional layer using a low-rank structure scheme [44,45] into two low-rank matrices with $r \times cd^2$ and $n \times r$, respectively. In networks, they can be viewed as two convolutional layers of r filters with c channels of $d \times d$ spatial resolution and n filters with r of $r \times 1 \times 1$ spatial resolution.

4.1.3. Parameter Introduction

Each iteration in this algorithm consists of a learning step and compression step. The learning step is solved by Nesterov SGD [46]. In each iteration, models are trained for 15 epochs (30 for MNIST) with momentum 0.9 over the batch size of 128 on CIFAR and 256 on MINST under the initial learning rate $\eta_0 = 0.001$, and the decayed learning rate is $\eta_0 a^x$ at the xth epoch, where $a = \{ 0.98, 0.99\}$ is the learning rate decay. The compression step is solved by low-rank MF with a scalar rank selection. It needs to run at least 60 iterations and is controlled by hyperparameter λ and the penalty parameter schedule of $\mu_0 b^y$ at yth iteration, where $b \in \{ 1.2, 1.5\}$ is the iterative expansion coefficient of $\mu_0 = 10^{-3}$.

4.2. Comparison Baselines Introduction

We use two single low-rank matrix compression methods as baselines and briefly describe the experimental procedure. In each compression experiment, a reference network with full-rank weight matrices is trained first, and then a lower rank is chosen for each layer. We first obtain the optimal ranks r (Equation (10)) using our compression algorithm (Algorithm 2) so that the architecture can be determined. Then we conduct experiments of compression using single TSVD (Equations (2) and (3)) and CUR decomposition (Equation (1)) under above optimal rank. Furthermore, to demonstrate our low-rank compression algorithm can be generally applied for model compression using other low-rank MFs, we also perform our compression algorithm based on TSVD as the compression step (replacing CUR decomposition with TSVD). It should be noted that all compression algorithms are applied in convolutional layers of DNNs models only.

Experimental Results

We use the processed MNIST dataset (normalize pixels in $[0, 1]$ at first, and then subtract their mean) to train LeNet300 and LeNet5. The experimental results with different λ in Table 1 clearly show that our algorithm has a slightly decreased or even a lossless classification test error.For the experiments on the MNIST dataset, compared to employing single SVD and the CUR decomposition techniques in LeNet300 models, we can easily recognize that our performance outperforms across the board from Table 1; the test error of our algorithm is lossless and about 1.90%. Additionally, on LeNet5, with a high storage compression ratio of 17.85×, our method has the same error test 0.94% as the original one.

Table 1. Comparison of compression rate and error rate for LeNet on MNIST.

Model	Algorithm	E_{test} (%)	ρ_{FLOPs}	ρ_{storage}
LeNet5	Uncompressed	0.94	1.00	1.00
	TSVD	1.61	3.50	4.12
	CUR	1.60	3.71	4.70
	Ours ($\lambda = 1 \times 10^{-4}$)	**0.94**	**7.02**	**17.85**
LeNet300	Uncompressed	1.90	1.00	1.00
	TSVD	2.62	1.67	1.67
	CUR	2.46	1.67	1.67
	Ours ($\lambda = 1 \times 10^{-4}$)	**1.90**	**1.95**	**1.95**

Similarly, for the experiments on CIFAR-10 dataset (Table 2), we also obtain impressive storage compression ratio and FLOPs compression ratio from using VGG16, ResNet20, ResNet32, and ResNet56 models.

Table 2. Comparison of compression rate and error rate for VGG16 and ResNets on CIFAR-10.

Model	Algorithm	E_{test} (%)	ρ_{FLOPs}	ρ_{storage}
VGG16	Uncompressed	6.75	1.00	1.00
	TSVD	7.93	1.08	1.65
	CUR	7.92	1.30	1.82
	BayesOpt [34]	8.71	-	8.97
	SLR-a [27]	7.41	-	9.17
	Ours ($\lambda = 0.2 \times 10^{-4}$)	**6.15**	**4.52**	**7.37**
	Ours ($\lambda = 0.4 \times 10^{-4}$)	**7.00**	**5.87**	**9.71**
ResNet20	Uncompressed	8.15	1.00	1.00
	TSVD	9.80	1.94	2.35
	CUR	9.77	2.01	2.59
	Standard TR [47,48]	12.50	-	5.40
	CNN-FCF [18]	10.47	3.226	3.17
	Learning LDNN [26]	10.57	3.26	-
	Ours($\lambda = 4 \times 10^{-4}$)	**9.38**	**2.66**	**2.65**
	Ours($\lambda = 16 \times 10^{-4}$)	**10.58**	**4.30**	**4.34**
ResNet32	Uncompressed	7.22	1.00	1.00
	TSVD	9.41	1.63	2.42
	CUR	9.36	1.82	2.60
	Standard TR [47,48]	9.40	-	5.10
	CNN-FCF [18]	9.26	3.35	3.27
	Learning LDNN [26]	9.45	3.93	-
	Ours ($\lambda = 4 \times 10^{-4}$)	**8.54**	**2.96**	**3.05**
	Ours ($\lambda = 8 \times 10^{-4}$)	**9.02**	**3.43**	**3.66**
ResNet56	Uncompressed	6.50	1.00	1.00
	TSVD	8.61	1.51	2.05
	CUR	8.42	1.56	2.30
	CNN-FCF [18]	6.62	1.77	1.76
	CNN-FCF [18]	8.08	3.44	3.30
	Learning LDNN [26]	8.45	3.75	-
	Ours ($\lambda = 4 \times 10^{-4}$)	**7.46**	**3.00**	**3.01**
	Ours ($\lambda = 8 \times 10^{-4}$)	**8.11**	**4.21**	**4.29**

Finally, for the experiments on the CIFAR-100 dataset (Table 3), the compressed ResNet20, ResNet32 and ResNet56 models have achieved moderate storage compression ratios of 2.34×, 3.82×, and 2.66×, respectively. Here, the test error in case of the compressed ResNet20 is lower than the TSVD model, around 5.74%, and most notably, our ResNet20 and ResNet32 networks surpass the uncompressed version by 1.38% and 1.04% in terms of accuracy, respectively.

It is not only that all error tests are excellent, but also the weight parameters are reduced significantly. It is worth noting that the part of VGG16 network of this table shows that its error test (6.15%) is higher than the uncompressed one (6.75%), under high storage–compression ratio (7.37×).

Each iteration contains the compression step and the learning step, and the former performs the low-rank matrix factorization (Algorithm 1) to obtain the low-rank approximation matrix $\boldsymbol{\Theta}$, and the latter performs multiple SGD methods to make the weight matrix $\mathbf{W}$ approximate $\boldsymbol{\Theta}$. The gradient descent curve of the model in the learning step is shown in Figure 1. It is obvious that the overall trend of the curve is declining, which means that the curves have similar convergence characteristics under different λ (convergence is reached sooner or later in different iterations). Therefore, these curves provide an experimental guarantee for the convergence of our algorithm, and indicate that different values of λ hardly affect the convergence of this algorithm. Additionally, in this figure, the numerical jitter of the enlarged part is caused by the learning step in multiple iterations. In other words, each compression step of iteration generates the new low-rank matrix $\boldsymbol{\Theta}$, and the weight matrix

W uses SGD methods to approximate the new **Θ** and calculate many loss values in the learning step. The first loss calculation in each iteration is a high value, and then gradually decreases, which causes this jitter phenomenon. It is worth noting that the learning loss of some models has a tend of increasing first and then decreasing (Figure 1c–f). We will explain this trend below. Our algorithm iteratively finds the optimal solution. For more complex networks, at the beginning, the optimal rank of the weight matrix in the convolutional layer has not been determined (or partially determined). There is no gap between **W** and **Θ**, thus the learning loss is small at this time. After a certain number of iterations, the ranks of each layer are gradually determined, and the low-rank approximation matrix **Θ** is also determined, so that the learning loss of the compressed DNN model increases until the ranks of each layer are all determined, and then gradually decreases.

Table 3. Comparison of compression rate and error rate for VGG16 ResNets on CIFAR-100.

Model	Algorithm	E_{test} (%)	ρ_{FLOPs}	ρ_{storage}
ResNet20	Uncompressed	34.60	1.00	1.00
	TSVD	38.96	2.05	1.96
	CUR	38.20	2.19	2.07
	Standard TR [47,48]	36.45	-	4.00
	Ours ($\lambda = 2 \times 10^{-4}$)	**33.22**	**2.53**	**2.34**
	Ours ($\lambda = 8 \times 10^{-4}$)	**36.08**	**4.17**	**4.00**
ResNet32	Uncompressed	31.90	1.00	1.00
	TSVD	36.47	3.05	2.74
	CUR	35.93	3.44	2.95
	Standard TR [47,48]	33.30	-	4.80
	Ours ($\lambda = 2 \times 10^{-4}$)	**30.86**	**4.08**	**3.82**
	Ours ($\lambda = 8 \times 10^{-4}$)	**33.24**	**5.01**	**4.87**
ResNet56	Uncompressed	30.15	1.00	1.00
	TSVD	35.28	2.20	1.94
	CUR	35.17	3.06	2.50
	Standard TR [47,48]	32.73	-	4.60
	Ours ($\lambda = 4 \times 10^{-4}$)	**31.62**	**3.13**	**2.66**
	Ours ($\lambda = 16 \times 10^{-4}$)	**32.74**	**5.03**	**4.86**

Considering that similar convergence behavior (e.g., Figure 1) generally does not mean that different values of λ lead to similar accuracy, we analyze the performance sensitivity of our algorithm to λ. Ideally, $\mathbf{W}_k$ should exhibit the same low-rank properties as the compressed low-rank matrix $\mathbf{\Theta}_k$, with high accuracy in classification tasks. To examine the desired low-rank behavior of **W**, we record and observe the Frobenius norm, which measures the similarity between **W** and **Θ**, as shown in Figure 2. According to the compression step, **Θ** is constantly updated using low-rank matrix factorization methods. The curves shown in Figure 2 show that **W** exhibits the low-rank features indeed. Figure 3 shows the classification accuracy of the DNN model with different λ values during compression. It can be seen from this figure that the compression process shows a gradual upward trend whether it is a simple or a complex network.

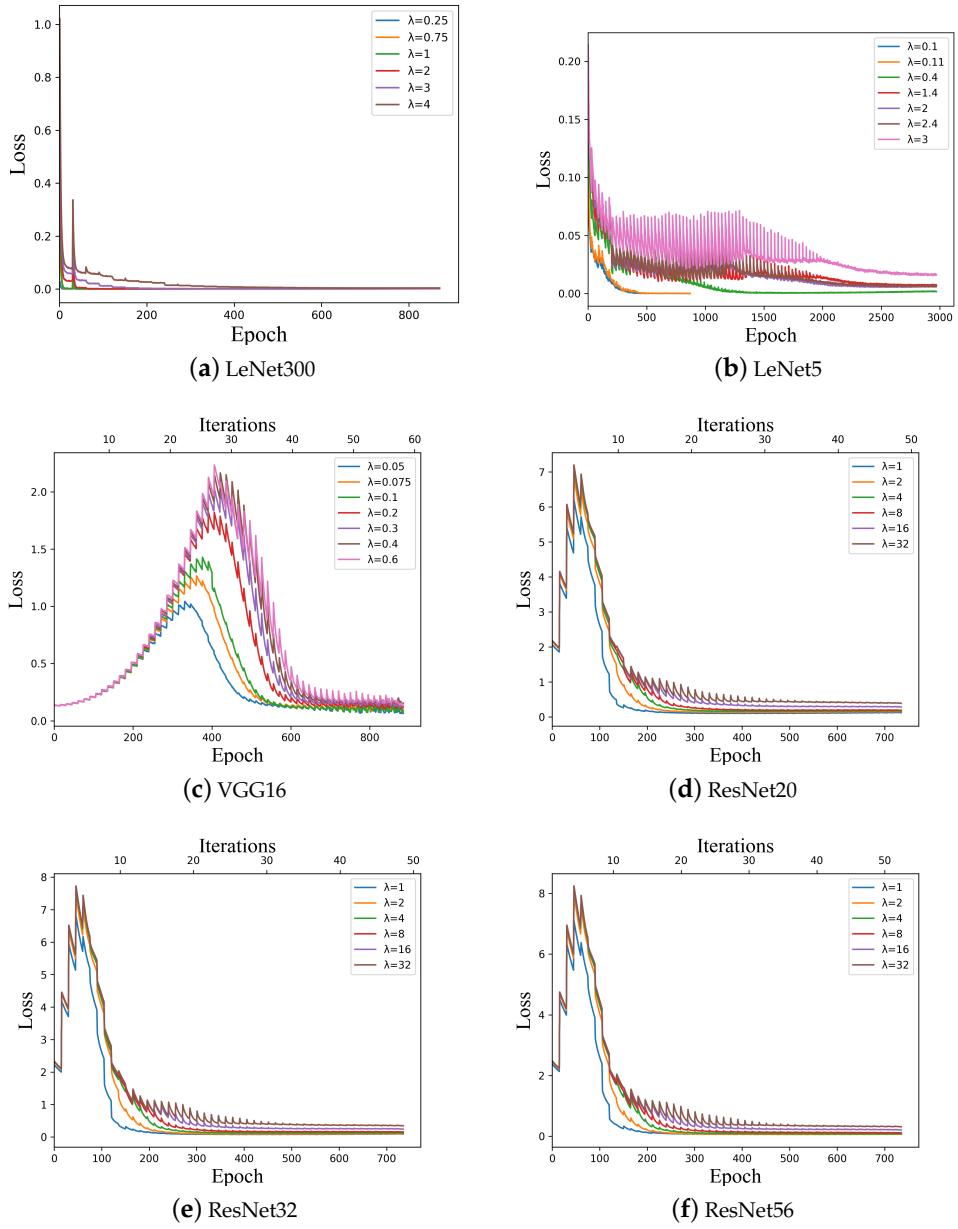


Figure 1. The model Learning Loss with different $\lambda \times 10^{-4}$, for LeNets on MNIST, VGG16 and ResNets on CIFAR-10.

In addition, Figure 3 not only examines the classification accuracy brought by different values of λ, but also measures the difference in classification accuracy brought by different λ values under similar convergence behavior. It can be seen from this figure that the smaller the value of λ, the higher the final classification accuracy, and vice versa. As shown in Figure 2, the gap in the Frobenius norm is relatively large at the beginning. The above situation occurs because the ranks of the weight matrices in the convolutional layers have not been determined at this time, thus the results in Figure 3 are equivalent to random classification. After a period of iterations, the $\mathbf{W}$ and $\mathbf{\Theta}$ in each layer begins to be determined, and the gap in the Frobenius norm gradually narrows, which makes the classification accuracy of the compressed DNN models increase rapidly.

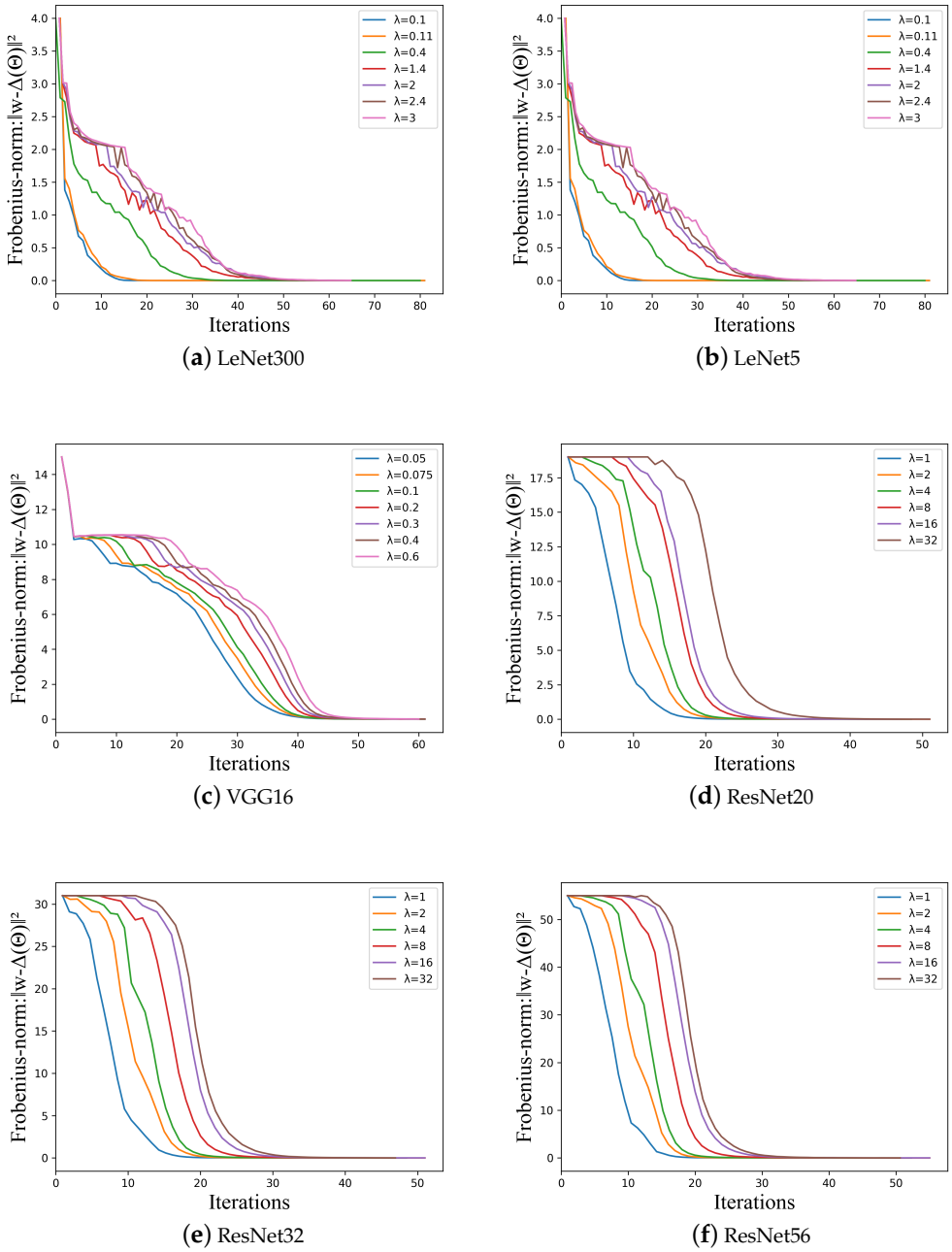


Figure 2. The F-norm between the origin and compression matrices with different $\lambda \times 10^{-4}$, for LeNets on MNIST, VGG16 and ResNets on CIFAR-10.

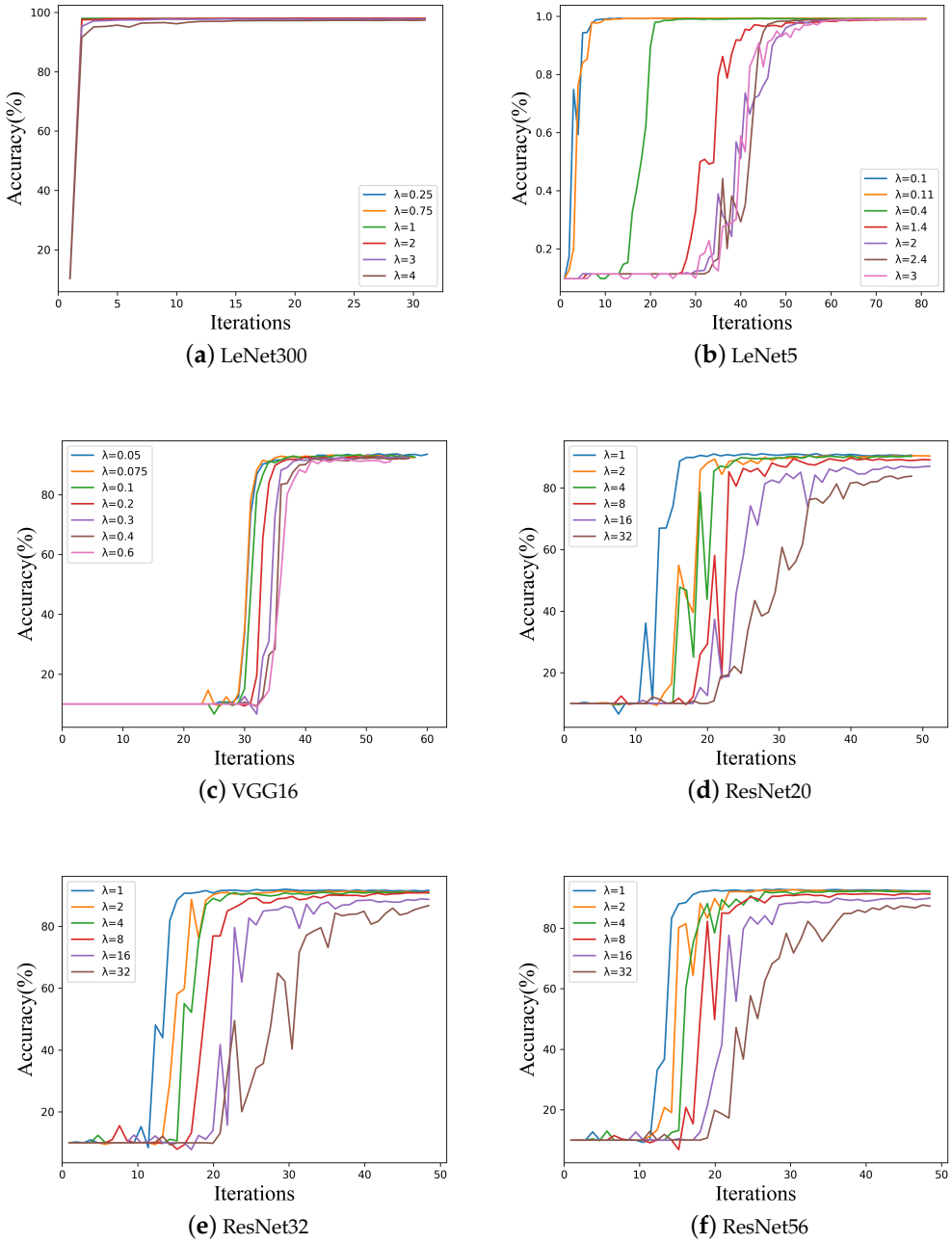


Figure 3. The model classification accuracy with different $\lambda \times 10^{-4}$, for LeNets on MNIST, VGG16 and ResNets on CIFAR-10.

In Figure 4, the results of our algorithm over different λ values for a given network span a curve, shown as connected circles which start on the lower right at the reference labeled "R" ($\lambda = 0$) and then move left and up. In this figure, ideal models are small balls (having few parameters) on the left-bottom (where both test error and FLOPs are the smallest). Each color corresponds to a different reference net. The area of a circle or square is proportional to the number of weight parameters (storage) in the corresponding compressed model.

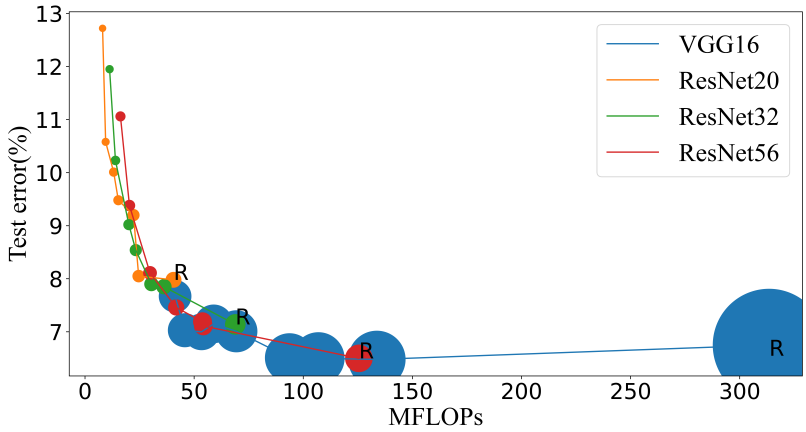
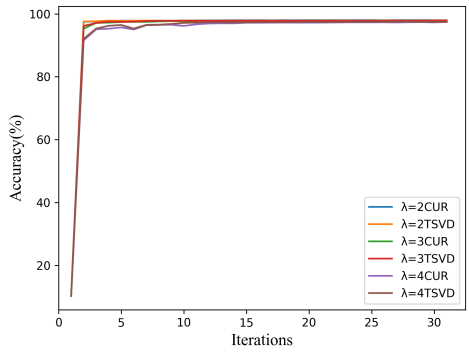


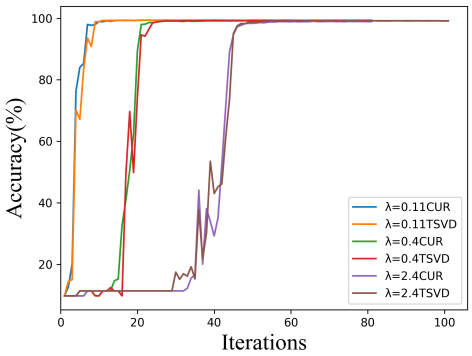
Figure 4. The error-compression space of test error (Y axis), inference FLOPs (X axis) and number of parameters (ball size), for LeNets on MNIST, VGG16 and ResNets on CIFAR-10.

We compute FLOPs based on fusing multiplication and addition as the assumption, and treat it as one FLOP. For example, if the forward passes through a fully connected layer with $n \times m$ weight matrix and $n \times 1$ bias, which has nm multiplications and n additions, then the FLOPs are nm. In this figure, each color corresponds to a different reference grid, and the ideal models are generally the small ball in the lower left corner, that is, with fewer parameters, lower error rate, and smaller FLOPs. It can be seen from the figure that the FLOPs of the VGG16 network are lower than that of the uncompressed ResNet model (labeled "R") when the classification accuracy is almost the same. That is, the VGG16 network compressed by our algorithm can be faster than the uncompressed ResNets, but with almost the same test error.

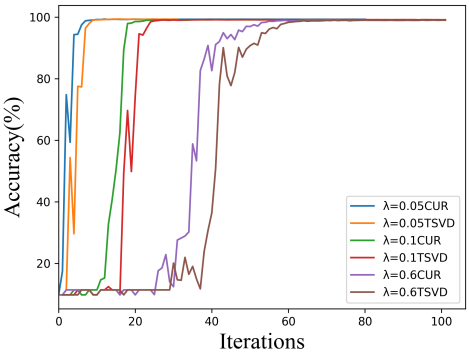
From the experimental results it can be seen that the compression ratio grows with the growth of λ. In addition to CUR decomposition, we also use TSVD as the low-rank matrix decomposition method of the compression step in our proposed algorithm, and compare the classification accuracy, loss, compression space (storage parameters and FLOPs), and Frobenius norm $\|\mathbf{W} - \mathbf{\Theta}\|_{\mathrm{F}}^2$ of the two decomposition methods in the compression step based on the classification task. All hyperparameters involved are controlled, and the experimental results are shown in Figures 5–8. It can be clearly seen from Figures 5–7 that the performance (classification accuracy, Frobenius norm and learning loss) of our algorithms using TSVD and CUR decomposition, respectively, is basically the same when λ is small. However, with the increase of λ, the experimental results of the algorithm using CUR decomposition gradually surpass the latter in terms of accuracy and have a faster classification-loss drop rate. In terms of F-norm, similarly, the numerical difference between above two decomposition methods is not obvious when λ is small. As λ gradually increases, compared with the joint optimization of TSVD, the one using CUR decomposition makes the inflection point recording the Frobenius norm appear earlier and take fewer iterations to reach the approximate zero point. The experimental performance in Figure 8 is also the same. Compared with the joint optimization of TSVD, the algorithm using CUR decomposition is closer to the lower left position (ideal position) when λ increases, which means that our algorithm using CUR decomposition performs better at high compression ratios.



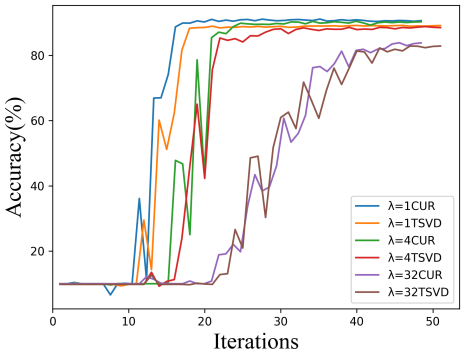
(**a**) Accuracy of LeNet300



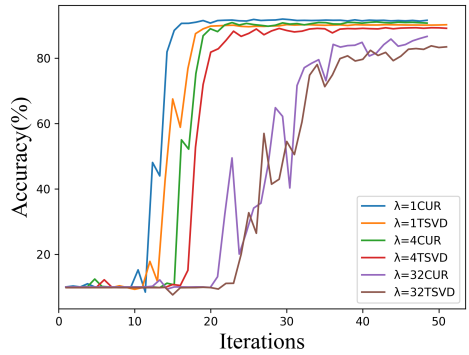
(**b**) Accuracy of LeNet5



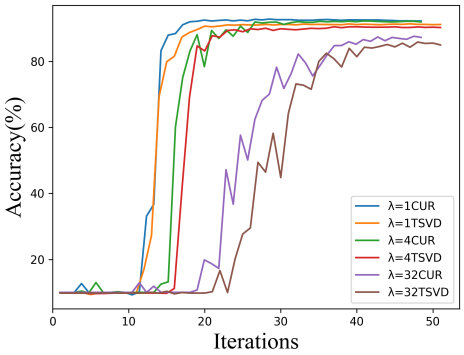
(**c**) Accuracy of VGG16



(**d**) Accuracy of ResNet20

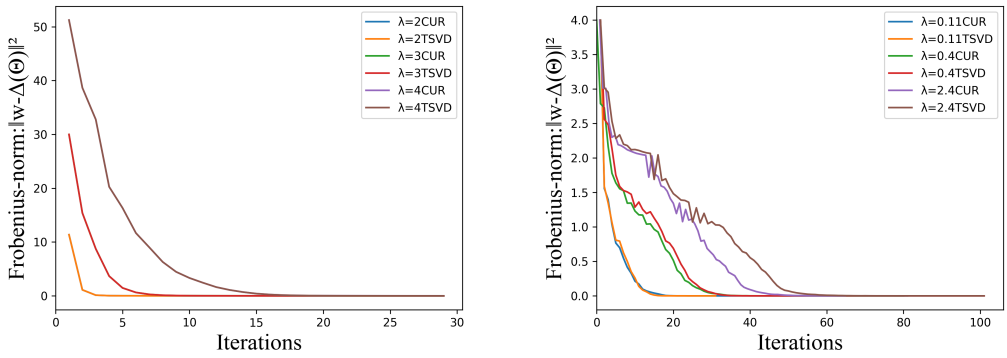


(**e**) Accuracy of ResNet32



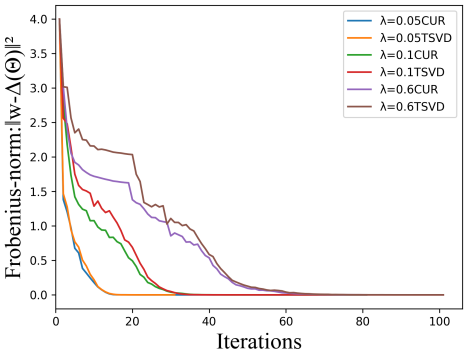
(**f**) Accuracy of ResNet56

Figure 5. Comparison of accuracy with different $\lambda 10^{-4}$, for LeNets on MNIST, VGG16 and ResNets on CIFAR-10.

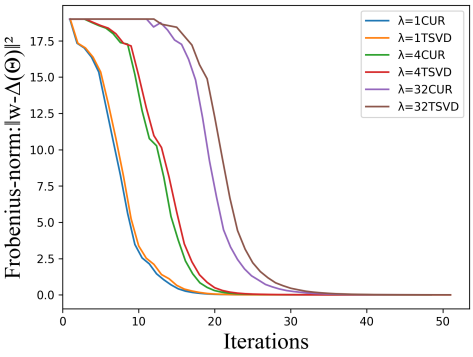


(**a**) F-norm of LeNet300

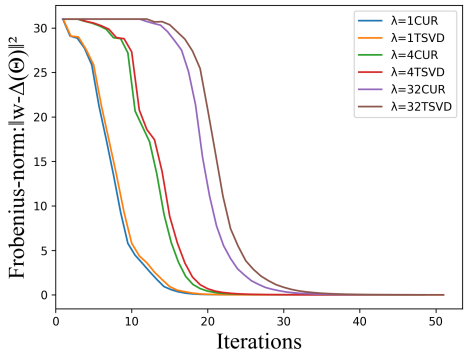
(**b**) F-norm of LeNet5



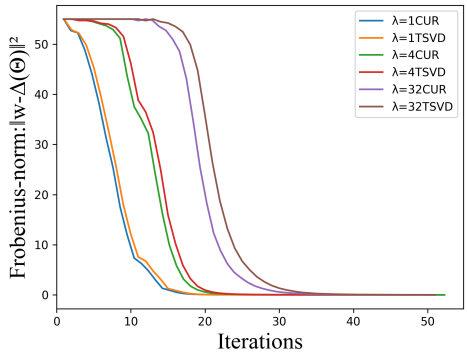
(**c**) F-norm of VGG16



(**d**) F-norm of ResNet20

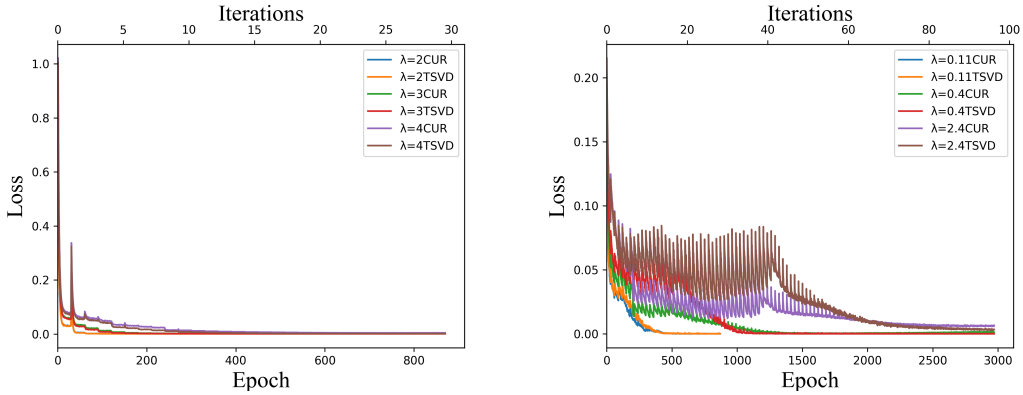


(**e**) F-norm of ResNet32



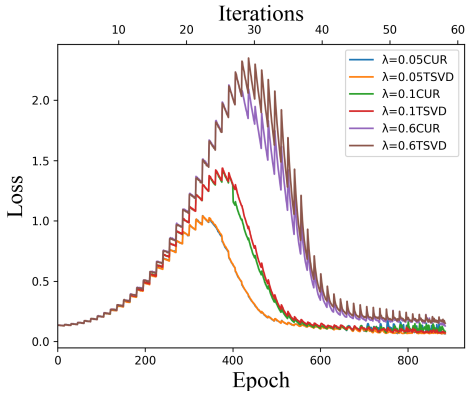
(**f**) F-norm of ResNet56

Figure 6. Comparison of F-norm with different $\lambda 10^{-4}$, for LeNets on MNIST, VGG16 and ResNets on CIFAR-10.

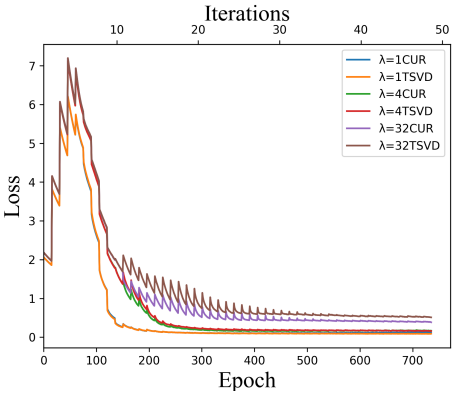


(**a**) Loss of LeNet300

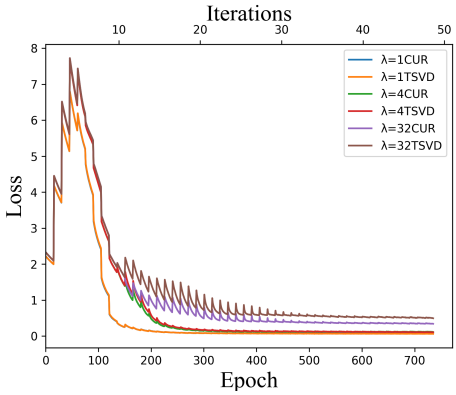
(**b**) Loss of LeNet5



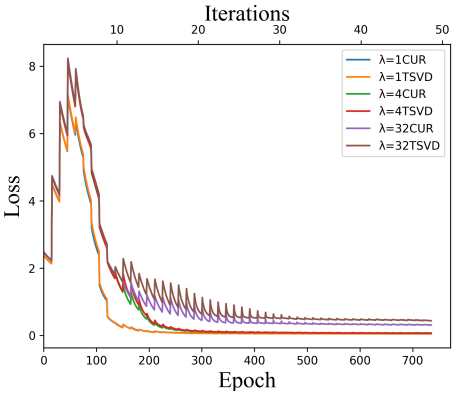
(**c**) Loss of VGG16



(**d**) Loss of ResNet20



(**e**) Loss of ResResNet32



(**f**) Loss of ResNet56

Figure 7. Comparison of learning loss with different $\lambda 10^{-4}$, for LeNets on MNIST, VGG16 and ResNets on CIFAR-10.

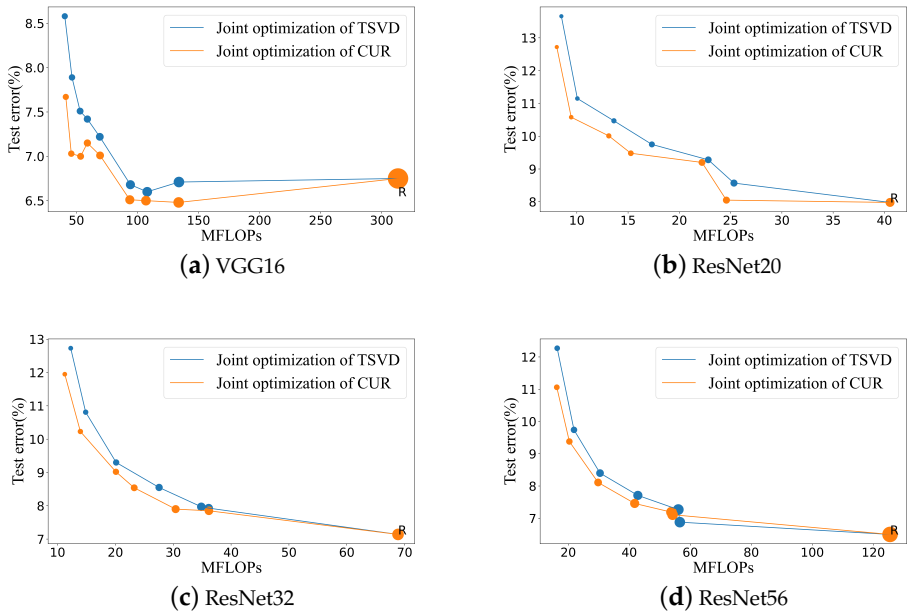


Figure 8. Comparison of classification test error (Y-axis), FLOPs (X-axis) and the number of storage parameters (ball size for each net) in Error-compression space, for ResNets and VGG16 trained on CIFAR-10, the algorithm using TSVD and CUR decomposition.

5. Conclusions

Due to recent technological advancements, DNNs have been more widely used as advanced and powerful methods. However, calculation resource and space are high as a result of the huge number of parameters. Moreover, the high-dimensional data in DNNs usually contains useless information, which will also reduce the accuracy to a certain extent. An optimization algorithm that reduces the calculation resource and space cost of the DNN model while maintaining the original accuracy is required to compress (and optimize) the DNN models. The main goal of this paper is to propose a DNN compression algorithm which incorporates the matrix factorization method and the linear compression function into the joint function optimization framework, reducing the number of parameters in the networks and thereby reducing storage space and computational resource consumption. We deal with this joint function using the ALM and split it into learning and compression steps, respectively, so that the networks can be trained and compressed independently without interfering with each other. All experiments are carried on classification tasks over three public datasets. It is worth noting that the experimental results show that our algorithm has the highest accuracy and compression ratios when compared with other baselines and the state-of-the-art research. For example, we can make a VGG network faster than ResNets and with nearly the same classification accuracy. In addition, the compressed models of VGG16 on CIFAR-10 and ResNet20 on CIFAR-100 using our algorithms achieve test errors of 6.15% and 33.22%, which are 0.60% and 1.38% lower than the uncompressed models, respectively. The current work also aims to examine the convergence and sensitivity of our algorithm. The results show that our algorithm has stable convergence and the weight matrices have low-rank properties, as can be seen from the values of learning loss and Frobenius norm $\|\mathbf{W} - \mathbf{\Theta}\|_{\mathrm{F}}^2$. Moreover, under same ranks, we use the classic TSVD, CUR decomposition, and the joint optimization framework-based TSVD compression methods as baselines to carry out experiments. The comparison results prove the superiority in compression effects and compatibility with MF methods. However, as mentioned above, both TSVD and CUR decomposition are matrix factorization methods with cubic level complexity, and GPU is not good at SVD and CUR decomposition in practice. In other words, the larger the

network size, the longer compression runtime required. For future works, we attempt to develop more research on how to cut down the runtime of compression for DNNs.

Author Contributions: Conceptualization, G.C.; Methodology, G.C. and J.L.; Software, G.C. and X.L.; Validation, G.C., J.L. and Z.C.; Formal analysis, G.C., X.L. and H.Z.; Investigation, G.C., J.L. and H.Z.; Resources, Z.C.; Data curation, G.C.; Writing—original draft, G.C.; Writing—review and editing, G.C., J.L., X.L., Z.C. and H.Z.; Visualization, G.C.; Supervision, J.L. and X.L.; Project administration, Z.C.; Funding acquisition, Z.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China, grant number 32071775.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable. All of the datasets used in this study were downloaded from a public online database.

Acknowledgments: This research was supported by the National Natural Science Foundation of China, grant number 32071775.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, Z.; Liu, F.; Yang, W.; Peng, S.; Zhou, J. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *33*, 6999–7019. [CrossRef] [PubMed]
2. Wang, Y.; Dong, M.; Shen, J.; Luo, Y.; Lin, Y.; Ma, P.; Petridis, S.; Pantic, M. Self-supervised Video-centralised Transformer for Video Face Clustering. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, 1–16. [CrossRef]
3. ErdoganyÄśmaz, C.; Mengunogul, B. An Original Natural Language Processing Approach to Language Modeling in Turkish Legal Corpus: Improving Model Performance with Domain Classification by Using Recurrent Neural Networks. In Proceedings of the 2022 Innovations in Intelligent Systems and Applications Conference (ASYU), Antalya, Turkey, 7–9 September 2022; pp. 1–6. [CrossRef]
4. Hu, D.; Si, Q.; Liu, R.; Bao, F. Distributed Sensor Selection for Speech Enhancement with Acoustic Sensor Networks. *IEEE/Acm Trans. Audio Speech Lang. Process.* **2023**, 1–15/ [CrossRef]
5. Zhang, H.; Si, N.; Chen, Y.; Zhang, W.; Yang, X.; Qu, D.; Zhang, W. Improving Speech Translation by Cross-modal Multi-grained Contrastive Learning. *IEEE/Acm Trans. Audio Speech Lang. Process.* **2023**, 1–12. [CrossRef]
6. Qi, J.; Yang, C.H.H.; Chen, P.Y.; Tejedor, J. Exploiting Low-Rank Tensor-Train Deep Neural Networks Based on Riemannian Gradient Descent With Illustrations of Speech Processing. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2023**, *31*, 633–642. [CrossRef]
7. Zhuang, T.; Zhang, Z.; Huang, Y.; Zeng, X.; Shuang, K.; Li, X. Neuron-level structured pruning using polarization regularizer. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 9865–9877.
8. Zhang, L.; Yang, C.; Lu, H.; Ruan, X.; Yang, M.H. Ranking saliency. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *39*, 1892–1904. [CrossRef] [PubMed]
9. Qu, S.; Li, B.; Zhao, S.; Zhang, L.; Wang, Y. A Coordinated Model Pruning and Mapping Framework for RRAM-based DNN Accelerators. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2022**, 1. [CrossRef]
10. Ma, X.; Yuan, G.; Li, Z.; Gong, Y.; Zhang, T.; Niu, W.; Zhan, Z.; Zhao, P.; Liu, N.; Tang, J.; et al. BLCR: Towards Real-time DNN Execution with Block-based Reweighted Pruning. In Proceedings of the 2022 23rd International Symposium on Quality Electronic Design (ISQED), Santa Jose, CA, USA, 6–7 April 2022; pp. 1–8. [CrossRef]
11. Carreira -Perpinán, M.A.; Idelbayev, Y. Model compression as constrained optimization, with application to neural nets. Part II: Quantization. *arXiv* **2017**, arXiv:1707.04319.
12. Nagel, M.; Baalen, M.v.; Blankevoort, T.; Welling, M. Data-free quantization through weight equalization and bias correction. In Proceedings of the Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019; pp. 1325–1334.
13. Vanhoucke, V.; Senior, A.; Mao, M.Z. Improving the speed of neural networks on CPUs. Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011. Available online: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/37631.pdf (accessed on 15 February 2023).
14. Li, H.; De, S.; Xu, Z.; Studer, C.; Samet, H.; Goldstein, T. Training quantized nets: A deeper understanding. In Proceedings of the NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; Volume 30.
15. Idelbayev, Y.; Carreira-Perpinán, M.A. A flexible, extensible software framework for model compression based on the LC algorithm. *arXiv* **2020**, arXiv:2005.07786.

16. Song, M.; Yang, T.; Cao, H.; Li, F.; Xue, B.; Li, S.; Chang, C.I. Bi-Endmember Semi-NMF Based on Low-Rank and Sparse Matrix Decomposition. *IEEE Trans. Geosci. Remote Sens.* **2022**, *60*, 1–16. [CrossRef]
17. Mai, A.; Tran, L.; Tran, L.; Trinh, N. VGG deep neural network compression via SVD and CUR decomposition techniques. In Proceedings of the 2020 7th NAFOSTED Conference on Information and Computer Science (NICS), Ho Chi Minh City, Vietnam, 26–27 November 2020; pp. 118–123.
18. Li, T.; Wu, B.; Yang, Y.; Fan, Y.; Zhang, Y.; Liu, W. Compressing convolutional neural networks via factorized convolutional filters. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 3977–3986. [CrossRef]
19. Lee, D.; Seung, H.S. Algorithms for non-negative matrix factorization. *Adv. Neural Inf. Process. Syst.* **2000**, *13*, 556–562.
20. Tepper, M.; Sapiro, G. Compressed Nonnegative Matrix Factorization Is Fast and Accurate. *IEEE Trans. Signal Process.* **2016**, *64*, 2269–2283. [CrossRef]
21. Guo, Z.; Zhang, S. Sparse deep nonnegative matrix factorization. *Big Data Min. Anal.* **2019**, *3*, 13–28. [CrossRef]
22. Yang, X.; Che, H.; Leung, M.F.; Liu, C. Adaptive graph nonnegative matrix factorization with the self-paced regularization. *Appl. Intell.* **2022**, 1–18. [CrossRef]
23. Aharon, M.; Elad, M.; Bruckstein, A. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Trans. Signal Process.* **2006**, *54*, 4311–4322. [CrossRef]
24. Kalman, D. A singularly valuable decomposition: The SVD of a matrix. *Coll. Math. J.* **1996**, *27*, 2–23. [CrossRef]
25. Benjamin Erichson, N.; Brunton, S.L.; Nathan Kutz, J. Compressed singular value decomposition for image and video processing. In Proceedings of the IEEE International Conference on Computer Vision Workshops, Venice, Italy, 22–29 October 2017; pp. 1880–1888.
26. Yang, H.; Tang, M.; Wen, W.; Yan, F.; Hu, D.; Li, A.; Li, H.; Chen, Y. Learning Low-rank Deep Neural Networks via Singular Vector Orthogonality Regularization and Singular Value Sparsification. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Seattle, WA, USA, 14–19 June 2020; pp. 2899–2908. [CrossRef]
27. Swaminathan, S.; Garg, D.; Kannan, R.; Andres, F. Sparse low rank factorization for deep neural network compression. *Neurocomputing* **2020**, *398*, 185–196. [CrossRef]
28. Aldroubi, A.; Hamm, K.; Koku, A.B.; Sekmen, A. CUR decompositions, similarity matrices, and subspace clustering. *Front. Appl. Math. Stat.* **2019**, *4*, 65. [CrossRef]
29. Chen, M.; Li, X. Robust Matrix Factorization With Spectral Embedding. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 5698–5707. [CrossRef]
30. Enríquez Pinto, M.A. Big Data Analysis Using CUR Algorithm. Ph.D. Thesis, Universidad de Investigación de Tecnología Experimental Yachay, Urcuqui, Ecuador, 2021.
31. Voronin, S.; Martinsson, P.G. Efficient algorithms for CUR and interpolative matrix decompositions. *Adv. Comput. Math.* **2017**, *43*, 495–516. [CrossRef]
32. Mahoney, M.W.; Drineas, P. CUR matrix decompositions for improved data analysis. *Proc. Natl. Acad. Sci. USA* **2009**, *106*, 697–702. [CrossRef] [PubMed]
33. Chung, H.; Chung, E.; Park, J.G.; Jung, H.Y. Parameter Reduction For Deep Neural Network Based Acoustic Models Using Sparsity Regularized Factorization Neurons. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; pp. 1–5.
34. Kim, T.; Lee, J.; Choe, Y. Bayesian optimization-based global optimal rank selection for compression of convolutional neural networks. *IEEE Access* **2020**, *8*, 17605–17618. [CrossRef]
35. Kim, H.; Kyung, C. Automatic Rank Selection for High-Speed Convolutional Neural Network. *arXiv* **2018**, arXiv:1806.10821.
36. Phan, A.H.; Sobolev, K.; Sozykin, K.; Ermilov, D.; Gusak, J.; Tichavskỳ, P.; Glukhov, V.; Oseledets, I.; Cichocki, A. Stable low-rank tensor decomposition for compression of convolutional neural network. In Proceedings of the Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, 23–28 August 2020; Part XXIX 16; Springer: Berlin/Heidelberg, Germany, 2020, pp. 522–539.
37. Idelbayev, Y.; Carreira-Perpiñán, M.Á. Beyond FLOPs in low-rank compression of neural networks: Optimizing device-specific inference runtime. In Proceedings of the 2021 IEEE International Conference on Image Processing (ICIP), Anchorage, Alaska, USA, 19–22 September 2021; pp. 2843–2847.
38. Carreira-Perpinán, Miguel A. Model compression as constrained optimization, with application to neural nets. Part I: General framework *arXiv* **2017**, arXiv:1707.01209.
39. Golub, G.H.; Van Loan, C.F. *Matrix Computations*; JHU Press: Baltimore, MD, USA, 2013.
40. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
41. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
42. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
43. Krizhevsky, A.; Hinton, G. Learning Multiple Layers of Features from Tiny Images. Master's Thesis, Department of Computer Science, University of Toronto, Toronto, ON, Canada, 8 April 2009.
44. Wen, W.; Xu, C.; Wu, C.; Wang, Y.; Chen, Y.; Li, H. Coordinating Filters for Faster Deep Neural Networks. In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 658–666. [CrossRef]

45. Idelbayev, Y.; Carreira-PerpinÃąn, M.A. Optimal selection of matrix shape and decomposition scheme for neural network compression. In Proceedings of the ICASSP 2021—2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, 6–11 June 2021; pp. 3250–3254.
46. Nesterov, Y. A method for solving the convex programming problem with convergence rate O($\frac{1}{k^2}$). *Proc. USSR Acad. Sci.* **1983**, *269*, 543–547.
47. Aggarwal, V.; Wang, W.; Eriksson, B.; Sun, Y.; Wang, W. Wide Compression: Tensor Ring Nets. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; IEEE Computer Society: Los Alamitos, CA, USA, 2018; pp. 9329–9338. [CrossRef]
48. Li, N.; Pan, Y.; Chen, Y.; Ding, Z.; Zhao, D.; Xu, Z. Heuristic rank selection with progressively searching tensor ring network. *Complex Intell. Syst.* **2021**, *8*, 771–785. [CrossRef]