Joint Matrix Decomposition for Deep Convolutional Neural Networks Compression

Shaowu Chen, Jiahao Zhou, Weize Sun*, Lei Huang

***Abstract*—Deep convolutional neural networks (CNNs) with a large number of parameters require intensive computational resources, and thus are hard to be deployed in resource-constrained platforms. Decomposition-based methods, therefore, have been utilized to compress CNNs in recent years. However, since the compression factor and performance are negatively correlated, the state-of-the-art works either suffer from severe performance degradation or have relatively low compression factors. To overcome this problem, we propose to compress CNNs and alleviate performance degradation via joint matrix decomposition, which is different from existing works that compressed layers separately. The idea is inspired by the fact that there are lots of repeated modules in CNNs. By projecting weights with the same structures into the same subspace, networks can be jointly compressed with larger ranks. In particular, three joint matrix decomposition schemes are developed, and the corresponding optimization approaches based on Singular Value Decomposition are proposed. Extensive experiments are conducted across three challenging compact CNNs for different benchmark data sets to demonstrate the superior performance of our proposed algorithms. As a result, our methods can compress the size of ResNet-34 by $22\times$ with slighter accuracy degradation compared with several state-of-the-art methods.**

***Index Terms*—deep convolutional neural network, network compression, model acceleration, joint matrix decomposition.**

I. Introduction

In recent years, deep neural networks (DNNs), including deep convolutional neural networks (CNNs) and multilayer perceptron (MLPs), have achieved great successes in various areas, *e.g.*, noise reduction, object detection and matrix completion [1]–[3]. To achieve satisfactory performance, very deep and complicated DNNs with a cumbersome number of parameters and billions of floating point operations (FLOPs) requirements are developed [4]–[7]. Although high-performance servers with GPUs can meet the requirements of the DNNs, it is problematic to deploy them on resource-constrained platforms such as embedded or mobile devices [8][9], especially when real-time forward inference is required. To tackle this problem, methods for network compression are developed.

It has been found that the weight matrices of fully connected (FC) layers and weight tensors of convolutional layers are of low rank [10], therefore the redundancy among these layers can be removed via decomposition methods. In [10]–[13], matrix decomposition alike methods are utilized to compress MLPs and CNNs in a one-shot manner or layer by layer progressively, in which a weight matrix $\mathbf{W}$ is decomposed as $\mathbf{W} \approx \mathbf{U}\mathbf{V}$ where $\text{size}(\mathbf{U}) + \text{size}(\mathbf{V}) \ll \text{size}(\mathbf{W})$, and relatively small compression factors (CF) are achieved at the cost of a drop in accuracy. Recent works focus more on compressing CNNs, and to avoid unfolding 4-D weight tensors of convolutional layers into 2-D matrices when implementing decomposition, tensor-decomposition-based methods [8][14]–[17] are introduced to compress CNNs. However, CNNs are much more compacted than MLPs because of the properties of parameter sharing, making the compression of CNNs challenging, thus previous works either compress CNNs with a small CF between $2\times$ and $5\times$ or suffer from severe performance degradation. Some methods are proposed to alleviate degradation with a scheme associating properties of low rank and sparsity [18][19], in which $\mathbf{W} \approx \mathbf{U}\mathbf{V} + \mathbf{S}$, where $\mathbf{S}$ is a highly sparse matrix. However, without support from particular libraries, the memory, storage and computation consumption of $\mathbf{S}$ is as large as $\mathbf{W}$ since $\text{size}(\mathbf{S}) = \mathbf{W}$, and the "0" elements as 2^n-bits numbers consume the same resources as those non-zero ones, not to mention the ones caused by $\mathbf{U}$ and $\mathbf{V}$, making the method less practical.

To greatly compress CNNs without severe performance degradation, we propose to compress them via joint matrix decomposition. The main difference between our scheme and the state-of-the-art works is that we jointly decompose layers with relationships instead of compressing them separately. The idea is inspired by a basic observation that the widely used CNNs tend to adapt repeated modules to attain satisfying performance [5]–[7], making lots of convolutional layers sharing the same structure. Therefore, by projecting them into the same subspace, weight tensors can share identical factorized matrices, and thus CNNs can be further compressed. Taking part of ResNet shown in Figure 1 as an example, $\mathbf{W}_1^n, n = 1, \cdots, N$ have the same structure and are placed in the consistent position of N BasicBlocks, thus might contain similar information and can be jointly decomposed, *i.e.*, $\mathbf{W}_1^n = \mathbf{G}_1^n \Sigma_1^n \mathbf{V}_1 = \mathbf{U}_1^n \mathbf{V}_1$ where $\mathbf{V}_1$ is identical for all $\mathbf{W}_1^n, n = 1, \cdots, N$, and $\mathbf{U}_1^n = \mathbf{G}_1^n \Sigma_1^n$. In this manner, there are only about half of the matrices needed to be stored, *i.e.*, $\mathbf{U}_1^1, \mathbf{U}_1^2, \cdots, \mathbf{U}_1^N$ and $\mathbf{V}_1$ instead of $\mathbf{U}_1^1, \mathbf{U}_1^2, \cdots, \mathbf{U}_1^N$ and $\mathbf{V}_1^1, \mathbf{V}_1^2, \cdots, \mathbf{V}_1^N$, and thus the requirement of storage and memory resources can be further reduced. With the shared $\mathbf{V}_1$ containing the common information across multiple layers in large CNNs, the joint decomposition methods can retain larger ranks compared with traditional rank-truncated decomposition methods under the same compression degree,

Shaowu Chen, Jiahao Zhou, Weize Sun and Lei Huang are with the College of Electronics and Information Engineering, Shenzhen University, Shenzhen 518060, Guangdong, China. (e-mail: shaowuchen@foxmail.com; plus_chou@foxmail.com; proton198601@hotmail.com; lhuang8sasp@hotmail.com).

*Corresponding author: Weize Sun.

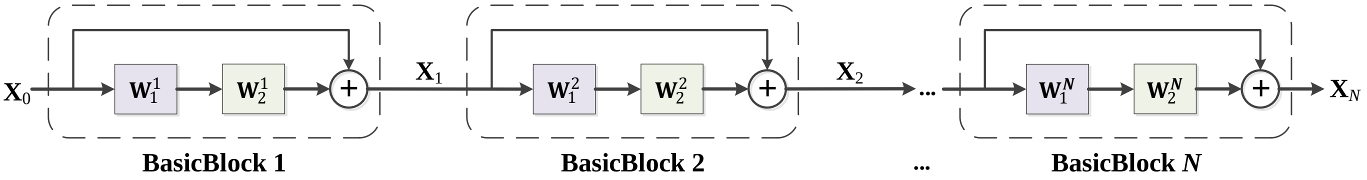

Fig. 1: A common network structure in ResNet. Here we assume that weights of convolutional layers are 2-D matrices instead of 4-D tensors for the convenience of explanation.

and thus alleviate performance degradation in the compressed models. After decomposing $\mathbf{W}_1^n, n = 1, \cdots, N$, the similar procedure can be implemented on $\mathbf{W}_2^n, n = 1, \cdots, N$ as well. In our previous work [20], the relationship of layers was also taken into account but with a clearly different idea, which considered a weight as the summation of an independent component and a shared one, and Tucker Decomposition [8] or Tensor Train Decomposition [21] was utilized to factorized them, *i.e.*, $\mathbf{W}_m^n = \mathbf{W}_{m,i}^n + \mathbf{W}_{m,s} = \mathbf{G}_{m,i,1}^n * \mathbf{G}_{m,i,2}^n * \mathbf{G}_{m,i,3}^n + \mathbf{G}_{m,s,1} * \mathbf{G}_{m,s,2} * \mathbf{G}_{m,s,3}$, for $n = 1, \cdots, N$, while in this paper we directly project weights into the same subspace via joint matrix decomposition.

In the above illustration, for convenience, the weight $\mathbf{W}$ in a convolutional layer is supposed to be two-dimensional, while they are 4-D tensors in reality. To implement joint matrix decomposition, we firstly unfold 4-D weight tensors into 2-D weight matrices following [13], and then introduce a novel Joint Singular Value Decomposition (JSVD) method to decompose networks jointly. Two JSVD algorithms for network compression are proposed, referred to as left shared JSVD (**LJSVD**) and right shared JSVD (**RJSVD**), respectively, according to the relative position of the identical factorized matrix. Besides, by combining LJSVD and RJSVD, another algorithm named Binary JSVD (**Bi-JSVD**) is also proposed, which can be considered as the generalized form of LJSVD and RJSVD. These algorithms can decompose pre-trained CNNs following the "pre-train→decompose→fine-tune" pipeline and train compressed CNNs from scratch by retaining the decomposed structures but randomly initializing weights. Furthermore, as a convolutional layer can be divided into two consecutive slimmer ones with less complexity by matrix decomposition, our methods can not only compress networks but also accelerate them without support from customized libraries and hardware.

The remainder of this paper is organized as follows. In Section II, some of the related works are reviewed. In Section III, necessary notations and definitions are introduced first, and then the proposed joint matrix decomposition methods for network compression are developed. Section IV presents extensive experiments evaluating the proposed methods, and the results are discussed in detail. Finally, a brief conclusion and future work are given in Section V.

II. Related Works

The methods for neural network compression can be roughly divided into four categories, namely, pruning, quantization, knowledge distillation and decomposition. Furthermore, the decomposition methods can be further divided into matrix-based and tensor-based ones. Note that the pruning, quantization and knowledge distillation approaches are orthogonal to decomposition methods, and thus can be combined to achieve better performance in some cases.

a) Pruning: Pruning methods evaluate the importance of neurons or filters and eliminate those with the lowest scores. Its origins can date back to the 1990s [22], which used second-order derivative information to prune neurons that bring minor changes to the loss function. In [23][24], weights with magnitudes lower than a threshold are pruned iteratively, while the works in [19][25] pruned CNNs using evolutionary computing methods. The aforementioned methods pruned the elements in weight tensors by setting them to zero, resulting in unstructured sparsity that is less effective for compression and acceleration. In contrast, the works in [26][27] directly removed redundant filters and thus can effectively compress CNNs.

b) Quantization: Some researchers compress and accelerate CNNs using low-precision and fixed-point number representations for weights. The works in [28] and [29] successfully utilized binary and integer numbers, respectively, to reduce the sizes and expensive floating point operations of CNNs. These works used quantization techniques in the training stage, while some post-quantization approaches [30][31] trained full-precision CNNs at first and then quantized them to obtain low-precision results. Similar to our methods, most post-training quantization methods require fine-tuning to alleviate performance degradation.

c) Knowledge distillation: In general, larger networks contain more knowledge and thus outperform smaller networks. Therefore, knowledge distillation aims at transferring the knowledge of large pre-trained teacher-CNNs to small student-CNNs [32]. After transferring, the large CNNs are discarded while the smaller teacher-CNNs with similar performance are utilized. Hinton *et al.* [32] implemented knowledge distillation by reducing the differentiation between softmax output of teacher-CNNs and student-CNNs, and researchers extended knowledge distillation by matching other statistics, such as intermediate feature maps [33]–[35] and gradient [36]. The idea of knowledge distillation can also be combined for our methods to help fine-tune the compressed CNNs and further improve performance.

d) Matrix decomposition: In [10], weight tensors of convolutional layers were considered of low rank. Therefore, they were unfolded to 2-D matrices and then compressed via low-rank decomposition, in which the reconstruction ICA [37] and ridge regression with the squared exponential kernel were used to predict parameters. Inspired by this, the work in [11] combined SVD and several clustering schemes to achieve a 3.9× weight reduction for a single convolutional layer. In [13], CNNs were compressed by removing spatial or channel redundancy via low rank regularization, and a 5× compression on AlexNet was achieved [13]. With the idea similar to knowledge distilling, some researchers initially trained an over-parameter network and then compressed it via matrix decomposition to meet budget requirements, thus making the compressed network outperform the one trained from scratch directly with the same sizes [38]. The works in [18][19] combined the property of sparsity with the low rank one to obtain higher compression factors, but need supports from customized libraries to implement storage compression and computation acceleration. In [39], the sparsity was embedded in the factorized low rank matrices, which can be considered as the combination of unstructured pruning and matrix decomposition.

e) Tensor decomposition: Since the weight of a convolution layer is a 4-D tensor, tensor decomposition methods were applied to compress neural networks naturally to achieve higher compression factors. Some compact modules can be regarded as derivations from tensor decomposition as well [40], such as the depthwise separable convolution in MobileNet [41]. In [16], Tensor Train Decomposition (TTD) was extended from MLPs [14] to compress CNNs. To achieve a higher degree of reduction in sizes, weight tensors in [16] were folded to higher dimensional tensors and then decomposed via TTD. As a result, a CNN were compressed by $4.02\times$ with a 2% loss of accuracy. In [8], spatial dimensions of a weight tensor were merged to form a 3-D tensor, and then Tucker Decomposition was utilized to divide the original convolutional layer into three layers with less complexity. Note that in this case, the Tucker Decomposition is equal to TTD. To alleviate performance degradation caused by Tucker, the work in [20] considered a weight tensor as the summation of the independent and shared components, and applied Tucker Decomposition to each of them, respectively. In [42], Hybrid Tensor decomposition was utilized to achieve $4.29\times$ compression across a CNN on CIFAR-10 at the cost of 5.18% loss in the accuracy. At the same time, TTD [16] only caused $1.27\times$ loss, indicating that TTD is more suitable for compressing convolution layers.

III. Methodology

In this section, some necessary notations and definitions are introduced first, and then three joint compression algorithms, RJSVD, LJSVD and Bi-JSVD, are proposed.

A. Preliminaries

Notations. The notations and symbols used in this paper are introduced as follows. Scalars, vectors, matrices (2-D), and tensors (with more than two dimensions) are denoted by italic, bold lowercase, bold uppercase, and bold calligraphic symbols, respectively. Following the conventions of Tensorflow, we represent an input tensor of one convolution layer as $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{H_1 \times W_1 \times I}$, the output as $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{H_2 \times W_2 \times O}$, and the corresponding weight tensor as $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{F_1 \times F_2 \times I \times O}$, where H_1, W_1, H_2, W_2 are spatial dimensions, $F_1 \times F_2$ is the size of a filter, while I and O are the input and output depths, respectively. Sometimes we would put the sizes of a tensor or matrix in its subscript to clarify the dimensions, such as $\mathbf{W}_{I \times O}$ means $\mathbf{W} \in \mathbb{R}^{I \times O}$.

To decompose a group of layers jointly, we further denote weights tensors or matrices with subscripts and superscripts such as $\boldsymbol{\mathcal{W}}_m^n$, for $n = 1, \cdots, N$ and $m = 1, \cdots, M$, where m distinguishes different groups and n distinguishes different elements inside a group. In other words, $\boldsymbol{\mathcal{W}}_m^n$, $n = 1, \cdots, N$ under the same m would be jointly decomposed. Figure 1 is a concrete example with $M = 2$, and $\mathbf{W}_1^n$, $n = 1, 2, \cdots, N$ is a group of weights that would be jointly decomposed, so as $\mathbf{W}_2^n$, $n = 1, 2, \cdots, N$.

General unfolding. When applying matrix decomposition to compress convolutional layers, the first step is to unfold weight tensors $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{F_1 \times F_2 \times I \times O}$ into 2-D matrices. Since the kernel sizes F_1 and F_2 are usually small values such as 3 or 5, following [13], we merge the first and third dimensions and then the second and fourth ones. We refer to this operation as general unfolding and denote it by **Unfold**$(\cdot)$, which swaps the first and third axes and then merges the first two and the last dimensions, respectively, to produce $\text{Unfold}(\boldsymbol{\mathcal{W}}) = \mathbf{W} \in \mathbb{R}^{F_1 I \times F_2 O}$.

General folding. We call the opposite operation of general unfolding as general folding, denoted as **Fold**$(\cdot)$. By performing general folding on $\mathbf{W} \in \mathbb{R}^{F_1 I \times F_2 O}$, there would be $\text{Fold}(\mathbf{W}) = \boldsymbol{\mathcal{W}} \in \mathbb{R}^{F_1 \times F_2 \times I \times O}$. For ease of presentation, in the remainder of this article, **notations $\boldsymbol{\mathcal{W}}$ and W would represent the general unfold weight tensor and the corresponding general folded matrix, respectively**, unless there are particular statements.

Truncated rank-r SVD. The $\text{SVD}_r(\cdot)$ represents the truncated rank-r SVD for networks compression, where r is the truncated rank. Note that r is a user-defined hyperparameter that decides the number of singular values and vectors in the compressed model. By performing truncated rank-r SVD, we have $\mathbf{W}_{F_1 I \times F_2 O} \approx \mathbf{G}_{F_1 I \times r} \Sigma_{r \times r} \mathbf{V}_{r \times F_2 O} = \mathbf{U}_{F_1 I \times r} \mathbf{V}_{r \times F_2 O}$, where Σ is a diagonal matrix consisting of singular values in descending order, $\mathbf{G}$, $\mathbf{V}$ are orthogonal factorized matrices, $\mathbf{U} = \mathbf{G}\Sigma$, and $r \leq \min\{F_1 I, F_2 O\}$. After the decomposition, it is $\mathbf{U}$ and $\mathbf{V}$ that are stored instead of $\mathbf{W}$ or $\boldsymbol{\mathcal{W}}$, therefore the network is compressed by a factor of $\frac{F_1 F_2 I O}{r(F_1 I + F_2 O)} \times$ where $r < \frac{F_1 F_2 I O}{(F_1 I + F_2 O)}$.

B. Proposed Algorithms

1) Right Shared Joint SVD (RJSVD): For $\boldsymbol{\mathcal{W}}_m^n \in \mathbb{R}^{F_1 \times F_2 \times I \times O}$, $n = 1, \cdots, N$ **under the same m**, we expect to decompose these N weight tensors jointly as follows:

$$\text{Unfold}(\boldsymbol{\mathcal{W}}_m^n) = \mathbf{W}_m^n \approx \mathbf{G}_m^n \Sigma_m^n \mathbf{V}_m = \mathbf{U}_m^n \mathbf{V}_m, \tag{1}$$

where $\mathbf{V}_m \in \mathbb{R}^{r_m^r \times F_2 O}$ is identical and shared for this group of $\boldsymbol{\mathcal{W}}_m^n, n = 1, \cdots, N$, while $\mathbf{U}_m^n \in \mathbb{R}^{F_1 I \times r_m^r} = \mathbf{G}_m^n \Sigma_m^n$ are different, and the r_m^r here is the truncated rank. In this manner, the sub-network is compressed by a factor of $\frac{F_1 F_2 I O N}{r_m^r (F_1 I N + F_2 O)} \times$.

The optimization problem for (1) can be formulated as:

$$\begin{aligned} &\min_{\mathbf{U}_m^n,\mathbf{V}_m} ||\mathbf{W}_m^n - \mathbf{U}_m^n\mathbf{V}_m||_2^2 \\ &s.t. \quad \text{rank}(\mathbf{U}_m^n\mathbf{V}_m) \leq r_m^r \\ &\quad \text{for} \quad n = 1, 2, \cdots, N. \end{aligned} \qquad (2)$$

Since it is difficult to optimize the rank, we take r_m^r as a given parameter, thus (2) can be rewritten as

$$\min_{\{\mathbf{U}_m^n\}_{n=1}^N,\mathbf{V}_m} \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{W}_m^n - \mathbf{U}_m^n\mathbf{V}_m||_2^2, \qquad (3)$$

which can be solved by randomly initializing $\mathbf{U}_m^n$ and $\mathbf{V}_m$ at first and then update them alternatively as follows:

(1) Given $\{\mathbf{U}_m^n\}_{n=1}^N$,
update $\mathbf{V}_m = \frac{1}{N}\sum_{n=1}^N(\mathbf{U}_m^{n\mathrm{T}}\mathbf{U}_m^n)^{-1}\mathbf{U}_m^{n\mathrm{T}}\mathbf{W}_m^n$.

(2) Given $\mathbf{V}_m$,
update $\mathbf{U}_m^n = \mathbf{W}_m^n\mathbf{V}_m^{\mathrm{T}}(\mathbf{V}_m\mathbf{V}_m^{\mathrm{T}})^{-1}$, for $n = 1, 2, \cdots, N$.

Alternatively, we could solve it by performing truncated rank-r_m^r SVD on the matrix produced by stacking $\mathbf{W}_m^n, n = 1, 2, \cdots, N$ vertically:

$$\begin{bmatrix} \mathbf{U}_m^1 \\ \mathbf{U}_m^2 \\ \cdots \\ \mathbf{U}_m^N \end{bmatrix}, \mathbf{V}_m = \text{SVD}_{r_m^r}(\begin{bmatrix} \mathbf{W}_m^1 \\ \mathbf{W}_m^2 \\ \cdots \\ \mathbf{W}_m^N \end{bmatrix}). \qquad (4)$$

The $\mathbf{V}_m$ here is the right singular matrix that is shared and identical for $\mathbf{W}_m^n, n = 1, 2, \cdots, N$, therefore, we refer to the algorithm proposed for compressing CNNs based on (4) as **Right Shared Joint SVD** (**RJSVD**). This algorithm can also accelerate the forward inference of CNNs, because the matrix decomposition shown above actually decomposes a single convolution layer into two successive ones with less complexity, whose weights are a "vertical" tensor $\boldsymbol{\mathcal{U}}_m^n = \text{fold}(\mathbf{U}_m^n) \in \mathbb{R}^{F_1\times 1\times I\times r_m^r}$ and a "horizontal" one $\boldsymbol{\mathcal{V}}_m = \text{fold}(\mathbf{V}_m) \in \mathbb{R}^{1\times F_2\times r_m^r\times O}$, respectively, as shown in Figure 2 and proved as follows:

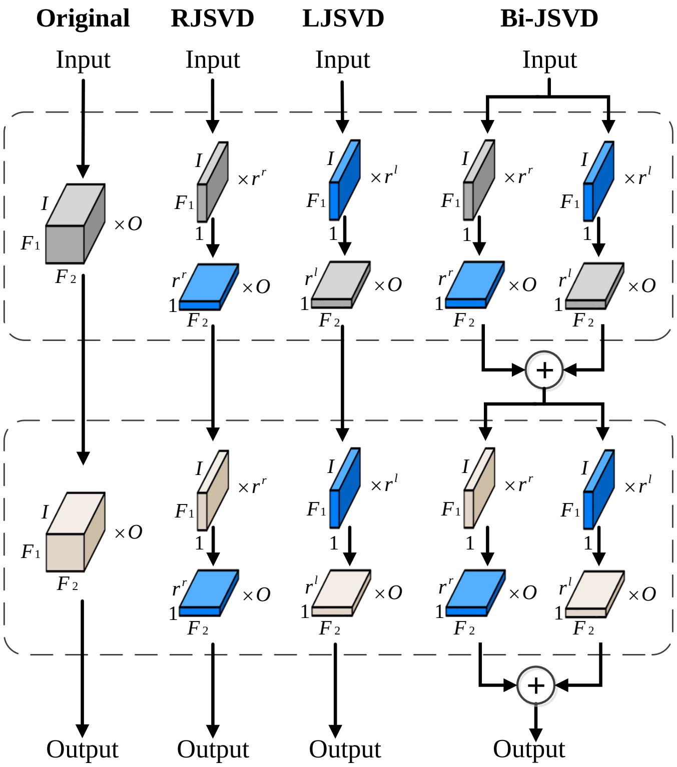

Fig. 2: Network structures of the original and the compressed networks with $N = 2$. RJSVD and LJSVD divide each convolution layer into two consecutive slimmer ones jointly with one half shared (the blue tensors), and Bi-JSVD can be seen as a combination of RJSVD and LJSVD.

Proof. Here we temporarily ignore superscripts and subscripts for simplicity. Suppose that there is a convolution layer with an input $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{F_1\times F_2\times I\times O}$ and a weight tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{F_1\times F_2\times I\times O}$ whose general unfolding matrix $\mathbf{W}$ can be decomposed into 2 matrices $\mathbf{U} \in \mathbb{R}^{F_1 I\times r}$ and $\mathbf{V} \in \mathbb{R}^{r\times F_2 O}$, then the convolutional output under settings of $[1, 1]$ strides and "SAME" padding is $\boldsymbol{\mathcal{Y}} = \text{Conv}(\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}) \in \mathbb{R}^{1\times 1\times O}$. Defining $\text{vec}(\cdot)$ as vectorization operator, $\mathbf{X} = \text{unfold}(\boldsymbol{\mathcal{X}}) \in \mathbb{R}^{F_1 I\times F_2 O}$ and $\mathbf{W}_{(:,i,F_2)} = \mathbf{W}[:, F_2 * (i-1) : F_2 * i]$, *i.e.*, the i-th slice along the second axe with a length of F_2, we have

$$\begin{aligned} \text{vec}(\boldsymbol{\mathcal{Y}}) &= \text{vec}(\text{Conv}(\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}})) \\ &= \begin{bmatrix} \text{vec}(\mathbf{W}_{(:,1,F_2)})^{\mathrm{T}} \\ \text{vec}(\mathbf{W}_{(:,2,F_2)})^{\mathrm{T}} \\ \cdots \\ \text{vec}(\mathbf{W}_{(:,O,F_2)})^{\mathrm{T}} \end{bmatrix} \text{vec}(\mathbf{X}) \\ &= \begin{bmatrix} \text{vec}(\mathbf{UV}_{(:,1,F_2)})^{\mathrm{T}} \\ \text{vec}(\mathbf{UV}_{(:,2,F_2)})^{\mathrm{T}} \\ \cdots \\ \text{vec}(\mathbf{UV}_{(:,O,F_2)})^{\mathrm{T}} \end{bmatrix} \text{vec}(\mathbf{X}) \\ &= \begin{bmatrix} \text{vec}(\mathbf{V}_{(:,1,F_2)})^{\mathrm{T}} \\ \text{vec}(\mathbf{V}_{(:,2,F_2)})^{\mathrm{T}} \\ \cdots \\ \text{vec}(\mathbf{V}_{(:,O,F_2)})^{\mathrm{T}} \end{bmatrix} \text{vec}(\mathbf{U}^{\mathrm{T}}\mathbf{X}) \\ &= \begin{bmatrix} \text{vec}(\mathbf{V}_{(:,1,F_2)})^{\mathrm{T}} \\ \text{vec}(\mathbf{V}_{(:,2,F_2)})^{\mathrm{T}} \\ \cdots \\ \text{vec}(\mathbf{V}_{(:,O,F_2)})^{\mathrm{T}} \end{bmatrix} \text{vec}(\text{Conv}(\boldsymbol{\mathcal{U}}, \boldsymbol{\mathcal{X}})) \\ &= \text{vec}(\text{Conv}(\boldsymbol{\mathcal{V}}, \text{Conv}(\boldsymbol{\mathcal{U}}, \boldsymbol{\mathcal{X}}))). \end{aligned} \qquad (5)$$

Therefore, $\text{Conv}(\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}) = \text{Conv}(\boldsymbol{\mathcal{V}}, \text{Conv}(\boldsymbol{\mathcal{U}}, \boldsymbol{\mathcal{X}}))$. □

A more complicated case with different sizes of $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{H\times W\times I}$ and strides $[s, s]$ can be proven in a similar way, thus in the compressed CNN, it is not necessary to reconstruct $\boldsymbol{\mathcal{W}}$ as in [16][42] in the forward inference. Instead, we only need to replace the original convolution layer with two consecutive factorized ones, where the first has the weight tensor $\boldsymbol{\mathcal{U}} \in \mathbb{R}^{F_1 \times 1 \times I \times r}$ with strides $[s, 1]$ and the second one $\boldsymbol{\mathcal{V}} \in \mathbb{R}^{1 \times F_2 \times r \times O}$ with strides $[1, s]$. In this manner, CNNs are not only compressed but also accelerated in the inference period, since the number of FLOPs in the convolutional layer drops from $H'W'F_1F_2IO$ to $H'WF_1Ir_m^r + H'W'F_2r_m^rO$, where $H' \times W'$ are the spatial sizes of the convolutional output.

After decomposition, fine-tuning can be performed to recover performance. Although the fine-tuning of the compressed CNNs would consume some extra time, it is acceptable since the process can be done quickly in high-performance servers with multiple GPUs. In contrast, we pay more attention to the models' size, inference time, and accuracy. The whole process of RJSVD is shown in Algorithm 1.

The compression factor (**CF**) in this algorithm can be defined now. Denoting the number of uncompressed parameters such as those in the BN layers and uncompressed layers as $\#$Other where $\#$ means sizes of parameters, the compression factor can be calculated by

$$\text{CF} = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N} \#\boldsymbol{\mathcal{W}}_m^n + \#\text{Other}}{\sum_{m=1}^{M} (\#\boldsymbol{\mathcal{V}}_m + \sum_{n=1}^{N} \#\boldsymbol{\mathcal{U}}_m^n) + \#\text{Other}}. \tag{6}$$

Algorithm 1: RJSVD

```
Input: A pre-trained CNN with weight tensors
       ${\{\boldsymbol{\mathcal{W}}_m^n\}_{n=1}^N\}_{m=1}^M$, and target ranks $\{r_m^r\}_{m=1}^M$.
for $m = 1, 2, \cdots, M$ do
    Obtain $\{\mathbf{U}_m^n\}_{n=1}^N, \mathbf{V}_m$ according to formula (4).
    $\boldsymbol{\mathcal{V}}_m = \text{fold}(\mathbf{V}_m)$.
    for $n = 1, 2, \cdots, N$ do
        $\boldsymbol{\mathcal{U}}_m^n = \text{fold}(\mathbf{U}_m^n)$.
        Replace the original convolutional layer $\boldsymbol{\mathcal{W}}_m^n$
          with two compressed ones $\boldsymbol{\mathcal{U}}_m^n$ and $\boldsymbol{\mathcal{V}}_m$.
    end
end
Fine-tune the compressed CNN.
Output: The compressed CNN and its weights
        $\boldsymbol{\mathcal{U}}_m^n, \boldsymbol{\mathcal{V}}_m$.
```

2) Left Shared Joint SVD (LJSVD): In the RJSVD, the right singular matrix is shared to further compress CNNs, which enables weight typing across layers. Following the same idea, it is natural to derive **Left Shared Joint SVD** (**LJSVD**) by sharing the left factorized matrix, which is:

$$\text{Unfold}(\boldsymbol{\mathcal{W}}_m^n) = \mathbf{W}_m^n \approx \mathbf{U}_m \mathbf{V}_m^n, \tag{7}$$

where $\mathbf{U}_m \in \mathbb{R}^{F_1 I \times r_m^l}$ are identical and shared for a group of $\boldsymbol{\mathcal{W}}_m^n, n = 1, \cdots, N$, while $\mathbf{V}_m^n \in \mathbb{R}^{r_m^l \times F_2 O}$ is different. The decomposed structure derived by LJSVD is shown in the Figure 2.

Similar to (3), the optimization problem of LJSVD can be formulated as:

$$\min_{\{\mathbf{U}_m^n\}_{n=1}^N, \mathbf{V}_m} \frac{1}{N} \sum_{n=1}^{N} ||\mathbf{W}_{n,m} - \mathbf{U}_m \mathbf{V}_m^n||_2^2, \tag{8}$$

which can be solved by performing truncated rank-r_m^l SVD on the matrix produced by stacking $\mathbf{W}_m^n, n = 1, 2, \cdots, N$ horizontally:

$$\mathbf{U}^m, [\mathbf{V}_m^1, \mathbf{V}_m^2, \cdots, \mathbf{V}_m^N] = \text{SVD}_{r_m^l}([\mathbf{W}_m^1, \mathbf{W}_m^2, \cdots, \mathbf{W}_m^N]). \tag{9}$$

The concrete steps are shown in the Algorithm 2. With this algorithm, the FLOPs of a convolutional layer will drop from $H'W'F_1F_2IO$ to $H'WF_1Ir_m^l + H'W'F_2r_m^lO$. And similar to (6), the CF for LJSVD is

$$\text{CF} = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N} \#\boldsymbol{\mathcal{W}}_m^n + \#\text{Other}}{\sum_{m=1}^{M} (\#\boldsymbol{\mathcal{U}}_m + \sum_{n=1}^{N} \#\boldsymbol{\mathcal{V}}_m^n) + \#\text{Other}}. \tag{10}$$

Algorithm 2: LJSVD

```
Input: A pre-trained CNN with weight tensors
       ${\{\boldsymbol{\mathcal{W}}_m^n\}_{n=1}^N\}_{m=1}^M$, and target ranks $\{r_m^l\}_{m=1}^M$.
for $m = 1, 2, \cdots, M$ do
    Obtain $\mathbf{U}_m, \{\mathbf{V}_m^n\}_{n=1}^N$ according to formula (9).
    $\boldsymbol{\mathcal{U}}_m = \text{fold}(\mathbf{V}_m)$.
    for $n = 1, 2, \cdots, N$ do
        $\boldsymbol{\mathcal{V}}_m^n = \text{fold}(\mathbf{V}_m^n)$.
        Replace the original convolutional layer $\boldsymbol{\mathcal{W}}_m^n$
          with two compressed ones $\boldsymbol{\mathcal{U}}_m$ and $\boldsymbol{\mathcal{V}}_m^n$.
    end
end
Fine-tune the compressed CNN.
Output: The compressed CNN and its weights
        $\boldsymbol{\mathcal{U}}_m, \boldsymbol{\mathcal{V}}_m^n$.
```

3) Binary Joint SVD (Bi-JSVD): The RJSVD and LJSVD jointly project the weight tensors of repeated layers into the same subspace by sharing the right factorized matrices or the left ones, and in this section, we combine RJSVD and LJSVD together to generate **Binary Joint SVD** (**Bi-JSVD**), which is

$$\text{Unfold}(\boldsymbol{\mathcal{W}}_m^n) = \mathbf{W}_m^n \approx \mathbf{U}_m^n \mathbf{V}_m + \mathbf{U}_m \mathbf{V}_m^n, \tag{11}$$

where $\mathbf{U}_m^n \in \mathbb{R}^{F_1 I \times r_m^r}, \mathbf{V}_m \in \mathbb{R}^{r_m^r \times F_2 O}, \mathbf{U}_m \in \mathbb{R}^{F_1 I \times r_m^l}, \mathbf{V}_m^n \in \mathbb{R}^{r_m^l \times F_2 O}$.

Note that Bi-JSVD is the generalization of RJSVD and LJSVD. In other words, RJSVD and LJSVD are particular cases of Bi-JSDV with $r_m^l = 0$ and $r_m^r = 0$, respectively. For convenience, we define the proportion of LJSVD in Bi-JSVD as

$$\boldsymbol{p} = \frac{r_m^l}{r_m^l + r_m^r}. \tag{12}$$

The optimization problem for (11) can be formulated as:

$$\min_{\{\mathbf{U}_m^n, \mathbf{V}_m^n\}_{n=1}^N, \mathbf{U}_m, \mathbf{V}_m} \frac{1}{N} \sum_{n=1}^{N} ||\mathbf{W}_{n,m} - \mathbf{U}_m^n \mathbf{V}_m - \mathbf{U}_m \mathbf{V}_m^n||_2^2. \tag{13}$$

Since it is inefficient to solve (13) with simultaneous optimization on $\{\mathbf{U}_m^n\}_{n=1}^N, \mathbf{V}_m$, and $\mathbf{U}_m, \{\mathbf{V}_m^n\}_{n=1}^N$, we solve it alternatively and iteratively as follows:

(1) Given $\mathbf{U}_m, \{\mathbf{V}_m^n\}_{n=1}^N$, let

$$\widehat{\mathbf{W}_m^n} = \mathbf{W}_m^n - \mathbf{U}_m \mathbf{V}_m^n, \text{ for } n = 1, \cdots, N, \tag{14}$$

then we have

$$\begin{bmatrix} \mathbf{U}_m^1 \\ \mathbf{U}_m^2 \\ \cdots \\ \mathbf{U}_m^N \end{bmatrix}, \mathbf{V}_m = \text{SVD}_{r_m^r}\left(\begin{bmatrix} \widehat{\mathbf{W}_m^1} \\ \widehat{\mathbf{W}_m^2} \\ \cdots \\ \widehat{\mathbf{W}_m^N} \end{bmatrix} \right). \tag{15}$$

(2) Given $\{\mathbf{U}_m^n\}_{n=1}^N, \mathbf{V}_m$, let

$$\widehat{\widehat{\mathbf{W}_m^n}} = \mathbf{W}_m^n - \mathbf{U}_m^n \mathbf{V}_m, \text{ for } n = 1, \cdots, N, \tag{16}$$

then we have

$$\mathbf{U}_m, \left[\mathbf{V}_m^1, \mathbf{V}_m^2, \cdots, \mathbf{V}_m^N\right] = \text{SVD}_{r_m^l}\left(\left[\widehat{\widehat{\mathbf{W}_m^1}}, \widehat{\widehat{\mathbf{W}_m^2}}, \cdots, \widehat{\widehat{\mathbf{W}_m^N}} \right] \right). \tag{17}$$

The decomposition will be converged by repeating the above steps K times, and then the fine-tuning of the compressed network can be used to recover its performance, which is summarized in Algorithm 3. The CF for Bi-JSVD is

$$\text{CF} = \frac{\sum_{m=1}^M \sum_{n=1}^N \#\boldsymbol{\mathcal{W}}_m^n + \#\text{Other}}{\sum_{m=1}^M (\#\boldsymbol{\mathcal{V}}_m + \#\boldsymbol{\mathcal{U}}_m + \sum_{n=1}^N (\#\boldsymbol{\mathcal{U}}_m^n + \#\boldsymbol{\mathcal{V}}_m^n)) + \#\text{Other}}, \tag{18}$$

and the complexity of a compressed convolutional layer is $\mathcal{O}((H'WF_1I + H'W'F_2O)(r_m^l + r_m^r))$.

Algorithm 3: Bi-JSVD

```
Input: A pre-trained CNN with weight tensors
       $\{\{\boldsymbol{\mathcal{W}}_m^n\}_{n=1}^N\}_{m=1}^M$, target ranks $\{r_m^r, r_m^l\}_{m=1}^M$,
       and iteration times $K$.
for $m = 1, 2, \cdots, M$ do
    Initialization: $\mathbf{U}_m = 0, \quad \{\mathbf{V}_m^n\}_{n=1}^N = 0$.
    for $(k = 0; k < K; k{+}{+})$ do
        Update $\{\mathbf{U}_m^n\}_{n=1}^N, \mathbf{V}_m$ according to (14) (15).
        Update $\mathbf{U}_m, \{\mathbf{V}_m^n\}_{n=1}^N$ according to (16) (17).
    end
    $\boldsymbol{\mathcal{U}}_m = \text{fold}(\mathbf{U}_m)$.
    $\boldsymbol{\mathcal{V}}_m = \text{fold}(\mathbf{V}_m)$.
    for $n = 1, 2, \cdots, N$ do
        $\boldsymbol{\mathcal{U}}_m^n = \text{fold}(\mathbf{U}_m^n)$.
        $\boldsymbol{\mathcal{V}}_m^n = \text{fold}(\mathbf{V}_m^n)$.
        Replace the original convolutional layer $\boldsymbol{\mathcal{W}}_m^n$
          with two parallel compressed sub-networks
          $\boldsymbol{\mathcal{U}}_m^n, \boldsymbol{\mathcal{V}}_m$ and $\boldsymbol{\mathcal{U}}_m, \boldsymbol{\mathcal{V}}_m^n$ as shown in the Figure
          2.
    end
end
Fine-tune the compressed CNN.
Output: The compressed CNN and its weights
        $\boldsymbol{\mathcal{U}}_m, \boldsymbol{\mathcal{V}}_m^n, \boldsymbol{\mathcal{U}}_m^n, \boldsymbol{\mathcal{V}}_m$.
```

IV. Experiments

To validate the effectiveness of the proposed algorithms, we conduct extensive experiments on various benchmark data sets and several widely used networks with different depths. The proposed algorithms are adopted to compress the networks, and we compare the results with some state-of-the-art decomposition-based compression methods. All experiments are performed on one TITAN Xp GPU under Tensorflow1.15 and Ubuntu18.04.[1]

A. Evaluation on CIFAR-10 and CIFAR-100

1) Overall settings:

a) Datasets: In this section, we evaluate the proposed methods on CIFAR-10 and CIFAR-100 [43]. Both data sets have 60,000 $32 \times 32 \times 3$ images, including 50,000 training images and 10,000 testing images. The former contains 10 classes, while the latter includes 100 categories, and thus is more challenging for classification. For data preprocessing, all the images are normalized with $mean = [0.4914, 0.4822, 0.4465]$ and standard deviation $std = [0.2023, 0.1994, 0.2010]$. For data augmentation, a 32×32 random crop is adopted on the zero-padded 40×40 training images followed by a random horizontal flip.

b) Networks: We evaluate the proposed method on the widely used ResNet with various depths *i.e.*, ResNet-18, ResNet-34, ResNet-50. There are two main reasons for choosing them: (1) the widely used ResNet is a typical representative of architectures that adopt the repeated module design; (2) by compressing these compact networks, the ability of the proposed algorithms can be clearly presented [26]. The architectures and Top-1 accuracy of the baseline CNNs are shown in Tables I and II, respectively. We randomly initialize the weights using the truncated normal initializer by setting the standard deviation to 0.01. The SGD optimizer with a momentum of 0.9 and a weight decay of 5e-4 is used to train the CNNs for 300 epochs. The batch size for ResNet-18 and ResNet-34 is 256, while it is 128 for ResNet-50 due to the limitation of graphics memory. The learning rate starts from 0.1 and is divided by 10 in the 140th, 200th, and 250th epochs.

c) Methods for comparison: To evaluate the effectiveness of the proposed algorithms, three state-of-the-art methods, including one matrix-decomposition-based method, Tai *et al.* [13], and two tensor-decomposition-based ones, Tucker [8] and NC_CTD [20], are employed for comparison.

d) CF and FLOPs: When setting CFs, we take [13] as the anchor. An equal proportion is set for each layer to determine the ranks for [13] and obtain the CF, with which the ranks for Tucker [8], NC_CTD [20] and our methods are then calculated under the same CFs. In NC_CTD [20], since the proportion of the shared component and independent component affects the final performance, without loss of generality, we set it to 1 : 1 and 2 : 1 and report the better result only. We use the "TensorFlow.profiler" function to calculate the FLOPs which indicates the theoretical acceleration of the compressed models.

[1] The codes are available at https://github.com/ShaowuChen/JointSVD.

layer name	output size	ResNet-18	ResNet-34	ResNet-50
conv1 (original)	32×32	3×3, 64, stride 1		
conv2_x (original)	32×32	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x (decom)	16×16	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4_x (decom)	8×8	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
conv5_x (decom)	4×4	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
average pool, 10-d fc for CIFAR-10/100-d fc for CIFAR-100, softmax				

TABLE I: The architectures of CNNs for CIFAR-10 and CIFAR-100. Layers remarked with "original" will not be decomposed in the following experiments, while those remarked with "decom" will be decomposed.

	CIFAR-10		CIFAR-100	
CNN	Acc. (%)	Size (M)	Acc. (%)	Size (M)
ResNet-18	94.80	11.16	70.73	11.21
ResNet-34	95.11	21.27	75.81	21.31
ResNet-50	95.02	23.50	75.67	23.69

TABLE II: Top-1 accuracies and parameter sizes of the baseline CNNs on CIFAR-10 and CIFAR-100. "Acc." means "Accuracy".

2) Tuning the Parameter K: In the beginning, we first determine the iteration number K for Bi-JSVD since it might affect the quality of the initialization after decomposition and before fine-tuning. Empirically, a more accurate approximation in decomposition would bring better performance after fine-tuning, making K an inconspicuous but vital hyper-parameter. A large K, such as 100, can ensure the convergence of decomposition, but it would consume more time. To find a suitable K, we conduct experiments on ResNet-34 for CIFAR-10, and take the raw accuracy of the decomposed networks before fine-tuning as the evaluation criterion.

a) Settings: We decompose the sub-networks "conv$i$_x" for $i = 3, 4, 5$ in Table I except for the conv1 and conv2_x since they have much fewer parameters comparing with the other sub-networks, and decomposing them would bring relatively severe accumulated error. In each sub-networks "conv$i$_x" for $i = 3, 4, 5$, the first convolutional layer of the first block has only half the input depth (**HID layer**) compared with the one in other blocks, thus they are not decomposed by Bi-JSVD jointly but the matrix method [13] separately. The CF and iteration times are set to 13.9 and $K \in \{10, 30, 50, 70\}$, respectively, and we set $\boldsymbol{p}$, the proportion of LJSVD, to $p = 0.3, 0.5, 0.7, 0.9$.

b) Results and analysis: The results are shown in Table III. It is shown that $K = 30$ and $K = 70$ would bring relatively higher raw accuracy, and the gaps between them are insignificant. Therefore, K will be set to 30 in all the following experiments to obtain high raw accuracy with less complexity. Furthermore, it seems that the proportion of LJSVD or RJSVD in Bi-JSVD indicated by p would affect the raw accuracy of the compressed network. To give a conclusion on how p affects the final performance, more experiments are needed, and we will discover it in the following sections.

	p			
K	0.3	0.5	0.7	0.9
10	57.64%	53.44%	51.04%	42.69%
30	**58.12%**	56.43%	**51.75%**	44.01%
50	57.33%	56.77%	50.23%	43.45%
70	57.65%	**56.85%**	49.47%	**44.13%**

TABLE III: Raw accuracies of compressed ResNet-34 with Bi-JSVD on CIFAR-10 before fine-tuning.

3) Performance Comparison:

a) Settings: We do not decompose conv1 and conv2_x in all CNNs as it is discussed in IV-A2. In ResNet-18 and ResNet-34, we compare the proposed methods with Tai *et al.* [13], Tucker [8], and NC_CTD [20] following the "pre-train→decompose→fine-tune" pipeline. Whereas, in ResNet-50, there are lots of 1×1 convolutional layers with weight tensors of dimensions $1 \times 1 \times I \times O$ that make the Tucker or tensor decomposition equal to matrix decomposition, thus only the Tai *et al.* [13] is used for comparing. To see the gap of performance clearly, the CF in ResNet-18 and ResNet-34 is set to be relatively large and roughly from 14 to 20. However, we set the CF for ResNet-50 as 5–6 since ResNet-50 with BottleNecks is very compact.

The HID layers with half the input depth are also decomposed for all the methods. For consistency, as in the Section IV-A2, the HID layers in LJSVD and Bi-JSVD will be decomposed via [13] separately, since LJSVD and Bi-JSVD are not compatible with the HID layers. However, RJSVD stacks folded weights vertically, thus the HID layers can be included for decomposing jointly. To evaluate RJSVD comprehensively and fairly, we implement **RJSVD-1** and **RJSVD-2** that decompose the HID layers jointly and separately, respectively. To answer the query posted in the last experiment, *i.e.*, how p in Bi-JSVD affects the performance of the compress networks, we set different p and name the

CNN & acc. & FLOPs	CF (×)	Method	Raw acc. (%)	Acc. (%) (mean±std)	Best acc. (%)	FLOPs
ResNet-18 94.80% 11.11E8	17.76	Tai *et al.* [13]	10.31	92.49±0.23	92.93	3.42E8
		Tucker [8]	**23.77**	91.93±0.13	92.41	3.41E8
		NC_CTD [20]	23.51	92.20±0.29	92.80	3.47E8
		LJSVD	11.44	92.88±0.08	93.00	3.45E8
		RJSVD-1	10.00	**93.19±0.04**	**93.36**	3.50E8
		RJSVD-2	10.27	92.75±0.09	93.09	3.45E8
		Bi-JSVD0.3	13.76	92.81±0.06	92.91	3.45E8
		Bi-JSVD0.5	13.09	92.70±0.11	92.80	3.44E8
		Bi-JSVD0.7	13.64	93.11±0.06	93.32	3.45E8
	11.99	Tai *et al.* [13]	54.42	93.55±0.08	93.83	3.65E8
		Tucker [8]	**57.51**	92.42±0.29	92.84	3.63E8
		NC_CTD [20]	54.15	92.85±0.25	93.34	3.74E8
		LJSVD	50.15	93.62±0.10	93.98	3.73E8
		RJSVD-1	30.53	93.75±0.07	**94.22**	3.83E8
		RJSVD-2	42.43	93.47±0.12	93.80	3.73E8
		Bi-JSVD0.3	52.84	93.77±0.17	94.15	3.73E8
		Bi-JSVD0.5	48.45	93.49±0.07	93.73	3.72E8
		Bi-JSVD0.7	52.31	**93.84±0.09**	94.16	3.73E8
ResNet-34 95.11% 23.19E8	22.07	Tai *et al.* [13]	12.93	93.66±0.13	93.98	5.20E8
		Tucker [8]	**19.73**	92.83±0.07	93.06	5.20E8
		NC_CTD [20]	19.12	92.98±0.15	93.35	5.71E8
		LJSVD	15.96	**93.97±0.10**	94.07	5.47E8
		RJSVD-1	10.21	93.82±0.07	94.00	5.54E8
		RJSVD-2	12.97	93.77±0.03	93.95	5.47E8
		Bi-JSVD0.3	17.38	93.66±0.05	93.88	5.44E8
		Bi-JSVD0.5	16.87	93.69±0.09	93.98	5.43E8
		Bi-JSVD0.7	15.83	93.77±0.13	**94.09**	5.44E8
	13.92	Tai *et al.* [13]	49.34	93.95±0.05	94.12	5.71E8
		Tucker [8]	**64.13**	93.04±0.21	93.39	5.70E8
		NC_CTD [20]	60.12	93.30±0.13	93.69	6.16E8
		LJSVD	39.53	**94.39±0.15**	94.61	6.27E8
		RJSVD-1	46.38	94.23±0.23	**94.73**	6.42E8
		RJSVD-2	45.60	93.94±0.06	94.12	6.27E8
		Bi-JSVD0.3	58.12	94.17±0.09	94.41	6.22E8
		Bi-JSVD0.5	56.43	94.08±0.06	94.32	6.24E8
		Bi-JSVD0.7	51.75	94.02±0.19	94.34	6.22E8
ResNet-50 95.02% 25.96E8	6.48	Tai *et al.* [13]	20.24	92.38±0.31	92.96	7.19E8
		LJSVD	12.07	93.08±0.17	93.37	7.63E8
		RJSVD-1	13.85	**93.28±0.09**	**93.58**	7.77E8
		RJSVD-2	**20.48**	92.75±0.28	93.26	7.73E8
		Bi-JSVD0.3	16.11	92.97±0.12	93.30	7.66E8
		Bi-JSVD0.5	13.46	92.90±0.15	93.19	7.66E8
		Bi-JSVD0.7	10.73	92.84±0.26	93.49	7.61E8
	5.37	Tai *et al.* [13]	28.40	92.99±0.42	93.44	7.97E8
		LJSVD	22.67	**93.39±0.18**	**93.81**	8.87E8
		RJSVD-1	24.94	92.97±0.10	93.33	9.17E8
		RJSVD-2	34.46	93.24±0.11	93.41	9.11E8
		Bi-JSVD0.3	32.18	92.86±0.19	93.17	8.99E8
		Bi-JSVD0.5	32.39	93.08±0.21	93.50	8.95E8
		Bi-JSVD0.7	**35.34**	93.06±0.09	93.28	8.89E8

TABLE IV: Comparison of compressed ResNet following the "pre-train→decompose→fine-tune" pipeline on CIFAR-10. "Raw acc." means accuracy before fine-tuning. "Acc." and "Best acc." represent the average accuracy and best accuracy after fine-tuning among repeated experiments, respectively.

CNN & acc. & FLOPs	CF (×)	Method	Raw acc. (%)	Acc. (%) (mean±std)	Best acc. (%)	FLOPs
ResNet-18 70.73% 11.11E8	16.61	Tai *et al.* [13]	2.07	71.15±0.16	71.86	3.42E8
		Tucker [8]	**7.69**	71.54±0.26	72.62	3.41E8
		NC_CTD [20]	3.94	**72.02±0.13**	72.74	3.47E8
		LJSVD	2.43	71.63±0.29	72.36	3.45E8
		RJSVD-1	1.67	**72.02±0.30**	**72.87**	3.50E8
		RJSVD-2	2.97	71.35±0.19	71.78	3.45E8
		Bi-JSVD0.3	3.78	71.56±0.13	72.04	3.45E8
		Bi-JSVD0.5	4.14	71.93±0.21	72.55	3.44E8
		Bi-JSVD0.7	3.95	71.61±0.11	72.28	3.45E8
	11.47	Tai *et al.* [13]	9.22	73.63±0.17	74.42	3.65E8
		Tucker [8]	**26.33**	72.24±0.17	72.78	3.64E8
		NC_CTD [20]	22.02	72.88±0.11	73.46	3.68E8
		LJSVD	6.42	**74.16±0.33**	**75.02**	3.73E8
		RJSVD-1	5.95	74.00±0.10	74.27	3.83E8
		RJSVD-2	8.88	73.71±0.14	74.17	3.73E8
		Bi-JSVD0.3	7.79	74.00±0.13	74.43	3.73E8
		Bi-JSVD0.5	8.31	73.71±0.17	73.93	3.72E8
		Bi-JSVD0.7	7.77	73.76±0.15	74.21	3.73E8
ResNet-34 75.81% 23.19E8	21.11	Tai *et al.* [13]	2.30	73.30±0.15	73.56	5.20E8
		Tucker [8]	**5.66**	73.53±0.14	73.81	5.20E8
		NC_CTD [20]	3.77	73.91±0.27	74.59	5.71E8
		LJSVD	4.75	**74.39±0.02**	**74.92**	5.47E8
		RJSVD-1	3.45	73.96±0.16	74.31	5.54E8
		RJSVD-2	4.33	73.81±0.13	74.33	5.47E8
		Bi-JSVD0.3	3.29	73.88±0.15	74.38	5.44E8
		Bi-JSVD0.5	5.12	74.03±0.12	74.55	5.43E8
		Bi-JSVD0.7	5.19	74.05±0.31	74.62	5.44E8
	13.55	Tai *et al.* [13]	6.35	75.29±0.23	75.50	5.71E8
		Tucker [8]	**19.69**	74.07±0.44	75.18	5.70E8
		NC_CTD [20]	17.51	74.53±0.12	75.18	6.67E8
		LJSVD	10.66	**75.84±0.16**	76.26	6.27E8
		RJSVD-1	6.09	75.43±0.07	75.76	6.42E8
		RJSVD-2	10.83	75.34±0.37	75.90	6.27E8
		Bi-JSVD0.3	13.52	**75.84±0.21**	**76.48**	6.22E8
		Bi-JSVD0.5	13.47	75.61±0.11	75.85	6.25E8
		Bi-JSVD0.7	10.73	**75.84±0.37**	76.47	6.22E8
ResNet-50 75.67% 25.96E8	6.21	Tai *et al.* [13]	7.65	74.36±0.23	74.90	7.20E8
		LJSVD	3.71	75.04±0.32	75.63	7.63E8
		RJSVD-1	2.18	73.56±0.41	74.52	7.78E8
		RJSVD-2	2.66	73.51±0.55	74.59	7.74E8
		Bi-JSVD0.3	3.00	74.79±0.14	75.14	7.66E8
		Bi-JSVD0.5	2.65	74.25±0.27	74.65	7.66E8
		Bi-JSVD0.7	3.34	74.47±0.09	74.92	7.61E8
	5.19	Tai *et al.* [13]	**24.67**	75.09±0.31	75.71	7.97E8
		LJSVD	5.57	**75.67±0.45**	**76.64**	8.88E8
		RJSVD-1	4.72	74.43±0.46	75.49	9.17E8
		RJSVD-2	8.02	74.36±0.33	74.81	9.11E8
		Bi-JSVD0.3	6.71	75.30±0.41	76.06	9.00E8
		Bi-JSVD0.5	5.76	74.81±0.62	75.81	8.95E8
		Bi-JSVD0.7	5.06	75.01±0.30	75.93	8.90E8

TABLE V: Comparison of compressed ResNet following the "pre-train→decompose→fine-tune" pipeline on CIFAR-100.

methods as **Bi-JSVD*p***, for $p = 0.3, 0.5, 0.7$. Note that here RJSVD-2, LJSVD are equivalent to Bi-JSVD0 and Bi-JSVD1, respectively.

To conduct a fair comparison and avoid early convergence, we use the same training settings in the fine-tuning stage as the baseline CNNs in Section IV-A1 for all the methods (including the methods for comparison) without particular tuning and tricks, except that the batch size for all networks is set to 128 for uniformity. There are 12 groups of experiments implemented in total, and all the experiments are repeated three times to obtain average performance. We also present the highest accuracy recorded during fine-tuning, since they indicate the potential of the corresponding models.

b) Results: The results are shown in Tables IV and V. To observe how p affects Bi-JSVD, we plot the curves of the average and best performance with $p = [0, 0.3, 0.5, 0.7, 1]$ under different CF in Figures 3 and 4.

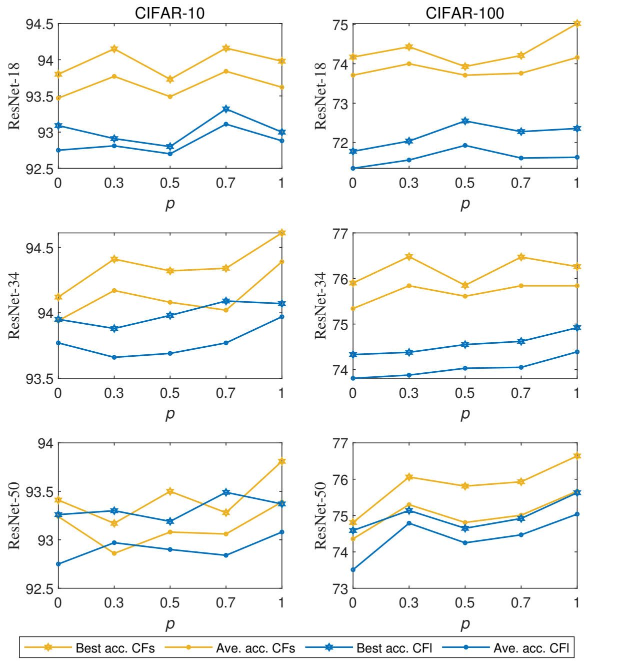

Fig. 3: The performance of Bi-JSVD for pre-trained CNNs under different p. "Ave. acc.", "Best acc.", "CFs" and "CFl" mean Average accuracy, Best accuracy, the smaller CF and the larger one for each CNN shown in the TABLE IV and V, respectively.

c) Analysis: From the results, we summarize and analyze several important observations:

(i) All the proposed methods outperform the compared methods after fine-tuning in most cases. CFs such as 22× much higher than previous works are achieved with relatively slight performance degradation, demonstrating the effectiveness of the proposed joint methods. Moreover, it is shown that a deeper network with more layers for joint decomposition would expand the advantages of the proposed methods in alleviating performance degradation. Besides, the advantages of our methods are further expanded in the more challenging CIFAR-100 tasks. For example, the accuracy of the ResNet-18 jointly decomposed by LJSVD on CIFAR-100 under CF = 11.47 is 1.92% higher than Tucker and 0.53% higher than Tai *et al.*. Furthermore, in the experiments on ResNet-34 for CIFAR-100 under CR=13.55, there is no significant loss in accuracy when our methods are applied.

(ii) Among the proposed methods, LJSVD is the most outstanding one that achieves the highest average accuracy in 8 out of 12 groups. The RJSVD-2 is inferior to LJSVD in most cases, indicating that the left shared structure is more powerful than the right shared one, since the only difference between the LJSVD and RJSVD-2 is the relative position of the shared components.

(iii) RJSVD-1 outperforms RJSVD-2 in most cases, and the reason behind the phenomenon is that RJSVD-1 includes the HID layers for joint decomposition while RJSVD-2 decomposes them separately. The more layers for joint decomposition, the better the performance, which is consistent with the observation i.

(iv) Compared with LJSVD and RJSVD, Bi-JSVDp, $p = 0.3, 0.5, 0.7$, can achieve not only the highest raw accuracy in most cases but also satisfying final performance. As shown in Figure 3, the performance of Bi-JSVDp increases when p gets close to 1. However, the tendency is unsteady, indicating that the optimal p is particular to the original networks and CF. For example, it is shown that $p = 1$ tends to achieve higher performance in the ResNet-50 or ResNet-34 for CIFAR-10 and CIFAR-100, while the optimal p in ResNet-18 for CIFAR-10 is 0.7. Therefore, there may not be a specific answer to the question left in the last experiment. However, empirically, one can take $p = 1$, *i.e.*, LJSVD, as the first choice unless the specific evaluating experiment is conducted.

(v) Tucker Decomposition and NC_CTD achieve the highest raw accuracies than other methods, but fail to achieve the best final performance in most cases, indicating that the final performance is not linearly correlated with the initial accuracy but decided by the structure of the decomposed network.

(vi) There is a particular phenomenon that the decomposed networks outperform the original in ResNet-18 for CIFAR-100. The reason for this is that decomposition followed by fine-tuning can alleviate overfitting in original networks and thus improve performance, which is consistent with [13].

B. Evaluation on ImageNet

a) Settings: To further verify the effectiveness of the proposed methods, we conduct comparison experiments on a large-scale data set, ImageNet (ILSVRC-12), which contains 1.2 million images from 1000 classes. For data pre-processing, all images are normalized with $mean = [0.485, 0.456, 0.406]$ and standard deviation $std = [0.229, 0.224, 0.225]$, and the test images are centrally cropped to 224 × 224. For data augmentation, training images are resized to 224 × 224 using four different methods provided by Tensorflow followed by a random horizontal flip.

The ResNet-34 [6] for ImageNet with 21.78 million of parameters is used for evaluation. To save time, the pre-trained weights are downloaded from Pytorch Model Zoo[2] and transferred to Tensorflow with several epochs of fine-tuning. The network's Top-1 and Top-5 accuracies are 71.03% and 90.25%, respectively.

The proposed LJSVD, RJSVD-1 and Bi-JSVD0.5 are utilized to compress the ResNet-34 following the "pre-train→decompose→fine-tune" pipeline, compared with Tai *et al.* [13], Tucker [13] and NC_CTD [20]. Following Section IV-A, K for Bi-JSVD is set to 30, and conv1, conv2_x in ResNet-34 are not decomposed. The proportion of the shared component and the independent one for NC_CTD is set to 1:1. In the fine-tuning period, the SGD is used as the optimizer with a momentum of 0.9. The batch size is set to 128, and

[2] https://download.pytorch.org/models/resnet34-333f7ec4.pth.

CNN & acc. & FLOPs	CF (×)	Method	Raw Top-1 acc.(%)	Raw Top-5 acc.(%)	Top-1 acc. (%)	Top-5 acc. (%)	FLOPs
ResNet-34 Top-1: 71.03% Top-5: 90.14% 7.33E9	10.98	Tai *et al.* [13]	0.15	0.87	60.17	83.27	1.82E9
		Tucker [8]	0.15	0.79	57.64	81.40	1.82E9
		NC_CTD [20]	**0.38**	**1.60**	58.37	82.29	1.98E9
		LJSVD	0.24	1.30	60.55	**83.68**	1.90E9
		RJSVD-1	0.24	1.11	60.74	83.65	1.92E9
		Bi-JSVD0.5	0.13	0.74	**60.83**	83.58	1.89E9
	5.75	Tai *et al.* [13]	0.24	0.97	62.52	84.88	1.98E9
		Tucker [8]	**0.84**	**3.09**	58.15	81.86	1.97E9
		NC_CTD [20]	0.39	1.67	61.12	84.26	2.29E9
		LJSVD	0.44	1.67	63.36	85.64	2.15E9
		RJSVD-1	0.32	1.51	63.43	85.74	2.19E9
		Bi-JSVD0.5	0.43	1.98	**64.12**	**86.16**	2.14E9

TABLE VI: Comparison of compressed ResNet-34 for ImageNet following the "pre-train→decompose→fine-tune" pipeline.

the weights are regularized by L2 with a weight decay of 5e-4. Since it is very time-consuming to fine-tune ResNet-34 for ImageNet, the warmup strategy [6][44] is applied to accelerate the fine-tuning process, with which the learning rates for the first 3 epochs are set to 0.0001, 0.001 and 0.01, respectively. After the warmup stage, the compressed networks are fine-tuned for 25 epochs, with the learning rate starting from 0.1 and divided by 10 in the 6th, 11th, 16th, and 21st epochs.

b) Results and analysis: Results are shown in TABLE VI. It is shown that for ImageNet classification, the proposed methods can still outperform the state-of-the-art methods and alleviate performance degradation. The Top-1 accuracy of all the proposed methods is more than 2% higher than NC_CTD [20] when CF=10.98, and Bi-JSVD0.5 achieves 0.52% higher Top-5 accuracy compared with Tai *et al.* [13] when CF is set to be 5.75, which further demonstrates the compatibility of the proposed algorithms and verify the effectiveness of the jointly decomposed methods for network compression.

C. Ablation Study

1) Train from scratch:

a) Settings: The comparison experiments demonstrate the superior performance of the proposed methods, and we consider that there are two main reasons contributing to their superiority: the joint structures, and prior knowledge inherited from the original networks. To verify this, we further conduct ablation experiments by training decomposed networks from scratch, which also evaluates the ability of the proposed method in compressing CNNs in another manner. We retain the structures of the compressed networks in Section IV-A3 and train them from scratch with randomly initialized parameters. Training settings such as the number of epochs, learning rate, and batch size are consistent with Section IV-A3.

b) Results: The results are shown in Tables VII and VIII, and Figure 4 shows how p affects the performance of compressed networks.

CNN & acc. & FLOPs	CF (×)	Method	Acc. (%) (mean±std)	Best acc. (%)	FLOPs
ResNet-18 94.80% 11.11E8	17.76	Tai *et al.* [13]	92.66±0.07	93.10	3.42E8
		Tucker [8]	90.83±0.13	91.33	3.41E8
		NC_CTD [20]	90.77±0.31	91.56	3.44E8
		LJSVD	92.80±0.17	**93.14**	3.45E8
		RJSVD-1	**92.83±0.13**	**93.14**	3.50E8
		RJSVD-2	92.74±0.13	92.97	3.45E8
		Bi-JSVD0.3	92.70±0.19	93.05	3.45E8
		Bi-JSVD0.5	92.78±0.11	93.11	3.44E8
		Bi-JSVD0.7	92.70±0.09	93.01	3.45E8
	11.99	Tai *et al.* [13]	92.73±0.13	92.94	3.65E8
		Tucker [8]	90.94±0.15	91.33	3.63E8
		NC_CTD [20]	91.22±0.13	91.72	3.68E8
		LJSVD	93.18±0.11	93.46	3.73E8
		RJSVD-1	**93.35±0.01**	**93.58**	3.83E8
		RJSVD-2	93.08±0.09	93.43	3.73E8
		Bi-JSVD0.3	93.34±0.13	**93.58**	3.73E8
		Bi-JSVD0.5	93.29±0.17	93.57	3.72E8
		Bi-JSVD0.7	93.13±0.09	93.48	3.73E8
ResNet-34 95.11% 23.19E8	22.07	Tai *et al.* [13]	92.90±0.03	93.11	5.20E8
		Tucker [8]	91.83±0.25	92.31	5.20E8
		NC_CTD [20]	91.03±0.13	91.53	5.43E8
		LJSVD	**93.73±0.09**	**94.10**	5.47E8
		RJSVD-1	93.59±0.14	94.00	5.54E8
		RJSVD-2	93.45±0.15	93.79	5.47E8
		Bi-JSVD0.3	93.65±0.13	93.92	5.44E8
		Bi-JSVD0.5	93.42±0.09	93.64	5.43E8
		Bi-JSVD0.7	93.34±0.15	93.74	5.44E8
	13.92	Tai *et al.* [13]	93.11±0.04	93.43	5.71E8
		Tucker [8]	91.78±0.15	92.20	5.70E8
		NC_CTD [20]	91.46±0.10	91.79	6.16E8
		LJSVD	**93.85±0.19**	**94.23**	6.27E8
		RJSVD-1	93.62±0.15	93.62	6.42E8
		RJSVD-2	93.49±0.14	93.73	6.27E8
		Bi-JSVD0.3	93.36±0.07	93.52	6.22E8
		Bi-JSVD0.5	93.60±0.09	93.94	6.24E8
		Bi-JSVD0.7	93.57±0.06	93.86	6.22E8
ResNet-50 95.02% 25.96E8	6.48	Tai *et al.* [13]	91.99±0.08	92.42	7.19E8
		LJSVD	**92.89±0.11**	**93.17**	7.63E8
		RJSVD-1	92.77±0.03	92.97	7.77E8
		RJSVD-2	92.46±0.15	92.80	7.73E8
		Bi-JSVD0.3	92.85±0.15	93.00	7.66E8
		Bi-JSVD0.5	92.83±0.18	93.13	7.66E8
		Bi-JSVD0.7	92.64±0.20	92.95	7.61E8
	5.37	Tai *et al.* [13]	92.22±0.19	92.55	7.97E8
		LJSVD	92.93±0.15	93.25	8.87E8
		RJSVD-1	93.00±0.02	93.37	9.17E8
		RJSVD-2	92.74±0.18	93.15	9.11E8
		Bi-JSVD0.3	92.83±0.07	93.06	8.99E8
		Bi-JSVD0.5	93.10±0.19	93.42	8.95E8
		Bi-JSVD0.7	**93.41±0.09**	**93.61**	8.89E8

TABLE VII: Comparison of compressed ResNet trained from scratch on CIFAR-10.

c) Analysis: Not surprisingly, the proposed methods outperform the baseline methods, demonstrating the power of the proposed joint structures in alleviating performance degradation, and conclusions similar to the previous subsection can still be summarized. However, there are some outcomes different from the previous results as follows:

(i) Compared with results in Table IV and V, the accuracy of compressed networks trained from scratch drops in different degrees, implying that factorized weights inheriting the prior knowledge from the original networks can give a more appropriate initialization to help CNNs settle at a better local minimum. Furthermore, the drops of the proposed methods are much smaller than the baseline approaches in most cases. For example, LJSVD

CNN & acc. & FLOPs	CF (×)	Method	Acc. (%) (mean±std)	Best acc. (%)	FLOPs
ResNet-18 70.73% 11.11E8	16.61	Tai *et al.* [13]	71.12±0.35	72.25	3.42E8
		Tucker [8]	69.67±0.23	69.88	3.41E8
		NC_CTD [20]	70.66±0.27	71.24	3.44E8
		LJSVD	71.35±0.19	71.92	3.45E8
		RJSVD-1	**71.59±0.39**	**72.48**	3.50E8
		RJSVD-2	71.33±0.16	72.08	3.45E8
		Bi-JSVD0.3	71.44±0.13	71.88	3.45E8
		Bi-JSVD0.5	71.12±0.19	71.82	3.44E8
		Bi-JSVD0.7	71.58±0.40	72.40	3.45E8
	11.47	Tai *et al.* [13]	71.63±0.25	72.26	3.65E8
		Tucker [8]	69.84±0.29	70.92	3.64E8
		NC_CTD [20]	70.72±0.06	71.43	3.68E8
		LJSVD	73.13±0.31	73.88	3.73E8
		RJSVD-1	**73.38±0.22**	**73.96**	3.83E8
		RJSVD-2	72.97±0.51	73.58	3.73E8
		Bi-JSVD0.3	73.11±0.23	73.88	3.73E8
		Bi-JSVD0.5	73.29±0.09	73.68	3.72E8
		Bi-JSVD0.7	73.04±0.11	73.37	3.73E8
ResNet-34 75.81% 23.19E8	21.11	Tai *et al.* [13]	72.59±0.49	73.81	5.20E8
		Tucker [8]	70.95±0.51	71.87	5.20E8
		NC_CTD [20]	71.52±0.03	72.14	5.71E8
		LJSVD	73.61±0.37	**74.43**	5.47E8
		RJSVD-1	**73.66±0.25**	74.14	5.54E8
		RJSVD-2	72.03±0.19	72.23	5.47E8
		Bi-JSVD0.3	72.82±0.29	73.41	5.44E8
		Bi-JSVD0.5	72.96±0.05	73.26	5.43E8
		Bi-JSVD0.7	73.17±0.21	73.80	5.44E8
	13.55	Tai *et al.* [13]	73.60±0.11	73.85	5.71E8
		Tucker [8]	70.88±0.07	71.48	5.70E8
		NC_CTD [20]	71.38±0.15	72.25	6.16E8
		LJSVD	**74.89±0.19**	**75.29**	6.27E8
		RJSVD-1	74.38±0.31	74.95	6.42E8
		RJSVD-2	73.75±0.12	74.26	6.27E8
		Bi-JSVD0.3	72.84±0.17	73.57	6.22E8
		Bi-JSVD0.5	73.13±0.29	73.54	6.25E8
		Bi-JSVD0.7	72.50±0.35	73.16	6.22E8
ResNet-50 75.67% 25.96E8	6.21	Tai *et al.* [13]	70.32±0.27	70.91	7.20E8
		LJSVD	**72.46±0.31**	73.14	7.63E8
		RJSVD-1	71.93±0.41	72.76	7.78E8
		RJSVD-2	71.40±0.20	71.98	7.74E8
		Bi-JSVD0.3	72.04±0.83	**73.65**	7.66E8
		Bi-JSVD0.5	71.23±0.26	72.02	7.66E8
		Bi-JSVD0.7	71.82±0.43	72.84	7.61E8
	5.19	Tai *et al.* [13]	71.35±0.38	72.02	7.97E8
		LJSVD	72.93±0.97	74.49	8.88E8
		RJSVD-1	72.81±0.45	73.68	9.17E8
		RJSVD-2	73.44±0.11	**74.66**	9.11E8
		Bi-JSVD0.3	72.96±0.78	73.88	9.00E8
		Bi-JSVD0.5	72.36±0.22	72.95	8.95E8
		Bi-JSVD0.7	**73.47±0.36**	74.18	8.90E8

TABLE VIII: Comparison of compressed ResNet trained from scratch on CIFAR-100.

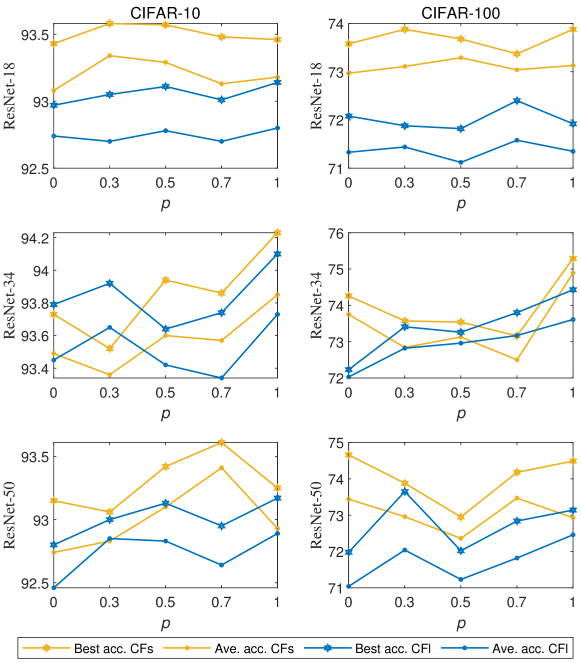

Fig. 4: The performance of Bi-JSVD for CNNs trained from scratch under different p. “Ave. acc.”, “Best acc.”, “CFs” and “CFl” mean Average accuracy, Best accuracy, the smaller CF and the larger one for each CNN shown in the TABLE VII and VIII, respectively.

from scratch outperforms Tai *et al.* with 2.14% higher accuracy on ResNet-50 for CIFAR-100 when CR $= 6.21$, while it is 0.68% with pre-train weights. This observation demonstrates that the proposed methods are more initialization-independent, and thus the joint structure is the main factor in improving performance.

(ii) Different from Figure 3, the curves in Figure 4 are more messy and have no definite trends, indicating that compressed networks with pre-train weights are trained following specific patterns decided by the original networks, whereas the ones trained from scratch are in random.

2) *Varying CF:*

a) Settings: In this section, we verify the performance of the proposed methods under different CFs on ResNet-34 for CIFAR-100. Different from Section IV-A3 that compresses networks with large CFs and decomposes the HID layers, we keep the HID layers uncompressed and compare the proposed methods with Tai *et al.* [13] and Tucker [8] under relatively small CFs ranking from 1.43 to 8.15. Other settings, such as fine-tuning schedules and repeated times, are the same as those in Section IV-A3.

b) Results and Analysis: The results are illustrated in Figure 5. It is shown that all the proposed methods outperform the counterparts under small CFs. Moreover, the proposed methods can effectively alleviate the over-fitting problem under small CFs and thus achieve higher accuracy than the original network, which is also observed in Table VIII. Furthermore, Bi-JSVDp for $p \in [0.3, 0.5, 0.7]$ are superior to RJSVD and LJSVD when CF is less than 5, while LJSVD achieves the best performance in other scenarios, followed by Bi-JSVD and then RJSVD. We believe that the reasons for different performance of the proposed methods are as follows:

(i) - Why can LJSVD outperform RJSVD?
- The only difference between LJSVD and RJSVD is the relative position of the shared sub-convolutional layers obtained by the joint decomposition, as shown in Figure 2. Although the shared sub-tensors are necessary to approximate the original weight tensor, they lack specificity for each layer because of their shareability, and thus may

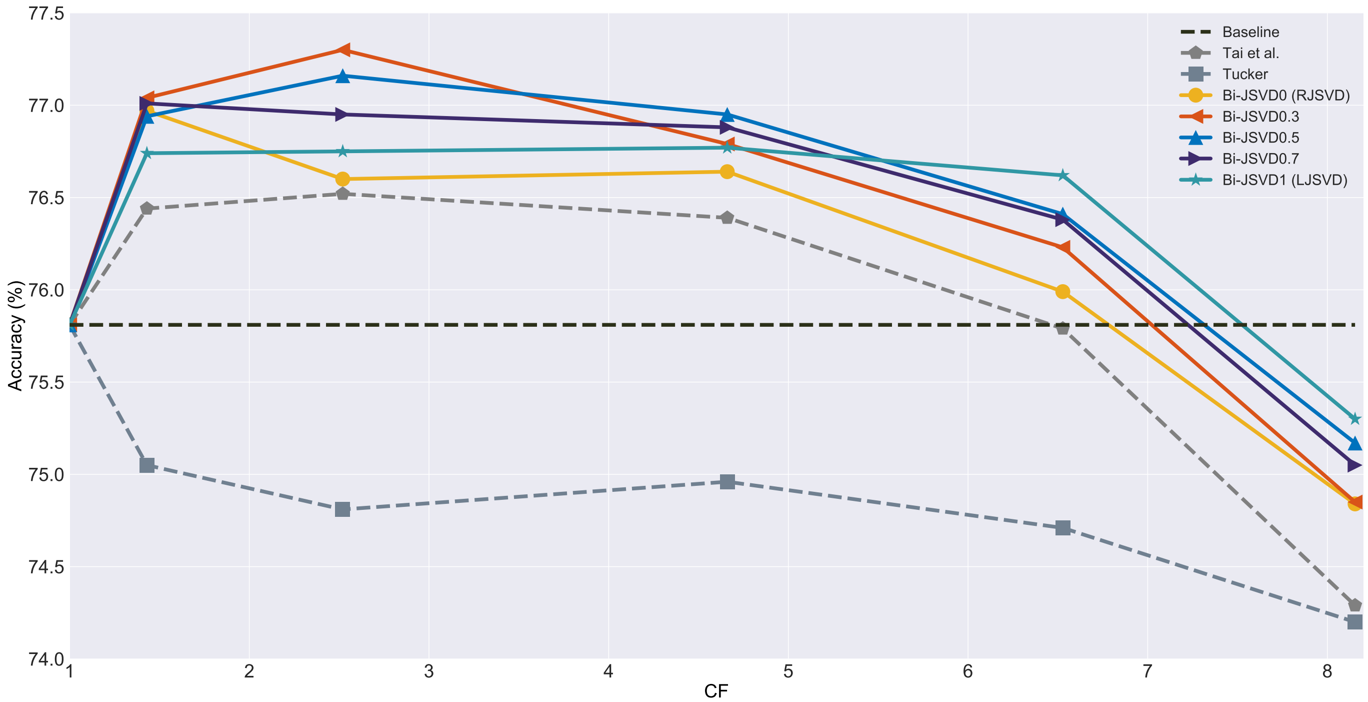

Fig. 5: Comparison of compressed ResNet-34 for CIFAR-100 in relatively small CFs following the “pre-train→decompose→fine-tune” pipeline. HID layers are not decomposed. The results are the average accuracy of three times repeated experiments.

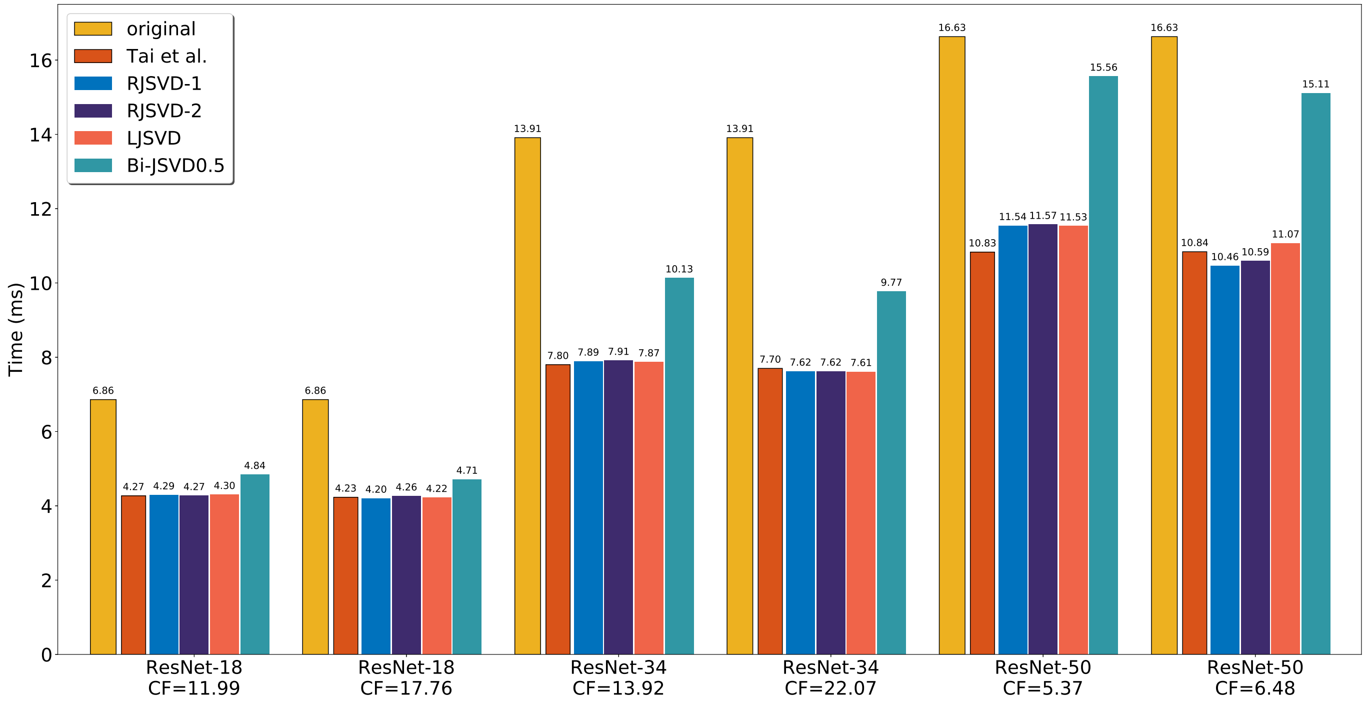

Fig. 6: Realistic acceleration for the forward procedure on CIFAR-10 (evaluated on an Intel E5-2690@2.6GHz CPU; repeated 50,000 times to obtain the average time consumption). Accuracy of the corresponding networks are shown in the Table IV.

yield vague feature maps. In RJSVD, the shared sub-convolutional layers come after the independent ones, making feature maps passed to the next blocks vague. While in LJSVD, the contrary is the case, and thus explicit features can be captured by following layers, which improves the performance of decomposed networks and makes LJSVD superior to RJSVD.

(ii) - Why can Bi-JSVD achieve good performance?
- As shown in Figure 2, Bi-JSVDp ($p \notin$ {0,1}) yields dual paths, which in fact widens the networks. Similar to the Inception Module in GoogLeNet [7], the dual paths can enrich the information carried by feature maps in each layer, thus improving the performance of BI-JSVD.

(iii) - Why does the impact of p in Bi-JSVD seem irregular?
- As discussed above, LJSVD has its advantages compared with RJSVD. Therefore, one may expect better performance with a larger p in Bi-JSVDp since Bi-JSVD0 is equivalent to RJSVD and Bi-JSVD1 to LJSVD. However, the extreme $p = 1$ (single-branch LJSVD) would eliminate the advantage of dual paths in Bi-JSVD. Therefore, there is a trade-off between the dual paths and left-shared structure, and p is the key for the balance. However, the optimal p depends on baseline networks and CFs, making the impact of p irregular. Since there is no uniform and closed-form solution to p, AutoML methods [19] can be utilized to find optimal p for specific scenarios in future work.

D. Realistic Acceleration

a) Settings: As mentioned in Section III, the complexity of compressed convolutional layers produced by the proposed methods is less than the original ones, thus the forward inference can be accelerated. However, there is a wide gap between theoretical and realistic acceleration, which is restricted by the IO delay, buffer switch, efficiency of BLAS libraries [27] and some indecomposable layers such as batch normalization and pooling. Therefore, to evaluate the realistic acceleration fairly, we measure the real forward time of the networks compressed by Tai *et al.* [13] based on the matrix decomposition, and the proposed LJSVD, RJSVD-1, RJSVD-2 and Bi-JSVD0.5 in TABLE IV. Each method is evaluated on an Intel E5-2690@2.6GHz CPU, and the measurement is repeated 50,000 times to calculate the average time cost for one image by setting the batch size to 1.

b) Results and analysis: As shown in Figure 6, all the networks are accelerated without support from customized hardware. The proposed LJSVD, RJSVD-1 and RJSVD-2 can significantly accelerate networks as Tai *et al.* [13] but with much higher accuracy, especially on ResNet-18 and ResNet-34. Since ResNet-50 is built with BottleNecks consisting of 1×1 convolutional layers, the acceleration on ResNet-50 is less obvious than on ResNet-18 and ResNet-34. Moreover, Bi-JSVD is not as outstanding in acceleration as LJSVD and RJSVD, since the dual paths shown in Figure 2 cause more processing time and IO delay. Furthermore, it is shown that a relatively significant gap in FLOPs brings little difference in inference time, indicating that IO delay and buffer switch mentioned above consume a large proportion of inference time.

E. Discussion of the proposed methods

The afore-presented experiments have demonstrated the effectiveness of the proposed RJSVD, LJSVD and Bi-JSVD. However, they have their own cons and pros:

(i) Bi-JSVD widens the networks and thus can provide satisfactory results. However, it requires efforts to find the optimal p for specific baseline models, costs more inference time than RJSVD and LJSVD, and consumes about twice the memory cache storing activations due to the dual paths.

(ii) Both RJSVD and LJSVD are faster than Bi-JSVDp ($p \notin \{0, 1\}$), but RJSVD is inferior to LJSVD because the shared sub-convolution layers in RJSVD are located before the independent ones. However, RJSVD is able to compress the HID layers, while LJSVD and Bi-JSVD fail to do so. As shown in Table IV–VII, RJSVD-1 that takes into account HID layers for joint decomposition can also provide competitive results.

Therefore, the three proposed variants are suitable for different scenarios. Bi-JSVD with a procedure verifying p is suitable for specific scenarios that require not only large CFs but also superior performance. For RJSVD, it would be a good choice if there are many HID layers to be decomposed jointly, while LJSVD is appropriate when there are requirements for fast inference and satisfactory performance.

V. Conclusion And Future Work

In this paper, inspired by the fact that there are repeated modules among CNNs, we propose to compress networks jointly to alleviate performance degradation. Three joint matrix decomposition algorithms, RJSVD, LJSVD and their generalization, Bi-JSVD, are introduced. Extensive experiments on CIFAR-10, CIFAR-100 and the larger scale ImageNet show that our methods can achieve better compression results compared with the state-of-the-art matrix or tensor-based methods, and the speed acceleration in the forward inference period is verified as well.

In future work, we plan to extend the joint compression method to other areas such as network structured pruning to further improve the acceleration performance of CNNs, and use AutoML techniques such as Neural Architecture Search [45], Reinforcement Learning [46] and Evolutionary Algorithms [47][48] to estimate the optimal p for Bi-JSVD as well as target ranks to compress networks adaptively.

Acknowledgement

The work described in this paper was supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2021A1515011706, in part by National Science Fund for Distinguished Young Scholars under Grant 61925108, in part by the Foundation of Shenzhen under Grant JCYJ20190808122005605, in part by Shenzhen Stabilization Support Grant 20200809153412001, and in part by the Open

Research Fund from the Guangdong Provincial Key Laboratory of Big Data Computing, The Chinese University of Hong Kong, Shenzhen, under Grant No. B10120210117-OF07.

References

[1] J. Guan, R. Lai, A. Xiong, Z. Liu, and L. Gu, "Fixed pattern noise reduction for infrared images based on cascade residual attention CNN," *Neurocomputing*, vol. 377, pp. 301–313, 2020.

[2] F. Taherkhani, H. Kazemi, and N. M. Nasrabadi, "Matrix completion for graph-based deep semi-supervised learning," in *AAAI*, 2019, pp. 5058–5065.

[3] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014, pp. 580–587.

[4] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, vol. 97, 2019, pp. 6105–6114.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015, pp. 1–9.

[8] Y. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *ICLR*, 2016.

[9] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," *arXiv:1602.07360*, 2016.

[10] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *NeurIPS*, 2013, pp. 2148–2156.

[11] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *NeurIPS*, 2014, pp. 1269–1277.

[12] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *BMVC*, 2014.

[13] C. Tai, T. Xiao, X. Wang, and W. E, "Convolutional neural networks with low-rank regularization," in *ICLR (Poster)*, 2016.

[14] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *NeurIPS*, 2015, pp. 442–450.

[15] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," in *ICLR (Poster)*, 2015.

[16] T. Garipov, D. Podoprikhin, A. Novikov, and D. P. Vetrov, "Ultimate tensorization: compressing convolutional and FC layers alike," *arXiv:1611.03214*, 2016.

[17] W. Wang, Y. Sun, B. Eriksson, W. Wang, and V. Aggarwal, "Wide compression: Tensor ring nets," in *CVPR*, 2018, pp. 9329–9338.

[18] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," in *CVPR*, 2017, pp. 67–76.

[19] J. Huang, W. Sun, and L. Huang, "Deep neural networks compression learning based on multiobjective evolutionary algorithms," *Neurocomputing*, vol. 378, pp. 260–269, 2020.

[20] W. Sun, S. Chen, L. Huang, H. C. So, and M. Xie, "Deep convolutional neural network compression via coupled tensor decomposition," *IEEE J. Sel. Top. Signal Process.*, vol. 15, no. 3, pp. 603–616, 2021.

[21] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, 2011.

[22] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *NeurIPS*, 1989, pp. 598–605.

[23] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *NeurIPS*, 2015, pp. 1135–1143.

[24] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *ICLR*, 2019.

[25] J. Chen, Y. Xu, W. Sun, and L. Huang, "Joint sparse neural network compression via multi-application multi-objective optimization," *Appl. Intell.*, vol. 51, no. 11, pp. 7837–7854, 2021.

[26] J. Zhou, H. Qi, Y. Chen, and H. Wang, "Progressive principle component analysis for compressing deep convolutional neural networks," *Neurocomputing*, vol. 440, pp. 197–206, 2021.

[27] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *CVPR*, 2019, pp. 4340–4349.

[28] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*, 2018, pp. 2704–2713.

[29] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, ser. Lecture Notes in Computer Science, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9908, 2016, pp. 525–542.

[30] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *ICLR*, Y. Bengio and Y. LeCun, Eds., 2016.

[31] P. Wang, Q. Chen, X. He, and J. Cheng, "Towards accurate post-training network quantization via bit-split and stitching," in *ICML*, vol. 119, 2020, pp. 9847–9856.

[32] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NeurIPS Deep Learning Workshop*, 2015.

[33] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *ICLR*, 2015.

[34] T. Chen, I. J. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," in *ICLR*, 2016.

[35] T. Li, J. Li, Z. Liu, and C. Zhang, "Few sample knowledge distillation for efficient network compression," in *CVPR*, 2020, pp. 14 627–14 635.

[36] S. Srinivas and F. Fleuret, "Knowledge transfer with jacobian matching," in *ICML*, vol. 80, 2018, pp. 4730–4738.

[37] Q. V. Le, A. Karpenko, J. Ngiam, and A. Y. Ng, "ICA with reconstruction cost for efficient overcomplete feature learning," in *NeurIPS*, 2011, pp. 1017–1025.

[38] M. Sun, D. Snyder, Y. Gao, V. K. Nagaraja, M. Rodehorst, S. Panchapagesan, N. Strom, S. Matsoukas, and S. Vitaladevuni, "Compressed time delay neural network for small-footprint keyword spotting," in *INTERSPEECH*, 2017, pp. 3607–3611.

[39] S. Swaminathan, D. Garg, R. Kannan, and F. Andrès, "Sparse low rank factorization for deep neural network compression," *Neurocomputing*, vol. 398, pp. 185–196, 2020.

[40] J. Guo, Y. Li, W. Lin, Y. Chen, and J. Li, "Network decoupling: From regular to depthwise separable convolutions," in *BMVC*, 2018, p. 248.

[41] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[42] B. Wu, D. Wang, G. Zhao, L. Deng, and G. Li, "Hybrid tensor decomposition in neural network compression," *Neural Netw*, vol. 132, pp. 309–320, 2020.

[43] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[44] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: training imagenet in 1 hour," *arXiv:1706.02677*, 2017.

[45] X. Dong and Y. Yang, "Nas-bench-201: Extending the scope of reproducible neural architecture search," in *ICLR*, 2020.

[46] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nat.*, vol. 518, no. 7540, pp. 529–533, 2015.

[47] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[48] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, 2007.