Holistic CNN Compression via Low-rank Decomposition with Knowledge Transfer

Shaohui Lin, Rongrong Ji*, *Senior Member, IEEE,* Chao Chen, Dacheng Tao, *Fellow, IEEE* and Jiebo Luo, *Fellow, IEEE*

Abstract—Convolutional neural networks (CNNs) have achieved remarkable success in various computer vision tasks, which are extremely powerful to deal with massive training data by using tens of millions of parameters. However, CNNs often cost significant memory and computation consumption, which prohibits their usage in resource-limited environments such as mobile or embedded devices. To address the above issues, the existing approaches typically focus on either accelerating the convolutional layers or compressing the fully-connected layers separatedly, without pursuing a joint optimum. In this paper, we overcome such a limitation by introducing a holistic CNN compression framework, termed LRDKT, which works throughout both convolutional and fully-connected layers. First, a low-rank decomposition (LRD) scheme is proposed to remove redundancies across both convolutional kernels and fully-connected matrices, which has a novel closed-form solver to significantly improve the efficiency of the existing iterative optimization solvers. Second, a novel knowledge transfer (KT) based training scheme is introduced. To recover the accumulated accuracy loss and overcome the vanishing gradient, KT explicitly aligns outputs and intermediate responses from a teacher (original) network to its student (compressed) network. We have comprehensively analyzed and evaluated the compression and speedup ratios of the proposed model on MNIST and ILSVRC 2012 benchmarks. In both benchmarks, the proposed scheme has demonstrated superior performance gains over the state-of-the-art methods. We also demonstrate the proposed compression scheme for the task of transfer learning, including domain adaptation and object detection, which show exciting performance gains over the state-of-the-arts. Our source code and compressed models are available at https://github.com/ShaohuiLin/LRDKT.

Index Terms—Convolutional neural networks, Low-rank decomposition, Knowledge transfer, CNN compression, CNN acceleration.

1 Introduction

In recent years, convolutional neural networks (CNNs) have shown impressive performance in various computer vision tasks, for instance image classification [1]–[6], object detection [7]–[9], semantic segmentation [10], *etc*. Coming with the increase of computation power and data scale, nowadays it is feasible to train very deep CNNs with tens of millions of parameters. For instance, a 19-layer VGGNet [2] with 1.5M nodes has 138M parameters, costs 553MB storage, and requires about 15B FLOPs[1] to classify one image with a size of 224×224.

In many emerging scenarios such as mobile and embedded applications, compressing CNNs has become essential and attracted ever-increasing focus from both academic and industrial research. In principle, the convolutional layers are the most time-consuming part, while the fully-connected layers involve most storage cost. Due to such an intrinsic difference, the speedup of convoutional layers and the memory reduction of fully-connected layers are typically treated as two seperated tasks. To speedup convolutional layers, removing redundancy among filters serves as a popular solution, for instance structured pruning [11]–[14], tensor decomposition [15]–[18] and binary network [19]–[21]. To compress parameters of fully-connected layers, various schemes have been proposed recently, including but not limited to, vector quantization [22], hashing trick [23], circulant projection/transformation [24], matrix decomposition [25] and network pruning [26]–[28].

However, a unified framework that simultaneously compresses both convolutional and fully-connected layers is still missing. On the one hand, compressing parameters [22], [23], [26], [27] is not guaranteed to accelerate network, since it might perform irregular memory access that adversely impacts the computation speed. For instance, the work in [22] adopted product quantization to partition the weights into sub-matrices and reconstruct the parameter matrices separately in online inference. The work in [23] adopted two hash functions to group network connections into hash buckets uniformly through weight sharing. The works in [26], [27] discarded low-weight connections below a threshold to reduce the total amount of network parameters. All above works [22], [23], [26], [27] may produce non-structured sparse connections and therefore require additional index calculation overhead, leading to irregular memory access. On the other hand, the acceleration of convolutional computation (*e.g.*, by using low-rank decomposition) typically involves highly complex iterative algorithms and data-dependent criterions [15], [16], [18]. It is therefore difficult to directly and efficiently deploy to fully-connected layers, which will take much longer computation time and lead to a bad local minimum.

Moreover, the existing compression and acceleration methods still operate on a layer-wise manner, which cannot explicitly model the overall accuracy loss throughout the entire network. From this perspective, the above schemes can be regraded as an "implicit" and "local" compression/acceleration. In terms of "implicit" compression, the existing schemes [15], [16], [22],

- *S. Lin, R. Ji (Corresponding author) and C. Chen are with the Fujian Key Laboratory of Sensing and Computing for Smart City, and the School of Information Science and Engineering, Xiamen University, 361005, China, E-mail: (rrji@xmu.edu.cn).*
- *D. Tao is with the UBTECH Sydney Artificial Intelligence Centre and the School of Information Technologies, the Faculty of Engineering and Information Technologies, the University of Sydney, 6 Cleveland St, Darlington, NSW 2008, Australia.*
- *J. Luo is with the Department of Computer Science, University of Rochester, Rochester, NY 14623 USA.*

1. FLOPs: The number of Floating-point operations.

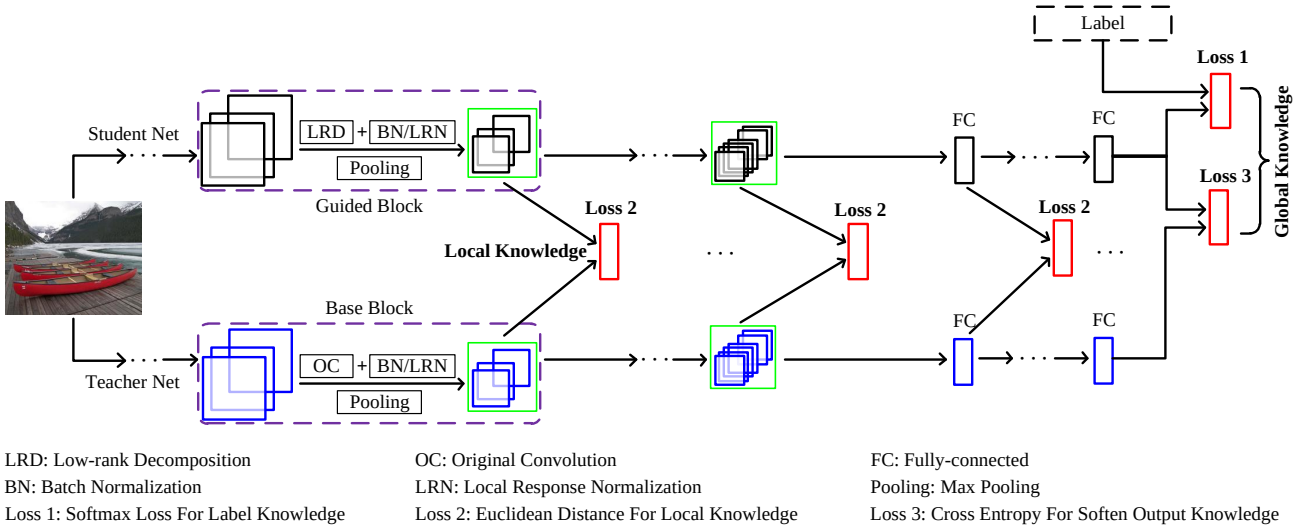


Fig. 1. The framework of the proposed LRDKT. A low-rank decomposition (LRD) based compression scheme is first constructed to form a guided block in the student network. Next, the "local" and "global" knowledge is transferred by KT in a unified way, which deals with the performance degradation caused by LRD compression. The based and guided blocks are defined in Sec. 3.3.

[23], [29] only consider to approximate the parameters $\mathbf{W}$ of convolutional or fully-connected layers with $\hat{\mathbf{W}}$ by minimizing their Euclidean distance $\|\mathbf{W} - \hat{\mathbf{W}}\|_F^2$. This setting is indeed problematic, which does not directly recover the output of CNNs, *i.e.*, the responses of the last fully-connected layer before sending to the softmax-like classifier. In terms of "local" compression, the existing schemes do not jointly compress parameters across different layers, *i.e.*, holistically compressing parameters throughout the entire network. In other words, inter-layer parameter correlations are simply ignored in [15], [19], [22], [28], [29]. However, due to the nonlinear transformation functions (*e.g.*, rectifier linear unit (ReLU) and BN [30]), the small reconstruction error between $\mathbf{W}$ and $\hat{\mathbf{W}}$ in each layer would be propagated and magnified throughout the entire network, leading to large accumulated error.

To address the above problems, in this paper we propose a unified framework towards *holistic* and *explicit* CNN compression. We target at jointly compressing both the convolutional and fully-connected layers, aiming to simultaneously speedup online inference and reduce memory consumption. In principle, our framework is first deployed based on an inter-layer compression formulation with a strictly closed-form guarantee, followed by a re-learning process to align the responses between the original network and compressed network. In particular, we first propose a novel compression scheme based on low-rank decomposition (LRD), which simutaneously accelerates the convolutional layers and compresses the fully-connected layers with an exact closed-form solution. It differs from the methods proposed by Jaderberg *et al.* [15], which obtained low-rank factors using a highly complex iterative optimization solver with a data-dependent criterion. The proposed closed-form low-rank decomposition is extremely fast to implement without any iteration, which also leads to better generalized ability for the produced low-rank CNN structure.

To further reduce the accuracy loss caused by LRD in high compression rates, we further present a novel and effective knowledge transfer (KT) scheme to "explicitly" align outputs and intermediate responses from the teacher (original) network to its student (compressed) network. The proposed KT scheme operates nonlinear transformation functions within and across all layers, and minimizes both "local" and "global" reconstruction errors in a unified way, which differs from the sectional schemes in [31], [32], and is much easier to converge and implement. KT is inspired from the Knowledge Distillation (KD) [33], which trains a student network from the global output of a teacher network with a similar depth. However, with the increasing depth of the student network (*e.g.*, the deep low-rank CNN structure investigated in this paper), KD based optimization becomes more difficult, which will lead to undesirable phenomenon of vanishing gradient [34]. In contrast, the proposed KT extends from KD to enable learning thinner and deeper student network, which uses the local knowledge from the hidden layers of teacher network to guide the training process of the student network. Additionally, inspired by ladder network [35] and FitNets [32], we introduce a simpler local loss function, based on measuring the Euclidean distance of the presentation between the specific layers' activation of the teacher network and the student network, which obtains the local knowledge from teacher network without any regressors. Using such specific local knowledge, KT can be trained in an end-to-end manner, and effectively improve the discriminability of the student network whilst overcoming the vanishing gradient when training a deeper student network.

Finally, LRD and KT are effectively integrated into a unified framework, in which the good initial weights provided by the closed-from LRD can be employed to improve the generalization ability of the student network, while training the thinner and deeper student network with higher discriminability is facilitated by using KT. We term the proposed scheme LRDKT, the workflow of which is depicted in Fig. 1.

Quantitatively, the proposed method has demonstrated significant advantages on LeNet for the task of MNIST classification [4], as well as on AlexNet, VGG-16 and ResNet-50 for the task of ImageNet classification [36]. Comparing to several state-of-the-art methods, the proposed scheme performs better for CNN compression and acceleration, *e.g.*, $4.5\times$ CPU speedup and $16.98\times$ compression, with a negligible classification accuracy loss for LeNet, $2.93\times$ CPU speedup, $2.12\times$ GPU speedup and $9.6\times$ compression with an increase of 0.85% Top-1 classification error for AlexNet, as well as $2.43\times$ CPU speedup, $2.27\times$ GPU speedup and $4.5\times$ compression with an increase of -0.50% Top-

1 classification error for VGG-16. By replacing the traditional fully-connected layers with global average pooling [3] (GAP), the AlexNet and VGG-16 can be further compressed by a factor of $2.0\times$ GPU speedup and $55.88\times$ compression, with an increase of 3.01% Top-1 error, and $2.33\times$ GPU speedup and $41.92\times$ compression, with only an increase of 0.18% Top-1 classification error, respectively. For ResNet-50, LRDKT achieves $2.02\times$ CPU speedup and $2.57\times$ compression with an increase of 0.72% Top-1 error, all of which are state-of-the-art in the existing literature. Moreover, we evaluate the transfer learning ability of LRDKT in compressing VGG-16 model when being applied to domain adaptation and objection detection. We have achieved $11.82\times$ FLOPs reduction and $14.21\times$ compression with only 0.9% increase in Top-1 error on the task of domain adaptation, and 0.2% mAP drops with a factor of $3.10\times$ GPU speedup (0.2% mAP drops with a factor of $2.51\times$ speedup on Jetson TX2) on the task of object detection, both of which are very competitive and demonstrate the strong generalization of the proposed scheme.

2 Related Work

The convolutional layers of CNNs cost most computation consumption, while the fully-connected layers cost most memory consumption, respectively. Related work in CNN acceleration can be categorized into four groups, *i.e.*, parameter quantization, structured pruning, compact model designs, and tensor approximation. For parameter quantization, Gong *et al.* [22] and Wu *et al*. [37] employed vector quantization over parameters to reduce the redundancy in the parameter space. Recently, directly learning binary weights to quantize model parameters was proposed in [20], [21], which achieved comparable accuracy to the original networks on small-scale datasets like MNIST. To improve the accuracy of the binary network on large-scale datasets like ImageNet, Rastegari *et al.* [19] proposed XNOR-Net to further incorporate a real coefficient to compensate for the binarization error. For structured pruning, Wen *et al.* [11] incorporated group constraints as a structured sparsity regularization to compress convolution layers. The magnitude-based pruning method [12] aims to prune filters with their corresponding feature maps, which is less efficient in determining the importance of filters. More recently, a Taylor expansion based pruning scheme was proposed in [38] to iteratively prune one filter and fine-tune the rest network, which is however costly in training especially to handle large amounts of redundant filters in deep networks. Luo *et al.* [13] and He *et al.* [39] utilized statistics information computed from the next layer to guide a greedy layer-wise pruning. However, it is time-consuming to collect new input/output pairs in the current pruning layer as training samples, which are expensive to be stored in training. For compact model designs, the key idea of designing a compact model is to replace the loose and over-parametric filters with a compact block to accelerate the convolutional computation. For example, comparing to simply stacking convolutional layers, the inception module in GoogLeNet [5] was proposed to stack on top of each other to increase the network depth with much lower computational budget. The bottleneck structure in ResNet [6] was proposed to achieve remarkable performance. And the fire module in SqueezeNet [40] was proposed to maintain competitive accuracy with few parameters. Besides, the group convolution is first used in AlexNet [1] for distributing the model over two GPUs to handle the memory issue, which has recently been widely used in model designs. For example, Ioannou *et al.* [41] proposed a root module to reduce the computational cost without compromising accuracy, which contains a specific layer with filter groups and a spatial convolutional layer with 1×1 convolution. Zhang *et al.* [42] proposed the ShuffleNet to employ a channel shuffle operator to help the information flowing across feature channels, which is brought by group convolutions. As an extreme case of group convolution, depthwise convolution performs lightweight filtering by applying a single convolutional filter into one input channel, which can be further combined with pointwise convolution to form a depthwise separable convolution. Recently, MobileNet [43] and Xception [44] have used the depthwise separable convolutions to achieve impressive performance among lightweight models. However, the above compact modules or convolutions are designed for specific models, which are less general for accelerating other deep models. Compared to designing a compact model, our method is more general to accelerate various deep networks (*e.g.*, VGG Nets, GoogLeNet and ResNet) by a closed-form low-rank decomposition on filters/weights. For tensor approximation, recent methods have been proposed to decompose a convolutional filter into a sequence of tensor convolutions with fewer parameters [15]–[17], [45]. Such approaches typically adopt inexact low-rank factorizations, which can considerably accelerate the computation by a low-rank based approximation. Notably, these approaches use simple fine-tuning to recover the accumulated loss of accuracy. It is, however, problematic due to the vanishing gradient in back propagation throughout the deep networks. Different from approximating filters in the pre-trained networks, Ioannou *et al.* [46] combined a set of small basis filters into more complex filters, and learned these basis filters from the scratch with an effective weight initialization. Although with the same spatial size of the small low-rank factors (*e.g.*, 1×3 and 3×1 spatial size) in both [15] and [46], the way to obtain these low-rank factors is quite different, *i.e.*, the work in [15] obtains these factors by minimizing the local reconstruction error via data-dependent iterative optimization, while the work in [46] obtains them by training the entire network directly. Our method differs from the schemes of [15] and [46]. We employ a low-rank decomposition across convolutional layers with a closed-form solution, instead of the traditional data-dependent iterations in [15], [17], [18], or training the deeper and thinner network from scratch [46].

Related work in CNN compression can be further categorized into four groups, *i.e.*, parameter sharing, parameter pruning, matrix decomposition, and knowledge distillation. For parameter sharing, Chen *et al*. [23] proposed a HashedNet, which uses a low-cost hash function to group weights between two connected layers into hash buckets. Cheng *et al*. [24] proposed to replace the linear projection in fully-connected layers with a circulant projection, which reduces the memory cost and enables the usage of Fast Fourier Transform to accelerate the computation. Yang *et al*. [47] introduced a novel Adaptive Fastfood Transform to reparameterize the matrix-vector multiplication of fully-connected layers. For parameter pruning, LeCun *et al.* [48] and Hassibi *et al.* [49] proposed a saliency measurement by computing the Hessian matrix of the loss function with respect to the parameters, and then pruned the parameters with low saliency values. However, such methods require computing second-order derivatives, which therefore add additional computation cost. Srinivas and Babu [28] explored the redundancy among neurons, upon which a data-free pruning was proposed to remove redundant nodes. Han *et al*. [26] proposed a pruning scheme based on low-weight connections to reduce the total amount of parameters and operations in the network.

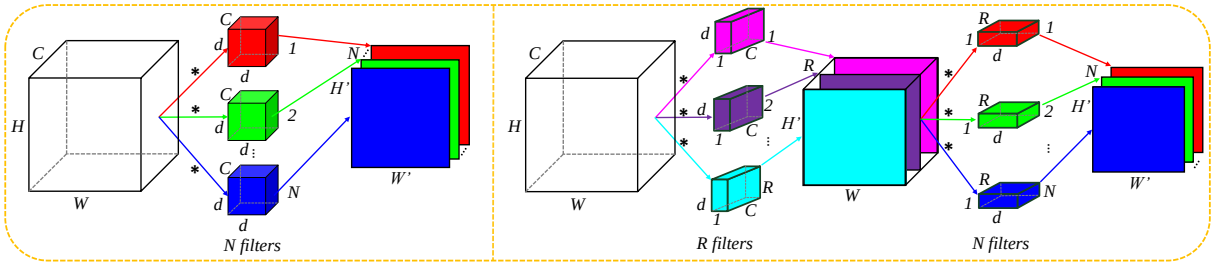


Fig. 2. Low-rank decomposition for compressing a generalized convolutional operation. The R, $H \times W \times C$, $d \times d \times C \times N$, $H' \times W' \times N$ are the rank, input size, filter size and output size, respectively. Left: Original convolution. Right: Low-rank constraint convolution with rank R

For matrix decomposition, Denil *et al.* [29] adopted a low-rank decomposition to compress the weights in the fully-connected layers in a layer-by-layer manner. Novikov *et al.* [50] converted the dense weight matrices of the fully-connected layers to the Tensor Train (TT) format, such that the number of parameters is reduced in scales while the representation power of the original network is still preserved. For knowledge distillation, Buciluǎ *et al.* [51] proposed to mimic a complicated network with a simple network, which shows the possibility of inter-network knowledge transfer. Following this idea, several methods [32], [33], [52] employ dark knowledge to train a student network based on the soft output of a complicated teacher network. Specifically, Romero *et al.* [32] proposed FitNet, which combines dark knowledge and the knowledge from the teacher's hint layer by regressors with additional parameters, which is trained in a stage-wise fashion. Although FitNet achieves state-of-the-art results in small-scale datasets (*e.g.*, CIFAR-10 and CIFAR-100), its performance is less exciting on large-scale dataset (*e.g.*, ImageNet). Moreover, such a limited performance gain requires a complex and costly training. Different from FitNets, the proposed knowledge transfer can perform deeper CNNs on ImageNet, and significantly reduce the performance degradation, which "explicitly" aligns the outputs and intermediate responses from the teacher network to its student network without any regressors, which overcomes the vanishing gradient by jointly minimizing both "local" and "global" reconstruction errors by an end-to-end training.

Recently, it has shown promising performance gains to combine multiple aforementioned schemes to compress or accelerate convolutional neural networks. Han *et al.* [27] removed the redundant connections [26], quantified weights [22], and used Huffman coding to compress CNNs. Wang *et al.* [53] handled convolutional filters in the frequency domain using the discrete cosine transform, and then employed K-means clustering, quantization and Huffman coding to compress CNNs with a high compression ratio. Our scheme can be also integrated with the aforementioned methods (*e.g.* structured pruning [11], [13] or parameter quantization [22], [37]) to further improve our compression/speedup ratio, which is however orthogonal to the core contribution of this paper.

3 The Proposed Framework

In this section, we first describe the preliminary. Then, we present our low-rank decomposition on both kernel tensors of convolutional layers and matrix-weights of fully-connected layers. Finally, we employ knowledge transfer to minimize both "local" and "global" accumulated errors between the original (teacher) network and the compressed (student) network.

3.1 The Preliminary

CNN [1]–[6] can be viewed as a feed-forward multi-layer architecture that maps the input image to a certain output vector. Units in CNN are organized as a sequence of 3D tensors with two spatial dimensions and a third "map" or "channel" dimension[2]. In the convolutional layer, the convolution transforms an input tensor $\mathcal{I}$ of size $H \times W \times C$ into an output tensor $\mathcal{O}$ of size $H' \times W' \times N$ by the following linear mapping:

$$\mathcal{O}_{h',w',n} = \sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{c=1}^{C} \mathcal{K}_{i,j,c,n}\mathcal{I}_{h_i,w_j,c}, \tag{1}$$

where the set of convolutional filters $\mathcal{K}$ is given by a tensor of size $d \times d \times C \times N$. Here, $d \times d$ corresponds to the spatial dimension, while C and N are the numbers of input and output channels, respectively. The height and width of the input are denoted as $h_i = h' + i - 1$ and $w_j = w' + j - 1$. For simplicity, we assume an unit stride with no zero-padding and skip biases. This computation process is shown in the left of Fig. 2. To describe the filter redundancy discussed subsequently, Eq. (1) is rewritten as:

$$\mathcal{O}_n = \mathcal{K}_n * \mathcal{I} = \sum_{c=1}^{C} \mathcal{K}_n^c * \mathcal{I}^c, \quad n = 1, 2, \cdots, N, \tag{2}$$

where $\mathcal{K}_n$ is the n-th 3-D filter ($n \in [1, 2, \cdots, N]$), which contains a stack of 2-D filters $\mathcal{K}_n^c \in \mathbb{R}^{d\times d}$, $c \in [1, 2, \cdots, C]$.

In the fully-connected layer, the main operation is the matrix-by-matrix multiplication, which multiplies an input matrix $\mathbf{X} \in \mathbb{R}^{d\times b}$ with a weight matrix $\mathbf{W} \in \mathbb{R}^{h\times d}$ to produce an output matrix $\mathbf{Z} \in \mathbb{R}^{h\times b}$, *i.e.*,

$$\mathbf{Z} = \mathbf{W}\mathbf{X}. \tag{3}$$

Eq. (1)-(3) enable us to introduce the low-rank decomposition to convolutional layers over tensor kernels for acceleration, as well as to fully-connected layers over matrix weights for compression.

3.2 Holistic Low-rank Decomposition

For tensor convolution in Eq. (1), we construct a set of low-rank filter bases with rank-1 in the spatial domain. Correspondingly, each convolutional layer is factorized as two new convolutional layers with rectangular filters. The first convolutional layer has R filters of spatial size $d \times 1$, resulting in a filter bank of $\{\mathcal{V}_r \in \mathbb{R}^{d\times 1\times C} : r \in [1, 2, \cdots, R]\}$ to produce output feature maps $\mathcal{S} \in \mathbb{R}^{H'\times W\times R}$. The second convolutional layer has N filters of spatial size $1 \times d$, resulting in a filter bank

2. When a CNN is applied to a batch of images, these tensors are 4D with the fourth dimension corresponding to the batch size. This dimension does not affect the derivation, and therefore in this paper we only consider the 3D case.

of $\{\mathcal{T}_n \in \mathbb{R}^{1\times d\times R} : n \in [1,2,\cdots,N]\}$ to produce output feature maps $\mathcal{O} \in \mathbb{R}^{H'\times W'\times N}$ with the same size of the convolutional output. Therefore, the convolution by the original filters $\mathcal{O}_n = \sum_{c=1}^{C} \mathcal{K}_n^c * \mathcal{I}^c$ in Eq. (2) is approximated by:

$$\begin{aligned} \mathcal{O}_n \approx \mathcal{T}_n * \mathcal{S} &= \sum_{r=1}^{R} \mathcal{T}_n^r * (\sum_{c=1}^{C} \mathcal{V}_r^c * \mathcal{I}^c) \\ &= \sum_{c=1}^{C} \Big(\sum_{r=1}^{R} \mathcal{T}_n^r * \mathcal{V}_r^c \Big) * \mathcal{I}^c, \end{aligned} \tag{4}$$

where an approximation $\hat{\mathcal{K}}$ of $\mathcal{K}$ can be obtained by the sum of r separable filters $\mathcal{T}_n^r * \mathcal{V}_r^c$, *i.e.*,

$$\hat{\mathcal{K}}_n^c = \sum_{r=1}^{R} \mathcal{T}_n^r * \mathcal{V}_r^c. \tag{5}$$

In Eq. (5), R is a hyper-parameter controlling the rank. The approximated convolution computation is shown in the right of Fig. 2. By far, we reduce the computation complexity from the original $\mathbf{O}(NCd^2H'W')$ to $\mathbf{O}\big(R(CW + NW')dH'\big)$. Assuming the image width $W \gg d$ and the variable $R = N = C$, the original convolution is accelerated about d times.

We obtain the approximated low-rank filter basis $\mathcal{T}$ and $\mathcal{V}$ by solving the optimization problem below:

$$\min_{\mathcal{T},\mathcal{V}} \mathcal{L}_1(\mathcal{T},\mathcal{V}) = \sum_{n=1}^{N}\sum_{c=1}^{C} \big\|\mathcal{K}_n^c - \sum_{r=1}^{R} \mathcal{T}_n^r * \mathcal{V}_r^c\big\|_F^2. \tag{6}$$

We further prove the following theorem to get a closed-form solution of Eq. (6). The corresponding solution $(\mathcal{T},\mathcal{V})$ is defined as a set of low-rank constrained filters:

Theorem 1. *Define a function that maps a tensor to a matrix $\sigma : \mathbb{R}^{d\times d\times C\times N} \to \mathbb{R}^{Cd\times Nd}, \mathcal{K} \mapsto \sigma(\mathcal{K})$, in which a tensor element (i_1,i_2,i_3,i_4) maps to a matrix element (j_1,j_2), such that*

$$j_1 = (i_3-1)d + i_1, \quad j_2 = (i_4-1)d + i_2.$$

Define $\mathbf{K} \coloneqq \sigma(\mathcal{K})$. Let $\mathbf{K} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ be the Singular Value Decomposition (SVD) of $\mathbf{K}$. We construct:

$$\begin{aligned} \hat{\mathbf{V}}_r^c(j) &= \mathbf{U}_{(c-1)d+j,r}\sqrt{\mathbf{\Sigma}_{\mathbf{r},\mathbf{r}}}, \\ \hat{\mathbf{T}}_n^r(j) &= \sqrt{\mathbf{\Sigma}_{\mathbf{r},\mathbf{r}}}\mathbf{V}_{r,(n-1)d+j}^\top. \end{aligned} \tag{7}$$

Then $(\hat{\mathcal{T}},\hat{\mathcal{V}})$ is a solution of Eq. (6).

Proof. We consider the following optimization problem:

$$\begin{aligned} &\min_{\bar{\mathbf{K}}} \mathcal{L}_2(\bar{\mathbf{K}}) = \big\|\bar{\mathbf{K}} - \mathbf{K}\big\|_F^2, \\ &\textit{s.t.} \quad \textit{rank}(\bar{\mathbf{K}}) \le R. \end{aligned} \tag{8}$$

Let $(\mathcal{T}^*,\mathcal{V}^*)$ be an optimal solution of Eq. (6). Then we can construct $\bar{\mathbf{K}}$ as follows:

$$\bar{\mathbf{K}} = \sum_{r=1}^{R} \begin{bmatrix} \mathcal{V}_r^{1*} \\ \mathcal{V}_r^{2*} \\ \vdots \\ \mathcal{V}_r^{C*} \end{bmatrix} \begin{bmatrix} \mathcal{T}_1^{r*}, & \mathcal{T}_2^{r*}, & \cdots, & \mathcal{T}_N^{r*} \end{bmatrix}.$$

Due to the separability of the Frobenius norm, we have:

$$\mathcal{L}_1(\mathcal{T}^*,\mathcal{V}^*) = \mathcal{L}_2(\bar{\mathbf{K}}).$$

As $rank(\bar{\mathbf{K}}) \le R$, $\bar{\mathbf{K}}$ is a feasible solution for Eq. (8). Therefore, for any solution $\mathbf{K}^*$ of Eq. (8), we have:

$$\mathcal{L}_2(\mathbf{K}^*) \le \mathcal{L}_2(\bar{\mathbf{K}}) = \mathcal{L}_1(\mathcal{T}^*,\mathcal{V}^*). \tag{9}$$

On the other hand, If $\mathbf{K}^*$ is an optimal solution of Eq. (8), we can construct a feasible solution $(\hat{\mathcal{T}},\hat{\mathcal{V}})$ of Eq. (6) using Eq. (7). Then we have:

$$\mathcal{L}_1(\mathcal{T}^*,\mathcal{V}^*) \le \mathcal{L}_1(\hat{\mathcal{T}},\hat{\mathcal{V}}) = \mathcal{L}_2(\mathbf{K}^*).$$

Together with Eq. (9), we have:

$$\mathcal{L}_1(\hat{\mathcal{T}},\hat{\mathcal{V}}) = \mathcal{L}_2(\mathbf{K}^*) = \mathcal{L}_1(\mathcal{T}^*,\mathcal{V}^*). \tag{10}$$

Therefore, we prove that $(\hat{\mathcal{T}},\hat{\mathcal{V}})$ is a solution of Eq. (6). □

Note that the solution of Eq. (6) is not unique. Indeed, if $(\mathcal{T},\mathcal{V})$ is a solution of Eq. (6), $(\frac{1}{\alpha}\mathcal{T},\alpha\mathcal{V})$ is also a solution for any $\alpha \neq 0$. However, these solutions are equivalent in our application.

The solution of Eq. (6) provided by Theorem 1 is extremely fast, which has a closed-form solution without any iteration, *i.e.*, significantly better than iterative algorithms widely adopted in the literature [15], [17], [18]. In comparison, the works in [15], [17], [18] typically take much more time to learn the approximated convolutional kernels by using a data-dependent criterion, which are very likely to stuck to bad local minimum.

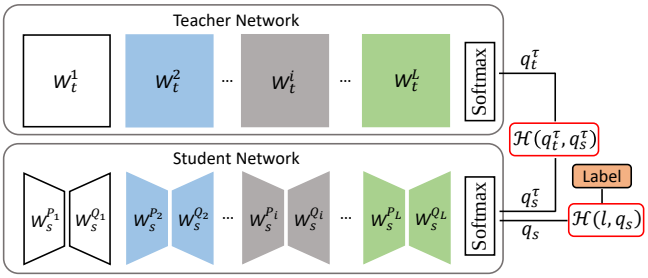
For matrix-by-matrix multiplication in the fully-connected layer described in Eq. (3), we also consider the low-rank approximation $\hat{\mathbf{W}} \in \mathbb{R}^{h\times d}$ of the original weight $\mathbf{W} \in \mathbb{R}^{h\times d}$. To obtain an approximated low-rank subspace, we solve the following optimization problem:

$$\begin{aligned} &\min_{\hat{\mathbf{W}}} \|\mathbf{W} - \hat{\mathbf{W}}\|_F^2 \\ &\textit{s.t.} \quad \textit{rank}(\hat{\mathbf{W}}) \le r. \end{aligned} \tag{11}$$

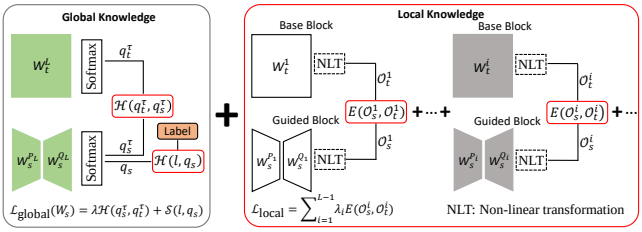
Eq. (11) is solved by a low-rank decomposition using SVD [54], *a.k.a.* $\hat{\mathbf{W}} = \hat{\mathbf{U}}_r\hat{\Sigma}_r\hat{\mathbf{V}}_r^\top$, where $\hat{\mathbf{U}}_r \in \mathbb{R}^{h\times r}$ and $\hat{\mathbf{V}}_r \in \mathbb{R}^{d\times r}$ are two submatrices that correspond to the top r singular vectors in $\mathbf{U}$ and $\mathbf{V}$. The diagonal elements in $\hat{\Sigma}_r \in \mathbb{R}^{r\times r}$ correspond to the r largest singular values in Σ, which is a diagonal matrix by running SVD over $\mathbf{W}$. We then obtain the decomposition of $\hat{\mathbf{W}}$ as $\hat{\mathbf{P}}\hat{\mathbf{Q}}^\top$, where $\hat{\mathbf{P}} = \hat{\mathbf{U}}_r\hat{\Sigma}_r^{\frac{1}{2}}$ and $\hat{\mathbf{Q}} = \hat{\mathbf{V}}_r\hat{\Sigma}_r^{\frac{1}{2}}$.

However, directly applying this low-rank decomposition to multiple layers (without retraining) will lead to approximated error of each layer, which would be further accumulated and propagated. Therefore, an asymmetric data reconstruction is required to alleviate this problem. In this paper, we construct a novel local loss function (*a.k.a.* local asymmetric reconstruction error) to align the output of approximated layers and original layers. To minimize the local asymmetric reconstruction error, one feasible solution is to employ the Stochastic Gradient Descent (SGD) to obtain the sub-optimal solution. Alternatively, we construct an overall loss function in the proposed knowledge transfer (Sec. 3.3), which considers both the local asymmetric reconstruction error and the global reconstruction error to reduce the overall error in the final output layer. Then, as an easy and effective solver, SGD is employed to minimize the overall loss function, as detailed in Sec. 3.3 subsequently.

Using low-rank decomposition, we replace the large convolutional kernel with two small ones in convolutional layers, and replace the large matrix-weights with two small ones in fully-connected layers, respectively. Therefore, compared to the original convolution in Eq. (1) and matrix multiplication in Eq. (3), we only



(a) Knowledge distillation [33]



(b) Knowledge transfer

Fig. 3. The pipeline of training a student network using knowledge distillation or knowledge transfer.

require $R(CW+NW')dH'$ multiplication-addition operations in the convolutional layer, as well as $r(h+d)$ parameters in the fully-connected layer, respectively. The corresponding speedup ratio S_r in the convolutional layer and compression ratio C_r in the fully-connected layer are:

$$S_r = \frac{d^2CNH'W'}{R(CW+NW')dH} = \frac{dCNH'W'}{R(CW+NW')H},$$

$$C_r = \frac{hd}{r(h+d)}.$$

3.3 Accuracy Recovery by Knowledge Transfer

Since we minimize the reconstruction error of linear filters/weights instead of the non-linear responses, the classification accuracy would be dropped after network compression. It is simple and straightforward to fine-tune the compressed network to recover the accuracy. However, it has been quantitatively shown that the recovered accuracy is far less comparable to the original network given a high compression rate [15], [16], [29].

Alternatively, knowledge distillation [33] is becoming a promising solution, which aims to transfer knowledge from a teacher network to a student network to boost the accuracy of the student network. The idea is to allow the student network to capture the global knowledge (*a.k.a.* dark knowledge), which contains the information provided by the true labels, as well as the finer structure learned by the teacher network.

As shown in Fig. 3(a), the framework of knowledge distillation can be summarized as follows:

Information flow from global knowledge. Let s and t be a student network with a "softmax" output $q_s = softmax\big(h_f(\mathcal{X};\mathcal{W}_s)\big)$, and a teacher network with a "softmax" output $q_t = softmax\big(g_f(\mathcal{X};\mathcal{W}_t)\big)$, where h_f and g_f are two global functions that map an input image $\mathcal{X}$ to the output of student and teacher networks, respectively. $\mathcal{W}_s$ and $\mathcal{W}_t$ stand for all parameters in the student and teacher networks, respectively. The student network will be trained such that the probability distribution q_s of its output not only approaches to that (q_t) of the teacher network, but also approaches to the true labels l. When q_t is close to the one-hot code presentation of the true labels l, less information provided by q_t will decrease the accuracy of the student network in training. To this end, a temperature parameter τ [33] is utilized to soften the information arising from the output of the teacher network, *i.e.*, $g_f(\mathcal{X};\mathcal{W}_t)$, which provides more information in training. The same temperature parameter is applied to the output of the student network to produce the softened target probability distribution q_s^τ, compared to the softened output q_t^τ of the teacher network:

$$q_s^\tau = softmax\big(\frac{h_f(\mathcal{X};\mathcal{W}_s)}{\tau}\big), q_t^\tau = softmax\big(\frac{g_f(\mathcal{X};\mathcal{W}_t)}{\tau}\big). \quad (12)$$

Therefore, the student network is trained to optimize the global loss function as follows:

$$\mathcal{L}_{\text{global}} = \lambda\mathcal{H}(q_s^\tau, q_t^\tau) + \mathcal{H}(l, q_s), \quad (13)$$

where $\mathcal{H}$ refers to the cross-entropy based loss function, and λ is a hyper-parameter to balance the cross-entropies of $\mathcal{H}(q_s^\tau, q_t^\tau)$ and $\mathcal{H}(l, q_s)$. In addition, the first term in Eq. (13) is to learn the student network from the softened output of the teacher network, whereas the second term is to guide the student network to learn from the true labels.

From this perspective, the "local" knowledge is unexploited in knowledge distillation, which affects the discriminability of the existing schemes and suffers from the vanishing gradients [55]. To explain, during backpropagation, the derivatives of shallow layers (bottom layers) in a network are formed using products of weight matrices and derivatives of non-linearities from the downstream layers (top layers). The eigenvalues of the filters are often small, while the derivatives of non-linearities are nearly zero in deeper network with a set of low-rank filters. Therefore, the multiplication by these terms annihilates information. And the resulting gradients in shallow layers contain little information about the error, which is thereby difficult to update in the shallow layers. Correspondingly, the discriminability of the learned model is reduced. Although hints training loss [32] has been proposed to help the training of deep FitNets, it still cannot overcome the vanishing gradients in a two-stage training. In fact, the whole parameters in a deep student network are only updated by KD training, while the hints training loss is used to initialize the parameters in the deep student network.

3.3.1 *Knowledge Transfer*

To further improve the discriminality of the compressed model and overcome the vanishing gradient, we propose a novel knowledge transfer (KT) method to transfer both global (*i.e.*, dark knowledge [33] [32]) and local knowledge (*i.e.*, the knowledge from hidden layers) between the teacher network and the student network. Fig. 3(b) shows the proposed knowledge transfer framework. In this framework, both local reconstruction error and global accumulated error between student network and teacher network are integrated into an overall objective function to explicitly model knowledge, which differs from the existing work that separates such errors in different sub-objective functions [31] and FitNets [32]. Our main inspiration also comes from DSNs [56] on domain adaptation. We consider the teacher network as a source domain, and the student network as a target domain. To produce better represent knowledge, we encourage the integration of global accumulated error with local reconstruction error to share the shared-space component of the representation, which is similar to the "collective knowledge" of student network in DenseNet [57].

To better train the deep low-rank network (*i.e.*, the student network) and to further understand the proposed KT, we further

introduce the concept of *base* block in the teacher network and the *guided* block in the student network. The *base* block is defined as the output of hidden layers in the teacher network, which is responsible to guide the learning of the student network. Analogously, the *guided* block is defined to learn from the hidden layers of the teacher network. Specifically, the original convolution operation, batch normalization or local response normalization, and max pooling are included in the base block of a single layer. Meanwhile as shown in Fig. 1, the low-rank approximated convolution, batch or local response normalization, and max pooling are included in the guided block of the same single layer[3].

To operate the above principles in the base and guided blocks, the following steps are conducted:

Step (1). Construct the supervision of hidden layers to learn the local knowledge. Instead of using the cross entropy loss function, the local loss function is established by using the Euclidean distance between the output of guided and base blocks at the i-th layer. Such a local loss acts like a companion objective [58] to the hidden layers, which provides an additional constraint (or a new regularization) in the learning process. And by using this local loss, the vanishing gradient can be further suppressed. Inspired by the symmetrical cost function of denoising autoencoder in ladder network [35], we extend it to construct an asymmetrical ladder, in which only the parameters of student network are required to be learned from a fixed teacher network, and the asymmetric connections between the teacher network and the student network with a different depth are directly formed. Therefore, for the decomposed low-rank network, the local loss function can be written as follows:

$$\mathcal{L}^i_{\text{local}} = E(\mathcal{O}^i_s, \mathcal{O}^i_t) = \frac{1}{m_i}\left\|\mathcal{O}^i_s - \mathcal{O}^i_t\right\|^2_F, \tag{14}$$

$$\mathcal{O}^i_s = h(\mathcal{X}; \mathcal{W}^i_s), \quad \mathcal{O}^i_t = g(\mathcal{X}; \mathcal{W}^i_t), \tag{15}$$

where $\mathcal{O}^i_s, \mathcal{O}^i_t \in \mathbb{R}^{H^i \times W^i \times C^i}$ are the outputs of guided and base blocks at the i-th layer with the dimension of m_i by $H^i \times W^i \times C^i$, respectively. h and g are two functions that map an input $\mathcal{X}$ at the i-th layer to the guided and base blocks, respectively. $\mathcal{W}^i_s$ and $\mathcal{W}^i_t$ stand for the parameters at the i-th layer in the student and teacher networks, respectively. Note that the parameters of the student network at the i-th layer contain two low-rank factors $\mathcal{W}^{P_i}_s$ and $\mathcal{W}^{Q_i}_s$ by LRD, as shown in Fig. 3(b).

The local loss function in Eq. (14) is more simple and effective than hint-based loss function in FitNets [32]. On the one hand, the proposed loss function directly transfers the local knowledge from teacher network to student network without any regressors, which can significantly reduce learned parameters. On the other hand, the proposed loss function is constructed between the base and guided block, which can learn more effective feature representation from teacher network, comparing to the hint loss after a regular convolutional layer in FitNets.

Step (2). Knowledge fusion. Integrating the global knowledge with the local knowledge described above, we train our student network by minimizing the overall loss function as follows:

$$\begin{aligned}\mathcal{L}(\mathcal{W}_s) &= \mathcal{L}_{\text{global}} + \sum_{i=1}^{L}\lambda_i \mathcal{L}^i_{\text{local}} \\ &= \lambda\mathcal{H}(q^\tau_s, q^\tau_t) + \mathcal{H}(l, q_s) + \sum_{i=1}^{L}\frac{\lambda_i}{m_i}\left\|\mathcal{O}^i_s - \mathcal{O}^i_t\right\|^2_F,\end{aligned} \tag{16}$$

3. If there is no batch normalization or local response normalization, the corresponding operations in both the base and guided blocks should be deleted.

TABLE 1
Comparison on the speedup/compression rates and classification error on MNIST for LeNet. The batch size is set to be 100 (Values are averaged over 5 runs in all figures and tables of this paper).

Method	Para.		LeNet		
	Conv.	FC.	Compr.	Speed.	Err. ↑
SSL [11]	-	-	-	4.13×	0.12%
Pruning [25]	-	-	12×	-	-0.11%
Data-free [27]	-	-	8.33×	-	1.07%
LRDFT	3	23	16.98×	4.5×	0.14%
LRDDK	3	23	16.98×	4.5×	0.09%
RKT	3	23	16.98×	4.5×	0.11%
LRDKT	3	23	16.98×	4.5×	0.06%

where $\mathcal{H}$ refers to the cross-entropy loss, which is described in knowledge distillation. L indicates the guided/base layer that we select, and λ_i, $(i = 1, 2, \cdots, L)$ is a set of penalty parameters for balancing the global loss and each i-th local loss. The advantage of integrating both local and global supervision is quite evident:

1. The local loss function acts as a strong "regularization" for classification accuracy, which improves the convergence of the student network.
2. The global loss function makes it more convenient to exploit the significant performance gains, rather than using the original softmax-loss function.

A stochastic gradient descent is further employed via back-propagation to obtain the local minimum of the loss function in Eq. (16) by an end-to-end training. In practice, we set every penalty parameter of local loss to be equal. As for the sensitivity of hyper-parameters λ_i and λ, we will discuss the detailed setups in our experiments in Sec. 4 subsequently.

Especially, to verify the performance of the proposed LRDKT scheme that combines both closed-form low-rank decomposition and knowledge transfer training, we compare the LRDKT with our alternative schemes, including:

1. The combination of the proposed low-rank decomposition and fine-tuning (LRDFT).
2. The combination of the proposed low-rank decomposition and dark knowledge (LRDDK).
3. The combination of random initialized weights and knowledge transfer (RKT).

4 EXPERIMENTS

To evaluate the performance of the proposed LRDKT scheme, we conduct comprehensive experiments on both MNIST and ImageNet 2012 datasets [36]. We deploy the proposed LRDKT on four widely-used CNNs (*a.k.a.* LeNet on MNIST [4], AlexNet [1], VGG-16 [2] and ResNet-50 [6] on ImageNet), with comparisons to a group of state-of-the-art schemes proposed very recently. The proposed LRDKT is implemented by Caffe [59]. All the pre-trained CNNs except LeNet are taken from the Caffe model zoo[4].

Evaluation Protocols. We quantize the performance by using the number of parameters (or parameter compression rate), CPU/GPU speedup rate, and Top-1/5 classification error. To make a fair comparison, the CPU and GPU speedup rates are measured

4. https://github.com/BVLC/caffe/wiki/Model-Zoo

TABLE 2
The amounts of parameters and FLOPs, computation time on CPU (ms), GPU (ms) in both convolutional and fully-connected layers, and classification error rates (Top-1/5 Err.) of AlexNet, VGG-16 and ResNet-50 with batch size 32, 32 and 16, respectively.

Model	#param.			FLOPs			CPU (ms)	GPU (ms)	Top-1 err.	Top-5 err.
	Conv	FC	Total	Conv	FC	Total				
AlexNet	2.85M	58.62M	61.47M	671.08M	58.53M	729.61M	950	36	42.27%	19.11%
VGG-16	14.71M	123.64M	138.35M	15.36B	0.12B	15.48B	10,846	322	31.66%	11.55%
ResNet-50	23.23M	2M	25.23M	3.86B	2M	3.86B	2,266	122	24.64%	7.76%

in a single-thread Intel Xeon E5-2620 CPU and NVIDIA GTX TITAN X GPU, respectively.

Rank Analysis. In light of various low-rank decomposition methods, without losing the generalization, we adopt PCA [60] to select the suitable rank R and r in the convolutional layers and fully-connected layers, respectively. We consider the rank selection in the convolutional layers, the procedure of which is the same for the rank selection in fully-connected layers. Specifically, the filter tensor $\mathcal{K}$ of size $d \times d \times C \times N$ can be reshaped to a matrix $\mathbf{K} \in \mathbb{R}^{ddC \times N}$, the column of which are the reshaped filters $\mathbf{K}_n (\forall n \in [1, \cdots, N])$. In this paper, the PCA energy is defined as $E_R = \sum_{j=1}^{R} \sigma_j$, where σ_j is the j-th largest eigenvalue of covariance matrix $\frac{\mathbf{K}^\top \mathbf{K}}{ddC-1}$. In particular, the total PCA energy E_N is calculated by $E_N = \sum_{j=1}^{N} \sigma_j$. Under the constraint of PCA energy ratio $\frac{E_R}{E_N}$ (*e.g.*, $\frac{E_R}{E_N} \geq 0.95$), lower-rank approximation can be obtained if the minimal rank R is smaller. In the experiments, we select 5 groups of rank R in all convolutional layers as the minimal rank R, such that the PCA energy ratio in each layer is greater than 0.95, 0.7, 0.5, 0.3 and 0.2, respectively.

5 MNIST

MNIST is an image classification benchmark [4] which contains a training set of 60,000 and a test set of 10,000 28×28 gray-scale handwritten digit images from 10 classes. In this dataset, LeNet is adopted as the baseline model, which is trained from scratch without data augmentation and achieves an error rate of 0.89% on the MNIST dataset.

For LRDFT, LRDDK, RKT and LRDKT, we speedup and compress the second convolutional layer and the first fully-connected layer in LeNet, instead of operating over the whole network. To explain, these two layers consume over 87% of the evaluation times and occupy about 99% of the model storage. Correspondingly, the ranks in the second filter and the first fully-connected are obtained by setting PCA energy ratio greater than 20%, the values of which are 3 and 23, respectively. The hyper-parameters of λ and λ_i are set to be 0.003 and 0.0005, respectively. We use the Top-1 classification error to evaluate the accuracy degeneration of different baselines. The results of LeNet in MNIST are shown in Table 1.

Comparing to all baselines, the proposed LRDKT scheme achieves the highest speedup and compression rates with minimal performance degradation. To explain, LRDKT first employs low-rank decomposition to significantly increase both speedup and compression rates, whilst providing better initial values for the student network. Then, the accuracy loss between the teacher network and the student network is significantly reduced by knowledge transfer. In fact, LRDKT is more effective for deeper networks (*e.g.*, AlexNet, VGG-16 and ResNet-50) as presented in Sec. 5.1, which demonstrates the effectiveness of combining closed-form low-rank decomposition and knowledge transfer.

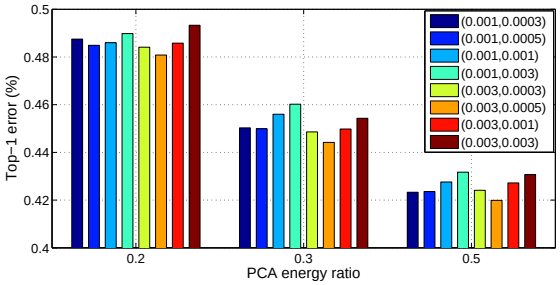


Fig. 4. Sensitivity of λ and λ_i on AlexNet. Number pairs in the top right refer to the group settings of (λ, λ_i).

5.1 ImageNet

We evaluate the proposed LRDKT scheme on ILSVRC 2012 classification benchmark [36], which contains a training set of more than 1.2 million images from 1,000 object classes, as well as a validation set of 50,000 images. Each image is associated with one ground truth class. We train the proposed model on the training set, over which we remove the pre-pixel mean for image preprocessing, and test it on the validation set using the single-view testing (central patch only). Additionally, we implement LRDKT on three CNNs *i.e.*, AlexNet [1], VGG-16 [2] and ResNet-50 [6]. The computation time and storage overhead of all these networks are shown in Table 2, together with their classification error on the ILSVRC 2012 dataset.

To train the proposed LRDKT scheme, we use the SGD solver with a learning rate starting at 0.001, dropped by a constant factor of 10 throughout 20 epochs, 5 epochs and 5 epochs on 60 epochs, 15 epochs and 15 epochs for AlexNet, VGG-16 and ResNet-50, respectively. The weight decay is set to be 0.0005 and the momentum is set to be 0.9. The batch sizes used to train the student net of AlexNet, VGG-16 and ResNet-50 are 256, 32 and 16, respectively. All the models in experiments are trained on NVIDIA GTX 1080Ti graphics card with 11GB and 128G RAM. For knowledge distillation and the proposed KT, we use two GPUs to train the corresponding model, in which one is for forward-propagation in the teacher net, and the other is for forward-propagation and back-propagation in the student net. Moreover, the total maximal GPU memory consumption of AlexNet, VGG-16 and ResNet-50 are 9,465MB, 14,356MB, 16,602MB, respectively. The hyper-parameters λ and $\lambda_i (i = 1, \cdots, L)$ control the balance of the global loss and each local loss, respectively. By observing the changes in the loss function, we set λ to be two different values, *i.e.*, 0.001 and 0.003, and set λ_i with the same value for all i to be four different values, *i.e.*, 0.0003, 0.0005, 0.001 and 0.003. For the temperature τ, we fix it to be 1, which can achieve good results.

In order to select the best λ and λ_i, we first analyze the results of these hyper-parameters settings in the convolutional layers with high computation complexity (*e.g.*, Conv2, Conv3 and Conv4 in

TABLE 3
Comparison of the LRE and the proposed low-rank decomposition in the convolutional layers with high computation complexity of AlexNet. R/N: low rank R over full filter numbers N. Top-1/5 err./+FT: the Top-1/5 error without fine-tuning or with fine-tuning.

Method	PCA energy ratio	Conv2	Conv3	Conv4	Time: initialized factors	Top-1 err./+FT	Top-5 err./+FT
LRE [15]	0.95	143/256	273/384	320/384	30 epochs≈99,000(s)	44.37%/43.12%	20.87%/19.87%
	0.7	70/256	146/384	203/384	30 epochs≈91,800(s)	44.86%/43.32%	21.78%/20.08%
	0.5	26/256	52/384	78/384	30 epochs≈84,600(s)	50.14%/44.06%	25.74%/20.59%
	0.3	12/256	24/384	33/384	30 epochs≈81,000(s)	60.87%/49.54%	35.49%/25.48%
	0.2	7/256	14/384	18/384	30 epochs≈79,200(s)	81.02%/50.23%	61.44%/25.90%
LRD	0.95	143/256	273/384	320/384	4(s)	43.00%/42.36%	19.67%/18.98%
	0.7	70/256	146/384	203/384	3.9(s)	45.22%/42.21%	22.08%/19.19%
	0.5	26/256	52/384	78/384	3.58(s)	87.06%/42.92%	72.36%/19.84%
	0.3	12/256	24/384	33/384	3.38(s)	99.50%/48.88%	98.18%/24.31%
	0.2	7/256	14/384	18/384	3.37(s)	99.78%/49.67%	99.30%/25.62%

TABLE 4
Comparison of fine-tuning epoch numbers using LRE and LRD with the same classification error on AlexNet.

FT epoch in methods		PCA energy ratio	Top-1 err.
LRE [15]	LRD		
25	15	0.5	44.69%
24	17	0.3	50.58%
25	20	0.2	50.68%

AlexNet), as presented in Fig. 4. We can see that the parameter (λ, λ_i) with a setting of (0.003, 0.0005) works the best. Therefore, we take a group of (0.003, 0.0005) as the final parameter setting for the LRDKT scheme.

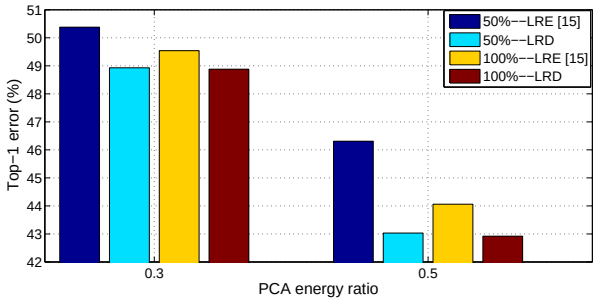


Fig. 5. Comparison of fine-tuning with the 50% and 100% number of training data on AlexNet.

5.1.1 Ablation Study

The core idea of LRDKT lies in both closed-form low-rank decomposition (LRD) and knowledge transfer (KT). In this subsection we evaluate them on the convolutional layers of AlexNet and VGG-16 for ablation study, respectively.

On the Importance of Low-rank Decomposition with a Closed-form Solution: To evaluate the importance of the proposed low-rank decomposition, we compare it with the cutting-edge low-rank decomposition methods (*e.g.*, the data-dependent (iteration based) low-rank decomposition (LRE) [15] and the Tucker-2 decomposition (TD) [45]) in three aspects, *i.e.*, time, classification error, and generalization ability. Different from LRE and LRD with two low-rank factors and one rank, TD decomposes the filter $\mathcal{K}$ of size $d \times d \times C \times N$ into three smaller filters of size $1 \times 1 \times C \times R_1, d \times d \times R_1 \times R_2$ and $1 \times 1 \times R_2 \times N$, where R_1, R_2 are two ranks. We determine rank R_1 and R_2 by PCA on mode-3 matricization (of size $C \times Nd^2$) and mode-4 matricization (of size $N \times d^2C$) of filter $\mathcal{K}$, respectively.

First, the closed-form low-rank decomposition performs better than LRE in the convolutional layers in AlexNet, as shown in Table 3. Comparing to LRE, through fine-tuning with the same training parameters, LRD requires extremely less time to obtain initialized low-rank factors (*i.e*, less than 4s in LRD *vs.* more than 79,000s in LRE on AlexNet), but with lower Top-1 and Top-5 classification errors for all PCA energy ratios (*e.g.*, at PCA energy ratio of 0.5, the gap between LRD and LRE is 1.14% Top-1 error on AlexNet). To explain, LRD provides better data-independent initialization to the compressed models. In addition, comparing to LRE, LRD requires a smaller number of fine-tuning epochs to restore the same classification error, as shown in Table 4.

Second, Fig. 5 shows that LRD has much stronger generalization ability than LRE, in which the low-rank model is fine-tuned to significantly restore the classification accuracy only by a part of training data. For example, when PCA energy ratio is set to be 0.5, LRD achieves extremely lower Top-1 classification error than LRE using 50% training dataset, *i.e.*, 43.03% *vs.* 46.31% in Top-1 error. Moreover, comparing to using the whole training data, LRD performs significantly better with a negligible (*e.g.*, 0.11%) increase in Top-1 error *vs.* an increase of 2.25% Top-1 error in LRE.

Third, Table 5 shows that LRD is most effective to be integrated into KT, which results in a significant increase of classification accuracy over LRE, TD and random initialization. For instance, LRD achieves the best performance by a factor of $5.33\times$ with 44.48% Top-1 error and 21.21% Top-5 error at the PCA energy ratio of 0.5, which improves from LRE by a factor of $5.33\times$ with 46.03% Top-1 error and 22.62% Top-5 error, and improves from TD by a factor of $5.04\times$ with 45.30% Top-1 error and 22.05% Top-5 error. It is note that TD achieves the highest speedup at the same PCA energy ratio, due to the internal decomposition to three smallest filters. Interestingly, KT with random weights achieves a comparable classification error to LRE. For example, comparing to Random, LRE achieves a lower Top-1 classification error (42.78% *vs.* 43.03%) at the PCA energy ratio of 0.5, and LRE achieves a higher Top-1 classification error (46.03% *vs.* 45.89%) at the PCA energy ratio of 0.3. To explain, the local and global knowledge in KT are able to help the model with random initialization, which is compressed only on the partial layers to restore the classification accuracy in distributed representations [61]. However, the random criterion is not robust

TABLE 5
Results of integrating LRE, TD and LRD into KT in the convolutional layers with high computation complexity of AlexNet. R/N and $(R_1, R_2)/N$: low rank R of LRE and LRD, and (R_1, R_2) of TD over filter number N, respectively.

Method	PCA energy ratio	Conv2	Conv3	Conv4	CPU speedup	Top-1 err.	Top-5 err.
LRE [15]+KT	0.5	26/256	52/384	78/384	3.94×	42.78%	19.73%
	0.3	12/256	24/384	33/384	5.33×	46.03%	22.62%
TD [45]+KT	0.5	(8,26)/256	(31,52)/384	(73,78)/384	5.04×	45.30%	22.05%
	0.3	(4,12)/256	(13,24)/384	(30,33)/384	6.43×	51.27%	27.22%
Random+KT	0.5	26/256	52/384	78/384	3.94×	43.03%	20.01%
	0.3	12/256	24/384	33/384	5.33×	45.89%	22.56%
LRD+KT	0.5	26/256	52/384	78/384	3.94×	41.99%	19.05%
	0.3	12/256	24/384	33/384	5.33×	44.48%	21.21%

TABLE 6
Comparison of different training methods on VGG-16 with the initialization of LRD.

Method	PCA energy ratio	Top-1 err.
LRD+FT	0.3	34.04%
	0.2	36.81%
LRD+DK	0.3	33.47%
	0.2	36.76%
LRD+KT	0.3	32.39%
	0.2	35.72%

and results in large accuracy loss when compressing the whole network, as presented in Table 10, Table 11 and Table 12.

Knowledge transfer vs. Fine-tuning and Dark Knowledge: Knowledge transfer is to transfer both global and local knowledge from the teacher network to the student network. Table 6 compares the performances of KT (with both global and local knowledge from teacher net), FT (without both global and local knowledge) and DK (with only global knowledge). The evaluations are performed under two smallest PCA energy ratios (*i.e.*, 0.2 and 0.3) on VGG-16. It is clear that integrating local knowledge into global knowledge consistently boosts the classification accuracy on VGG-16. Especially, when the PCA energy ratio is set to be 0.3, the compressed model trained by KT outperforms the FT and DK by a significant margin with 1.65% and 1.08% Top-1 error (*i.e.*, 32.39% Top-1 error in KT *vs.* 34.04% Top-1 error in FT and 33.47 Top-1 error in DK), which shows the importance of using both global and local knowledge.

5.1.2 Results in All Convolutional Layers

We further explore the speedup performance of the proposed LRDKT in all convolutional layers on both AlexNet and VGG-16 with comparisons to other CNN acceleration methods, which are shown in Table 7 and Table 8, respectively.

For AlexNet, we construct two local loss functions after the 2nd and final pooling layers instead of each convolutional layer. As shown in Table 7, XNOR [19] can speedup the convolutional layers by a factor of $9.42\times$ on GPU, which is the highest speedup rate compared to all the baselines. To explain, both the filters and the input to convolutional layers[5] are binary, and therefore XNOR approximates convolutional operation using XNOR-Bitcounting operations (without multiplication), which increasingly accelerates the original network. However, the quantization error in these

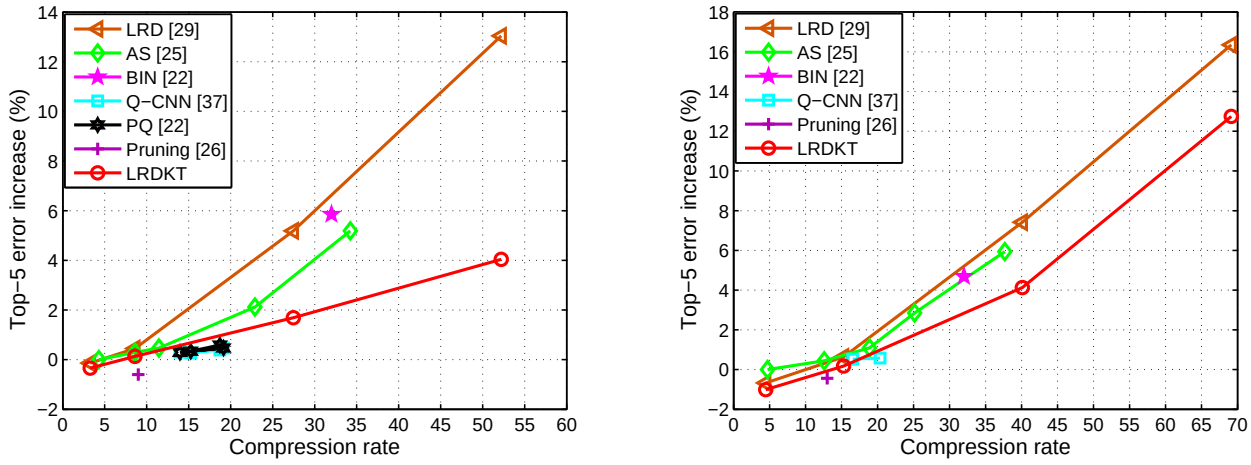
TABLE 7
Speedup all convolutional layers on AlexNet. SSL* is the previous published results by Wen *et al.* [11].

Method	PCA energy ratio	CPU speedup	GPU speedup	Top-1 err. ↑	Top-5 err. ↑
LRE [15]	0.7	2.24×	1.78×	1.32%	1.21%
	0.5	3.06×	2.13×	2.18%	1.79%
	0.3	4.32×	2.46×	7.76%	6.67%
	0.2	5.46×	3.56×	9.69%	7.29%
CPD [16]	-	3.27×	2.36×	3.47%	1.86%
XNOR [19]	-	-	9.42×	12.45%	11.06%
SSL [11]	-	4.18×	2.78×	3.42%	2.96%
SSL* [11]	-	5.06×	3.07×	2.42%	-
L1 [12]	-	2.89×	2.10×	2.87%	1.67%
LRDKT	0.7	2.24×	1.78×	-0.24%	-0.53%
	0.5	3.06×	2.13×	0.57%	0.31%
	0.3	4.32×	2.46×	3.36%	2.62%
	0.2	5.46×	3.56×	5.72%	4.76%

binary methods is also very high. For L1 [12], we prune 50% filters in each convolutinoal layers that are selected by a simple ℓ_1 norm, and achieve a factor of $2.89\times$ on CPU with an increase of 2.87% Top-1 classification error[6]. Comparing to directly removing the whole filters based on ℓ_1 norm [12], SSL [11] is able to learn a compact structure from the original DNNs during training, which achieves a higher speedup by $4.18\times$ on CPU, but with an increase of 3.42% Top-1 classification error based on our implementation. The reported results of SSL* [11] achieves a better performance based on multiple physical threads in CPU, specific libraries on GPU (*e.g.*, cuSPARSE) and speed evaluation on average layer-wise acceleration. In contrast, our implementation of SSL is based on a total-layer acceleration evaluated on a single thread in CPU, and without the tools for supporting sparse matrix calculation on GPU. To explain such a difference, average layer-wise acceleration is a relatively unfair speed evaluation that ignores the calculation time for data memory access across layers. For LRE [15], we set different PCA energy ratios to obtain different speedup rates and different classification error increases, which is similar to LRDKT. As the speedup increases, we achieve consistent increases of the classification error, which is significant (*e.g.*, from an increase of 2.18% Top-1 error to 7.76% Top-1 error), when the PCA energy ratio drops from 0.5 to 0.3. We observe the similar result in LRDKT. Since the selected rank in each layer is greatly reduced from PCA energy ratio 0.5 to 0.3, the loss of accuracy cannot be restored effectively. Comparing to LRE [15], our method

5. Since XNOR-Net has not binarized the first convolutional layer and the last fully-connected layer, their actual speedup is much smaller than $32\times$ layer-wise speedup on a 32-bit machine.

6. Li *et al.* [12] used their own framework to achieve compression, which may result in a different accuracy due to the different fine-tuning framework and parameter setting in their paper.



(a) AlexNet (b) VGG-16

Fig. 6. Comparison on the compression rate and the increase of Top-5 error for compressing all the fully-connected layers in AlexNet and VGG-16.

TABLE 8
Speedup all convolutional layers on VGG-16. LRE* and L1* are the results based on the implementation of [25] and [39], respectively.

Method	PCA energy ratio	CPU speedup	GPU speedup	Top-1 err. ↑	Top-5 err. ↑
LRE [15]	0.3	3.0×	2.90×	15.14%	10.45%
L1 [12]	-	1.99×	2.51×	2.51%	0.96%
TE [38]	-	2.38×	2.89×	-	3.94%
Thinet-C [13]	-	1.99×	2.51×	-1.46%	-1.09%
LRE* [15]	-	3.8×	2.90×	-	9.7%
L1* [12]	-	2×	-	-	0.8%
	-	4×	-	-	8.6%
LRDKT	0.7	2.46×	2.33×	-0.57%	-0.79%
	0.5	2.63×	2.64×	1.93%	0.85%
	0.3	3.00×	2.90×	7.81%	5.89%

achieves much smaller increase of classification error with the same speedup rate. Comparing to CPD [16], our method achieves both smaller increase of classification error and higher speedup. For example, with the smaller increase of Top-1 error (3.36% in LRDKT *vs.* 3.47% in CPD), we attain a CPU speedup of $4.32\times$ and GPU speedup of $2.46\times$ (for CPD the corresponding CPU speedup is $3.27\times$ and GPU speedup is $2.36\times$).

For VGG-16, we construct two local loss functions after the 2nd and final pooling layers instead of each convolutional layer using LRDKT. As shown in Table 8, TE [38] uses Taylor expansion based pruning to iteratively remove a filter that has least influence on the loss function, and then fine-tunes the rest network. Although the compressed model of TE achieves higher speedup, the calculating and re-training procedure can be prohibitively costly. Comparing to TE, ThiNet-C [13] prunes 50% filters in each convolutional layer and can improve the classification accuracy, with a slight reduction of CPU and GPU speedup. As for L1 [12], L1 achieves the increases of 0.8% and 8.6% Top-5 error by a factor of $2\times$ and $4\times$ CPU speedup based on the implementation of [39]. Alternatively, we re-implement it and simply prune the same number of filters in each layer to ThiNet-C for a fair comparison, which achieves a relatively consistent result by a factor of $2\times$ CPU speedup. However, L1 still achieves a higher increase of classification error by a lower or same CPU speedup, comparing to LRDKT or ThiNet-C. To explain, due to the nonlinear transformation in the network, the filters with a small ℓ_1 norm are still likely to be important, which have large impacts on the final loss function. Comparing to ThiNet-C, we can achieve a comparable result by a factor of higher GPU speedup ($2.33\times$ *vs.* $1.99\times$), but with a slightly higher increase of classification error (*e.g.*, -0.79% *vs.* -1.09% in an increase of Top-5 error). We re-implement LRE at PCA energy ratio of 0.3, which achieves a slightly higher increase of Top-5 error than LRE* (*i.e.*, implemented by Zhang *et al.* [25]). This inconsistency is mainly caused by the different filter selection on each layer and fine-tuning framework. Comparing to both LRE and LRE*, we achieve a lower increase of Top-5 error (*i.e.*, 10.45% in LRE, 9.7% in LRE* *vs.* 5.89% in LRDKT) with the same GPU speedup.

5.1.3 Results in All Fully-connected Layers

In Fig. 6, we also evaluate the performance of LRDKT for compressing all fully-connected layers in both AlexNet and VGG-16. Instead of using each local loss function after each fully-connected layer, we only construct one local loss function after the second fully-connected layer in AlexNet and VGG-16. For LRD [29], we use the same rank selection criterion with LRDKT. For BIN [22], which has no parameter to tune, the compression rate is fixed as 32. For the parameter setting of PQ [22] and Q-CNN [37] in AlexNet, we vary the segment dimension and the number of sub-codewords in each subspace among (2,16), (3,16), (3,32) and (4,32).

From Fig. 6(a), we discover that LRDKT significantly improves the classification accuracy compared to LRD and AS [25], especially in the case of high compression rate. Note that PQ and Q-CNN have achieved better performance comparing to that of LRD, AS and LRDKT. However, it is hard to achieve high compression rate, due to the limited codebook size. In contrast, LRDKT achieves the best trade-off between compression rate and classification accuracy by a factor of $27.46\times$, with an increase of 1.69% Top-5 classification errors and a factor of $8.60\times$ with a negligible loss in classification accuracy. The simplest BIN works well when we fix the compression rate to 32. It is noted that, the scheme of pruning nodes [26] also works well when the compression rate is set to be less than $10\times$.

For PQ in VGG-16, we fix the number of sub-codewords to 32 in each subspace and vary the segment dimension s between 3 and 4. From Fig. 6(b), a consistent trend is also observed in

TABLE 9
Comparison of FitNets and LRDKT for compressing VGG-16. FitNet:hint-$num/num-num-num$ means training FitNet by adding one hint loss or three hint losses at the corresponding num-th or $num-num-num$-th layer with initialization of LRD.

Method	PCA energy ratio	Top-1 err.	Top-5 err.
FitNet:hint-4	0.7	45.57%	21.63%
	0.5	54.29%	27.64%
FitNet:hint-7	0.7	31.66%	11.08%
	0.5	38.88%	15.81%
FitNet:hint-10	0.7	32.03%	11.16%
	0.5	39.78%	16.14%
FitNet:hint-13	0.7	32.33%	11.42%
	0.5	42.66%	18.43%
FitNet:hint-4+7+10	0.7	31.53%	11.03%
	0.5	37.86%	15.28%
LRDKT	0.7	31.16%	10.84%
	0.5	35.77%	13.90%

AlexNet. To sum up, LRDKT achieves the best trade-off between compression rate and Top-5 classification error by a factor of $40.11\times$, with an increase of 4.13% Top-5 classification error, as well as a factor of $15.28\times$ with an increase of 0.18% Top-5 classification error.

5.1.4 Results in the Whole Network

We further evaluate the performance of the proposed LRDKT scheme in the whole network on AlexNet, VGG-16 and ResNet-50. The proposed LRDKT scheme is compared with state-of-the-art CNN compression methods (*e.g.*, FitNets [32], XNOR [19], L1 [12] and ThiNet [13]), as well as our 3 alternative schemes (*i.e.*, LRDFT, LRDDK, and RKT). Instead of using each local loss function after each layer, we construct two local loss functions after the final pooling layer and the 2nd fully-connected layer for AlexNet and VGG-16, as well as building two local loss functions after the 1st pooling layer and a specific residual block output (*i.e.*, res4f_relu[7]) for ResNet-50. As we could not find applications of FitNets on ImageNet in the literature, we re-implement FitNets to compress AlexNet and VGG-16 on ImageNet. For the first stage of FitNets, hint-based training is implemented during 30 epochs and 15 epochs on AlexNet and VGG-16, respectively. For a fair comparison, we choose the decomposed low-rank networks as our student networks, which are same to LRDKT. Furthermore, We choose the middle layer of student network and teacher network as the hint and guided layer. The learning rate stated at 0.001 initially, which was dropped by a constant factor of 10 after 10 epochs and 5 epochs on AlexNet and VGG-16, respectively. For the second stage of FitNets, we use the same training setting as LRDKT, except that τ is set to 5.

Fig. 7 represents the results of FitNets, LRDKT and three alternative schemes for compressing AlexNet. The results reveal four key observations: (1) The hint training losses of different layers in different versions of FitNet have large impacts on the performance of student network. For the design of using only one hint training loss, the loss applied to the 5th layer achieves the lowest Top-1 error by 44.24% and 42.13% at PCA energy ratio of 0.5 and 0.7, respectively. (2) By deploying over an LRD based initialization, we test different versions of FitNet by adding three losses on three intermediate layers, one of which

7. The name of layer is defined by Kaiming He on Caffe: https://github.com/KaimingHe/deep-residual-networks.

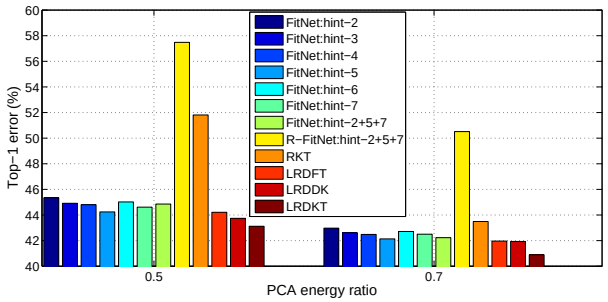


Fig. 7. Comparison of FitNets, LRDKT and three alternative scheme for compressing AlexNet. FitNet:hint-$num/num-num-num$ means training FitNet by adding one hint loss or three hint losses at the corresponding num-th or $num-num-num$-th layer with initialization of LRD. R-FitNet is training FitNet from scratch.

performs at the 5th layer. We can see that the one-loss FitNet at the 5th layer outperforms the three-loss FitNet. (3) We then train FitNet from scratch (*i.e.*, R-FitNet) by adding only one hint training loss. However, it simply cannot converge. To address this problem, we add three losses on the 2nd, 5th and 7th layers, which achieve 57.48% and 50.51% Top-1 error at PCA energy ratio of 0.5 and 0.7, respectively. However, the performance of R-FitNet is far inferior to LRD+FitNet with the same number of losses. To explain, initial weights provided by LRD enable the low-rank network to obtain a better local minima during the training. Interestingly, comparing to R-FitNet, the proposed RKT achieves lower Top-1 error, which is due to the effectiveness of KT for transferring both global and local knowledge from the teacher network. (4) The proposed LRDKT scheme is the most effective scheme to compress AlexNet among LRDKT, FitNets and their alternative schemes. As shown in Fig. 7, comparing to FitNet with one-loss at the 5th layer, LRDKT achieves better performance both at PCA energy ratio of 0.5 and 0.7 (*e.g.*, 40.89% Top-1 error in LRDKT *vs.* 42.13% Top-1 error in FitNet at PCA energy ratio of 0.7), respectively.

Table. 9 represents the results of FitNets and LRDKT for compressing VGG-16. We observe that one hint loss applied to the 7th layer achieves the lowest Top-1 error and Top-5 error among all designs of one loss. Different from compressing AlexNet, FitNet with three losses on three intermediate layers (*e.g.*, FitNet:hint-4+7+10) performs better than the one-loss FitNets. For example, comparing to FitNet:hint-7, FitNet:hint-4+7+10 achieves a lower Top-1 error (37.86% *vs.* 38.88%) at the PCA energy ratio of 0.5. Finally, equipped with a simpler local loss by an end-to-end training, LRDKT significantly reduces the classification error of low-rank network, and performs much better than FitNets along with the complex regressors in the two-stage training. For example, comparing to the best FitNet (*i.e.*, FitNet:hint-4+7+10) amongst its variants, LRDKT achieves lower Top-1 error at PCA energy ratio of 0.5 and 0.7 (*i.e.*, 35.77% and 31.16% *vs.* 37.86% and 31.53%), respectively.

Table 10, Table 11 and Table 12 show the performances compared against state-of-the-art methods on AlexNet, VGG-16 and ResNet-50, respectively. For Alexnet, XNOR [19] still achieves the highest speedup rate by a factor of $8.00\times$ GPU speedup, compared to all the baselines, as shown in Table 10. However, large quantization error with binary weights and activations leads to the high increase of classification error, *e.g.*, an increase of 13.56% Top-1 classification error. By pruning the 50% filters in

TABLE 10
Comparison of different methods on compressing the whole AlexNet. "Rank" refers to PCA energy ratio.

Method	Rank	#Param.	CPU speedup	GPU speedup	Top-1 Err. ↑	Top-5 Err. ↑
XNOR [19]	-	-	-	8.0×	13.56%	11.69%
L1 [12]	-	16.3M	2.51×	2.00×	3.75%	2.69%
LRDFT	0.7	18.7M	2.04×	1.8×	-0.31%	-0.46%
	0.5	6.4M	2.93×	2.12×	1.94%	1.50%
	0.3	2.3M	4.22×	2.4×	13.70%	10.21%
LRDDK	0.7	48.0M	2.04×	1.8×	-0.34%	-0.50%
	0.5	6.4M	2.93×	2.12×	1.47%	1.16%
	0.3	2.3M	4.22×	2.4×	11.45%	9.17%
RKT	0.7	18.7M	2.04×	1.8×	1.22%	0.75%
	0.5	6.4M	2.93×	2.12×	9.54%	7.71%
	0.3	2.3M	4.22×	2.4×	17.95%	15.48%
LRDKT	0.7	18.7M	2.04×	1.8×	-1.38%	-0.89%
	0.5	6.4M	2.93×	2.12×	0.85%	0.56%
	0.3	2.3M	4.22×	2.4×	8.69%	6.53%
LRDKT-GAP	0.7	1.1M	2.22×	2.0×	3.01%	2.77%

TABLE 11
Comparison of different methods on compressing the whole VGG-16. "Rank" refers to PCA energy ratio.

Method	Rank	#Param.	CPU speedup	GPU speedup	Top-1 Err. ↑	Top-5 Err. ↑
L1 [12]	-	8.3M	2.03×	2.53×	5.86%	4.29%
ThiNet-GAP [13]	-	8.3M	2.03×	2.53×	1.00%	0.52%
ThiNet-Tiny [13]	-	1.3M	3.17×	4.35×	9.00%	6.47%
LRDFT	0.7	30.5M	2.43×	2.27×	-0.14%	-0.49%
	0.5	9.5M	2.61×	2.55×	5.52%	3.38%
	0.3	3.7M	3.00×	2.80×	20.09%	14.24%
LRDDK	0.7	30.5M	2.43×	2.27×	-0.34%	-0.55%
	0.5	9.5M	2.61×	2.55×	5.28%	2.95%
	0.3	3.7M	3.00×	2.80×	18.03%	12.18%
RKT	0.7	30.5M	2.43×	2.27×	5.36%	2.04%
	0.5	9.5M	2.61×	2.55×	12.02%	8.43%
	0.3	3.7M	3.00×	2.80×	21.97%	16.86%
LRDKT	0.7	30.5M	2.43×	2.27×	-0.50%	-0.71%
	0.5	9.5M	2.61×	2.55×	4.11%	2.35%
	0.3	3.7M	3.00×	2.80×	16.81%	10.89%
LRDKT-GAP	0.7	3.3M	2.46×	2.33×	0.18%	-0.12%

each convolutional layer and 50% nodes in each fully-connected layer, L1 [12] only achieves $3.77\times$ compression (*i.e.*, 16.3M number of parameters *vs.* 61.47M in the original AlexNet), $2.51\times$ CPU speedup and $2\times$ GPU speedup, but with an increase of 3.75% Top-1 error. Comparing to L1, LRDFT employs the proposed closed-form low-rank decomposition to initialize the network, followed by a simple fine-tuning, which achieves better performance with only 1.94% Top-1 error increase with a factor of $9.6\times$ compression, $2.93\times$ CPU speedup and $2.12\times$ GPU speedup. With the same closed-form low-rank decomposition, LRDKT achieves the lowest increase of classification error, comparing to LRDFT and LRDDK. For example, at PCA energy ratio of 0.5, LRDKT achieves an increase of 0.85% Top-1 error, while an increase of 1.94% Top-1 error and an increase of 1.47% Top-1 error in LRDFT and LRDDK, respectively. Note that RKT obtains a large increase of classification error, which is totally different from the result to compress the partial layers in Table. 5. To explain, random initialization of the whole layers leads to large instability to train a deep network. Even with local supervised information using KT, it is not robust to train the network to obtain a bad local minimum. To further compress the network, the traditional fully-connected layers are removed, and are replaced with a GAP (global average pooling [3]) layer. LRDKT-GAP achieves $55.88\times$ compression (*i.e.*, 1.1M parameters), $2.22\times$ CPU speedup and $2.0\times$ GPU speedup, with an increase of 3.01% Top-1 error.

For VGG-16, L1 [12] and ThiNet [13] use the same GAP layer to remove the fully-connected layers to obtain a high compression rate. As shown in Table 11, comparing to L1 with an increase of 5.86% Top-1 error, ThiNet-GAP achieves better performance with an increase of 1.00% Top-1 error (by pruning the same 50% filters in each convolutional layer) by a factor of $16.67\times$ (*i.e.*, 8.3M parameters), and $2.53\times$ CPU speedup and $2.03\times$ GPU speedup. Although LRDKT and its variants LRDFT and LRDDK achieve lower Top-1 error than ThiNet-GAP at PCA energy ratio of 0.7, they require much more storage (*i.e.*, 30.5M parameters). By employing the same GAP layer, we further compress the network by a factor of $41.92\times$ compression (*i.e.*, 3.3M parameters) and $2.46\times$ CPU speedup in LRDKT-GAP, but with only 0.18% Top-1 error increase, which is much better than ThiNet-GAP. Note that ThiNet-Tiny prunes 75% filters in each convolutional layer, and accelerates CPU and GPU inference greatly, *i.e.*, $3.17\times$ CPU speedup and $4.35\times$ GPU speedup. However, ThiNet-Tiny significantly reduces the discriminability of the pruned network,

TABLE 12
Comparison of different methods on compressing ResNet-50. "Rank" refers to PCA energy ratio.

Method	Rank	#Param.	CPU speedup	GPU speedup	Top-1 Err. ↑	Top-5 Err. ↑
ThiNet [13]	-	16.94M	1.58×	1.42×	0.84%	0.47%
	-	12.38M	1.86×	1.69×	1.87%	1.12%
	-	8.66M	2.21×	2.03×	4.46%	2.84%
LRDFT	0.7	9.8M	2.02×	1.44×	2.36%	1.84%
	0.5	6.3M	2.26×	1.61×	8.52%	4.94%
LRDDK	0.7	9.8M	2.02×	1.44×	1.93%	1.31%
	0.5	6.3M	2.26×	1.61×	7.33%	4.76%
RKT	0.7	9.8M	2.02×	1.44×	5.33%	3.69%
	0.5	6.3M	2.26×	1.61×	14.01%	11.55%
LRDKT	0.7	9.8M	2.02×	1.44×	0.72%	0.38%
	0.5	6.3M	2.26×	1.61×	6.29%	3.74%

which is not suitable for lossless compression (*i.e.*, less than 1% increase of Top-1 error). In addition, comparing to three alternative schemes (*i.e.*, LRDFT, LRDDK and RKT), LRDKT still achieves the lowest classification error at the same PCA energy ratio.

For ResNet-50, each residual block contains three convolutional layers and shortcut connections. To perform the sum operator, the number of output feature maps in the last convolutional layer needs to be consistent with that of the projection shortcut layer. To match the dimension of the last convolutional layer and shortcut layer in each residual block, ThiNet only prunes the first two layers of each block with 3 different compression rates (*i.e.*, 70%, 50% and 30% filter preservation of the first two convolutional layers in each residual block), leaving the last output and projection shortcuts unchanged, which results in a limited compression. Unlike ThiNet, we compress the entire layers, which leads to higher compression rate. For example, we achieve a $4.00\times$ compression (*i.e.*, 6.3M parameters *vs.* 25.23M parameters in the original ResNet-50) at PCA energy ratio of 0.5, with an increase of 6.29% Top-1 error, which might affect the practical usage in high-precision applications. To achieve acceptable or no accuracy loss (*e.g.*, less than an increase of 1% Top-1 error), we reduce the PCA energy ratio to 0.7, and achieve much better performance comparing to ThiNet. For example, comparing to ThiNet, with a slight lower increase of Top-1 error (0.72% *vs.* 0.84%), LRDKT attains higher compression rate of $2.57\times$ (9.8M parameters) *vs.* $1.49\times$ (16.94M parameters) and CPU speedup of $2.02\times$ *vs.* $1.58\times$.

Improved Gradient Strength in Knowledge Transfer. We provide more insights into KT to support the reason why KT can

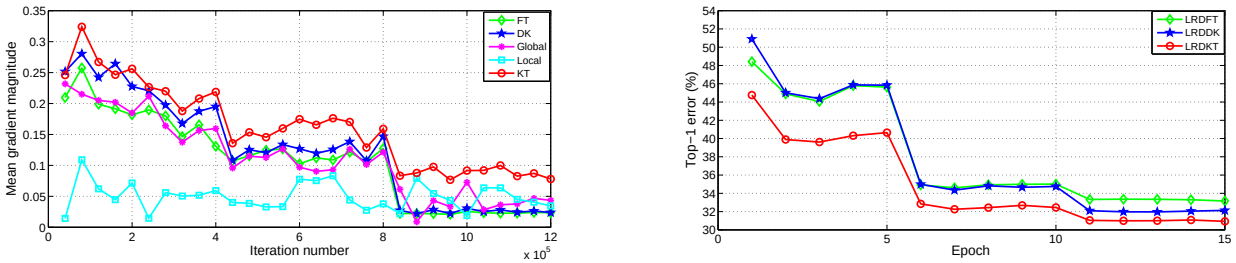


Fig. 8. The magnitude of mean absolute gradient of the first convolutional layer, and the trends of testing Top-1 error in the decomposed ResNet-50 at PCA energy ratio of 0.5.

TABLE 13
Comparison of different compressed model for fine-grained classification on CUB-200.

Method	PCA energy ratio	#param.	FLOPs	Top-1 err.
VGG-16	-	135.07M	15.48B	27.60%
LRDFT	0.5	9.5M	1.31B	30.77%
	0.3	3.7M	0.64B	41.53%
LRDDK	0.5	9.5M	1.31B	30.07%
	0.3	3.7M	0.64B	39.11%
LRDKT	0.5	9.5M	1.31B	28.50%
	0.3	3.7M	0.64B	36.82%

TABLE 14
The speedup for Faster R-CNN detection on different devices.

Device	PCA energy ratio	Speedup	mAP	Δ mAP
Titan X GPU	-	Baseline	68.7	-
	0.7	3.10	68.5	0.2
	0.5	3.44	66.9	1.8
Jetson TX2	-	Baseline	68.7	-
	0.7	2.51	68.5	0.2
	0.5	3.05	66.8	1.9

effectively address the vanishing gradients in the back propagation. Since the number of layers is deepened by low-rank decomposition, the local reconstruction error from hidden layers provides sufficient gradient information to compensate the problem that the global gradient doesn't flow from the top layer to the bottom layer. As a result, we expect to use KT to reduce the vanishing gradient in the back propagation. To quantitatively validate this effect, we compare the magnitude of gradients to the first convolutional layer in the decomposed ResNet-50 with fine-tuning/dark knowledge/knowledge transfer on the ImageNet 2012.

Fig. 8 shows the mean absolute gradients of the first convolutional layer and the trends of testing Top-1 error in the decomposed ResNet-50 at PCA energy ratio of 0.5. The gradients in KT are from two parts, *i.e.*, global loss and local loss. First, there are several significant drops in terms of the mean gradient magnitude by using FT, DK and KT, which are due to the scheduled reductions in learning rate by a factor of 10. At the beginning of training, the gradients are almost the same in FT, DK and KT. With the increase of iteration number, the gradients in FT and DK are significantly reduced with a large magnitude of gradients in DK, especially after 800K iterations. To explain, after 800K iterations, the gradients from the global loss function gradually diminish. Instead, the local loss provides relatively more gradients to properly train the network. Second, the improved gradient strength helps LRDKT to effectively train the student network and achieves much lower classification error, which is shown in the right panel of Fig. 8. This quantitative results support our claim that KT significantly reduces the problem of vanishing gradient, which enables the decomposed network to be trained more effectively.

5.2 Applications

LRDKT has demonstrated its effectiveness to compress and accelerate CNNs on MNIST and ImageNet ILSVRC 2012 classification tasks. We further evaluate the generalization ability of the compressed model for transfer learning, including domain adaptation and object detection. We also apply the compressed model to different embedded devices. For better discussion, we take the VGG-16 model as our baseline model.

5.2.1 *Domain adaptation*

We evaluate the ability of domain adaptation for the compressed VGG-16 at two PCA energy ratios (*i.e.*, 0.3 and 0.5). To better evaluate this ability, we select a public domain-specific dataset CUB-200 [62] for fine-grained classification. CUB-200 contains 11,788 images of 200 different bird species, which contains 5,994 training images and 5,794 testing images. To make a fair comparison, we fine-tune the LRDKT-based compressed models with the same hyper-parameters and epochs. The results of fine-grained classification are shown in Table 13.

The pre-trained VGG-16 is fine-tuned on the CUB-200 dataset, which achieves the lowest error (*i.e.*, 27.60% Top-1 error) but with the huge memory cost and slow inference speed (*i.e.*, 135.07M parameters and 15.48B FLOPs). We then fine-tune the compressed networks, which are first compressed in the ImageNet domain by LRDFT, LRDDK and LRDKT, respectively. As shown in Table 13, the model compressed by LRDKT achieves the best performance by an increase of only 0.9% Top-1 error (*i.e.*, 28.50% Top-1 error), with only 9.5M parameters and 1.31B FLOPs, *i.e.*, $14.22\times$ lower memory cost and theoretical $11.82\times$ faster inference speed than VGG-16.

5.2.2 *Acceleration for Detection*

We also evaluate the ability of transfer learning for the compressed VGG-16 models at two PCA energy ratios (*i.e.*, 0.5 and 0.7), which is deployed over Faster R-CNN [9] for object detections. We select the PASCAL VOC 2007 object detection benchmark to evaluate the performance of our models by mean Average Precision (mAP), which contains about 5K training/validation images and 5K testing images. In our experiments, we first compress the VGG-16 by LRDKT on the ImageNet, and then use the compressed models as the pre-trained models for Faster R-CNN with its default training settings.

The actual running time of Faster R-CNN is 189ms/image on Titan X GPU and 1,034ms/image on Jetson TX2 embedded

TABLE 15
Inference time on mobile device by testing on each image with a size of 240×240 by CPU on Intel Xeon E5-2620 *vs.* iphone 6s A9. LRDKT-a refers to the compressed model by LRDKT at PCA energy ratio of a.

Model	FLOPs	PC-time	ARM-time	Top-1 err.
VGG-16	15.48B	495ms	3,186ms	31.66%
LRDKT-0.7	2.43B	172ms	936ms	31.16%
LRDKT-0.5	1.31B	134ms	735ms	35.77%
LRDKT-0.3	0.64B	110ms	654ms	48.47%

device. Comparing to VGG-16, we get an actual detection time of 55ms with $3.44\times$ acceleration and 3.05ms with $3.5\times$ acceleration on Titan X and Jetson TX2, respectively. As shown in Table 14, comparing to VGG-16, we achieve 0.2% mAP drops with a factor of $3.10\times$ speedup on Titan X, and 0.2% mAP drops with a factor of $2.51\times$ speedup on Jetson TX2, which is still very practical for real-world application.

5.2.3 Speedup Evaluation on ARM Based Mobile Device

We further evaluate the actual speedup of the compressed VGG-16 using LRDKT on iphone 6s with TensorFlow Lite. As shown in Table 15, comparing to PC device on Intel Xeon 2620, the original VGG-16 on iphone 6s A9 achieves much slower running time for inference (3,186ms *vs.* 495ms). However, using the same compressed model by LRDKT, iphone 6s A9 achieves much more speed improvement than PC device on Intel Xeon 2620. For example, LRDKT-0.7 model evaluated on iphone 6s A9 achieves $3.40\times$ (936ms) speedup, which is higher on PC Intel Xeon 2620 (*i.e.*, $2.88\times$ speedup with 172ms inference time), while with 0.5% decrease in the Top-1 error. Therefore, CNN compression (*e.g.*, LRDKT) can obtain higher speedup rate to make it possible for real-time applications of the deep models, when directly being applied to the ARM based mobile device.

6 Conclusion

In this paper, we present a unified framework towards *holistic* and *explicit* compression of CNNs, which jointly compresses both the convolutional and fully-connected layers to simultaneously reduce time and memory. First, a low-rank decomposition (LRD) based inter-layer compression is proposed to remove redundancies across both convolution kernels and fully-connected matrices, with a novel closed-form solution to significantly surpass the efficiency of the existing iterative methods, Second, to deal with the performance degradation by compression, a novel knowledge transfer (KT) method is proposed to explicitly align output and intermediate responses from the teacher network to its student network, which involves operating nonlinear activations within and across all layers. The proposed KT minimizes both "local" and "global" reconstruction errors in a unified way that makes the approximated network much easier to converge and implement. We have demonstrated that the proposed LRDKT scheme can lead to state-of-the-art compression and speedup rates, whilst maintaining very comparable classification accuracy on LeNet for MNIST classification, as well as on AlexNet, VGG-16 and ResNet-50 for ImageNet ILSVRC 2012 classification.

Acknowledgment

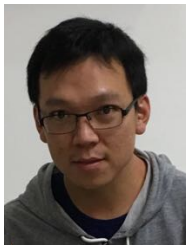
This work is supported by the National Key R&D Program (No.2017YFC0113000, No.2016YFB1001503), Natural Science Foundation of China (No.U1705262, No.61705262, No.61772443, No.61572410).

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.
[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.
[3] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.
[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015, pp. 1–9.
[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014, pp. 580–587.
[8] R. Girshick, "Fast r-cnn," in *ICCV*, 2015, pp. 1440–1448.
[9] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *NIPS*, 2015.
[10] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *TPAMI*, vol. 39, no. 4, pp. 640–651, 2017.
[11] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *NIPS*, 2016.
[12] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *ICLR*, 2017.
[13] J. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *ICCV*, 2017.
[14] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang, "Accelerating convolutional networks via global & dynamic filter pruning." in *IJCAI*, 2018.
[15] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *BMVC*, 2014.
[16] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," in *ICLR*, 2015.
[17] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *NIPS*, 2014, pp. 1269–1277.
[18] S. Lin, R. Ji, C. Chen, and F. Huang, "Espace: Accelerating convolutional neural networks via eliminating spatial & channel redundancy," in *AAAI*, 2017, pp. 1424–1430.
[19] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, 2016.
[20] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *NIPS*, 2015.
[21] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.
[22] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.
[23] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *ICML*, 2015.
[24] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *ICCV*, 2015, pp. 2857–2865.
[25] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *TPAMI*, vol. 38, no. 10, pp. 1943–1955, 2016.
[26] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *NIPS*, 2015, pp. 1135–1143.
[27] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *ICLR*, 2016.
[28] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *arXiv preprint arXiv:1507.06149*, 2015.
[29] M. Denil, B. Shakibi, L. Dinh, N. de Freitas *et al.*, "Predicting parameters in deep learning," in *NIPS*, 2013, pp. 2148–2156.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.
[31] S. Lin, R. Ji, X. Guo, and X. Li, "Towards convolutional neural networks compression via global error reconstruction," in *IJCAI*, 2016.
[32] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *ICLR*, 2015.
[33] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.
[34] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010.
[35] H. Valpola, "From neural pca to deep unsupervised learning," in *Advances in Independent Component Analysis and Learning Machines*. Elsevier, 2015, pp. 143–171.
[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.
[37] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *CVPR*, 2016, pp. 4820–4828.
[38] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *ICLR*, 2017.
[39] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017.
[40] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," in *ICLR*, 2017.
[41] Y. Ioannou, D. Robertson, R. Cipolla, A. Criminisi *et al.*, "Deep roots: Improving cnn efficiency with hierarchical filter groups," in *CVPR*, 2017.
[42] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *CVPR*, 2018.
[43] 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 preprint arXiv:1704.04861*, 2017.
[44] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017.
[45] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *ICLR*, 2016.
[46] Y. Ioannou, D. Robertson, J. Shotton, R. Cipolla, and A. Criminisi, "Training cnns with low-rank filters for efficient image classification," in *ICLR*, 2016.
[47] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang, "Deep fried convnets," in *ICCV*, 2015, pp. 1476–1483.
[48] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage." in *NIPS*, vol. 2, 1989, pp. 598–605.
[49] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *NIPS*, 1993.
[50] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *NIPS*, 2015, pp. 442–450.
[51] C. Bucilu, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *SIGKDD*, 2006, pp. 535–541.
[52] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *NIPS*, 2014, pp. 2654–2662.
[53] Y. Wang, C. Xu, S. You, D. Tao, and C. Xu, "Cnnpack: Packing convolutional neural networks in the frequency domain," in *NIPS*, 2016.
[54] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.
[55] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.
[56] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain separation networks," in *NIPS*, 2016, pp. 343–351.
[57] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017.
[58] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Artificial Intelligence and Statistics*, 2015, pp. 562–570.
[59] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM Multimedia*, 2014.
[60] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.
[61] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *TPAMI*, vol. 35, no. 8, pp. 1798–1828, 2013.
[62] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-UCSD Birds 200," California Institute of Technology, Tech. Rep. CNS-TR-2010-001, 2010.

Shaohui Lin received M.S. degree from Jimei University, Fujian, China, in 2014. He is currently pursuing the Ph.D. degree in Information and Computing Science at Xiamen University. His research interests include machine learning, and computer vision.



Rongrong Ji serves as the Professor of Xiamen University. He has been a Postdoc research fellow in the Department of Electrical Engineering, Columbia University from 2010 to 2013, worked with Professor Shih-Fu Chang. He obtained his Ph.D. degree in Computer Science from Harbin Institute of Technology, graduated with a Best Thesis Award at HIT. His research interests include image and video search, content understanding, mobile visual search, and social multimedia analytics. He is the recipient of the Best Paper Award at ACM Multimedia 2011 and Microsoft Fellowship 2007. He is a guest editor for IEEE Multimedia Magazine, Neurocomputing, and ACM Multimedia Systems Journal.

Chao Chen received M.S degree from Xiamen University, Fujian, China, in 2017. He mainly focus on deep learning and computer vision.

Dacheng Tao is Professor of Computer Science and ARC Laureate Fellow in the School of Information Technologies and the Faculty of Engineering and Information Technologies, and the Inaugural Director of the UBTECH Sydney Artificial Intelligence Centre, at the University of Sydney. He mainly applies statistics and mathematics to Artificial Intelligence and Data Science. His research results have expounded in one monograph and 200+ publications at prestigious journals and prominent conferences, such as IEEE T-PAMI, T-IP, T-NNLS, IJCV, JMLR, NIPS, ICML, CVPR, ICCV, ECCV, ICDM; and ACM SIGKDD, with several best paper awards, such as the best theory/algorithm paper runner up award in IEEE ICDM07, the best student paper award in IEEE ICDM13, the distinguished paper award in the 2018 IJCAI, the 2014 ICDM 10-year highest-impact paper award, and the 2017 IEEE Signal Processing Society Best Paper Award. He is a Fellow of the Australian Academy of Science, AAAS, IEEE, IAPR, OSA and SPIE.

Jiebo Luo received the PhD degree from the University of Rochester in 1995. He is a senior principal scientist at Kodak Research Laboratories, Rochester, New York. His research interests include image processing, computer vision, machine learning, data mining, social media. He is a pioneer for contextual inference in semantic understanding of visual data, and social multimedia data mining. He has published extensively in these fields with over 300 papers and 90 US patents. He has been involved in numerous technical conferences, including serving as the program co-chair of ACM Multimedia 2010, IEEE CVPR 2012 and IEEE ICIP 2017. He is a Fellow of the SPIE, IEEE, and IAPR.