Received: 19 February 2018 | Accepted: 3 May 2018

DOI: 10.4218/etrij.2018-0065

SPECIAL ISSUE

WILEY ETRI Journal

Deep compression of convolutional neural networks with low-rank approximation

Marcella Astrid[1] | **Seung-Ik Lee**[1,2]

[1]Department of Computer Software, University of Science and Technology, Daejeon, Rep. of Korea.

[2]SW·Contents Research Laboratory, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea.

Correspondence
Seung-Ik Lee, SW·Contents Research Laboratory, Electronics and Telecommunications Research Institute and Department of Computer Software, University of Science and Technology, Daejeon, Rep. of Korea.
Email: the_silee@etri.re.kr

Funding information
Institute for Information and communications Technology Promotion (IITP), Grant/Award Number: 2017-0-00067; Institute for Information and Communications Technology Promotion (IITP) of the Korean government (MSIT), Grant/Award Number: 2017-0-00067.

The application of deep neural networks (DNNs) to connect the world with cyber physical systems (CPSs) has attracted much attention. However, DNNs require a large amount of memory and computational cost, which hinders their use in the relatively low-end smart devices that are widely used in CPSs. In this paper, we aim to determine whether DNNs can be efficiently deployed and operated in low-end smart devices. To do this, we develop a method to reduce the memory requirement of DNNs and increase the inference speed, while maintaining the performance (for example, accuracy) close to the original level. The parameters of DNNs are decomposed using a hybrid of canonical polyadic–singular value decomposition, approximated using a tensor power method, and fine-tuned by performing iterative one-shot hybrid fine-tuning to recover from a decreased accuracy. In this study, we evaluate our method on frequently used networks. We also present results from extensive experiments on the effects of several fine-tuning methods, the importance of iterative fine-tuning, and decomposition techniques. We demonstrate the effectiveness of the proposed method by deploying compressed networks in smartphones.

KEYWORDS
convolutional neural network, CP-decomposition, cyber physical system, model compression, singular value decomposition, tensor power method

1 | INTRODUCTION

A cyber physical system (CPS) is an important technology that helps to connect computational resources with the physical world, in fields such as aerospace [1], healthcare [2], and manufacturing [3], where an essential requirement is to understand and interact with the world. However, because of limited resources, particularly in low-end devices, the design of perception software for CPS devices remains a major challenge.

Deep neural networks (DNNs) and convolutional neural networks (CNNs) have successfully exhibited state-of-the-art performance in various applications utilized to understand the world. For example, VGG [4] and GoogleNet [5] achieved a top-five accuracy of more than 90%, and AlexNet [6] achieved a top-five accuracy of 80% in ImageNet 1000 class classification [7]. In object detection using the PASCAL VOC 2007 dataset, YOLOv2 [8] achieved a rate of 78.6%, and SSD500 [9] achieved a rate of 76.9%, which are much higher than those obtained for non-deep learning methods.

However, CNNs typically rely on a large number of parameters and significant computational power. For example, AlexNet requires 240 MB of memory for the parameters, and more than 700 million float point operations for just one inference. As such, it is not easy to implement

CNNs in resource-limited CPS devices, and they often cannot be loaded into memory. Furthermore, as is typical with small CPS devices, inference in a CNN becomes very slow, and thus CPS systems cannot respond to or perceive the world in a timely manner, which is demonstrated using an autonomous driving car [10] that limits the speed to 72 km/h in order to reliably perceive objects 30 m away.

In this article, we aim to answer the following question: Can we make CNNs smaller and faster than original networks for efficient operation in low-end CPS devices? To represent CNNs more efficiently, we use tensor decomposition, under the assumption that the original tensor should have some degree of redundancy. Then, one question arises: Do CNNs have redundancy in their tensors? As empirically proven by Denil and others [11], a total of 95% of the CNN weights can be predicted (that is, they are redundant) from only the remaining 5% without a loss of accuracy.

However, using a tensor approximation for CNNs raises a number of unique challenges. First, CNNs are typically composed of many layers with their own set of parameters or weights, which are called tensors in the deep-learning field. As we approximate the tensors of a CNN up through the layers, approximation errors will accumulate, and this error accumulation makes the CNN behave quite differently compared to the original, uncompressed networks; this eventually results in a significant drop in performance. This type of instability was reported in [12] and [13].

Second, the aim is for the decomposition to be as compact as possible while maintaining the performance close to that of the original. Finally, a further practical consideration is that the decomposition technique should be readily implemented using widespread deep-learning toolkits, such as PyTorch [14], Tensorflow [15], and Caffe [16]; otherwise, it should be implemented from scratch.

We address these challenges by developing a combination of canonical polyadic–singular value decomposition (CP–SVD) and the tensor power method (TPM), as well as a hybrid fine-tuning method. CP–SVD decomposes convolutional layers using CP-decomposition as a linear combination of outer products of vectors, and decomposes the fully connected layers using SVD. Then, using the TPM, we estimated the value of each vector in the decomposition. A hybrid fine-tuning method is proposed to train the decomposed convolutional layers using iterative fine-tuning, whereas one-shot fine-tuning is used for fully connected layers.

We evaluated our proposed method by decomposing pre-trained weights of several representative CNN networks, that is AlexNet [6], GoogleNet [5], and VGG16 [4], and by deploying a decomposed network to Android smartphones to show its effectiveness in actual smart devices. Our results demonstrate improvements in the compression rate and speed from CP–SVD decomposition and TPM, as well as iterative fine-tuning, when compared with other tensor decomposition approaches [12,13]. We also present several issues, such as the importance of TPM and iterative fine-tuning or freezing strategies, which are naturally treated as decision points when dealing with decomposition.

2 | METHODOLOGY

CNNs typically have a sequence of convolutional layers followed by fully connected layers at the end, as shown in Figure 1. Given an input, a set of convolution filters (or weights) was applied to produce feature maps. Each filter strides across the spatial dimension, creating a linear combination of its weights and the input, and producing one feature map (or channel) of the next layer. In this article, the focus considered for decompression is the weights shown in Figure 1. The weights in convolutional layers are represented as tensors, and the weights in FC layers are represented as matrices. We treat the convolutional layers and FC layers differently by applying CP-decomposition for convolutional layers and SVD [17] for FC layer weights.

2.1 | CP-based tensor decomposition for convolutional layers

CP-decomposition decomposes a tensor as a linear combination of the outer products of vectors. Equation (1) shows a three-way tensor $\mathcal{X}$ decomposed by R number of components, as visualized in Figure 2. The rank R is a key factor of the compression rate, where a low R will lead to a higher compression rate.

$$\mathcal{X} \approx \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r. \tag{1}$$

A convolution operation with a convolution filter tensor $\mathcal{K}$ of size $T \times S \times (D \times D)$ maps a three-way input tensor $\mathcal{X}$ of size $S \times W \times H$ to a three-way output tensor $\mathcal{Y}$ of size $T \times$

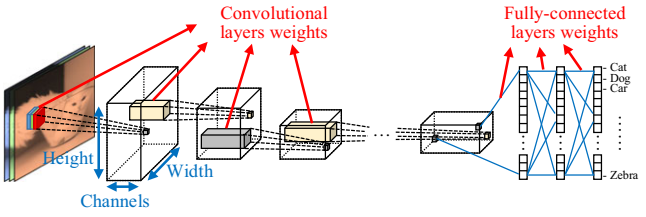

FIGURE 1 Typical CNN architecture

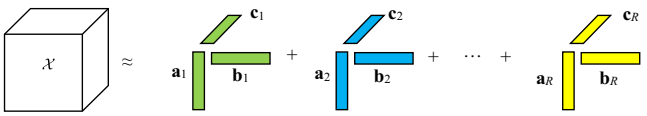

FIGURE 2 CP-decomposition of a three-way tensor

$W' \times H'$, where S is the filter depth corresponding to the number of input channels, D is the filter spatial size, H and W represent input spatial dimensions, T is the number of filters, and H' and W' are output spatial dimensions (2). Note that for simplicity, we assume square filters and an odd D.

$$\mathcal{Y}_{t,w',h'} = \sum_{s=1}^{S}\sum_{j=1}^{D}\sum_{i=1}^{D}\mathcal{K}_{t,s,j,i}\mathcal{X}_{s,w_j,h_i},$$
$$w_j = (w'-1)\Delta + j - p, \quad \text{and} \quad h_i = (h'-1)\Delta + i - p \tag{2}$$

where Δ is the stride and p is zero-padding.

With this setting, we want to approximate the filter tensor $\mathcal{K}$ through CP-decomposition assuming that rank R is given in (3). Note that we do not decompose the spatial dimensions because they are usually small enough, for example, 3×3 and 5×5.

$$\mathcal{K}_{t,s,j,i} = \sum_{r=1}^{R}\mathcal{K}^{S}_{r,s}\mathcal{K}^{DD}_{r,j,i}\mathcal{K}^{T}_{t,r} \tag{3}$$

where $\mathcal{K}^S$, $\mathcal{K}^{DD}$, and $\mathcal{K}^T$ are the three components of a decomposition representing tensors of sizes $R \times S$, $R \times (D \times D)$, and $T \times R$, respectively.

Substituting (3) into (2), we obtain (4) and (5), which indicate how the original convolution operation can be approximated through three consecutive convolutions. First, input $\mathcal{X}$ is convolved with filter $\mathcal{K}^S$ (6), giving an intermediate result $\mathcal{Z}$. This $\mathcal{Z}$ is convolved with filter $\mathcal{K}^{DD}$ (7), producing another intermediate result, $\mathcal{Z}'$. Finally, $\mathcal{Z}'$ is convolved with filter $\mathcal{K}^T$ (8), and we obtain the intended output $\mathcal{Y}$. As a result, the original convolution is converted into three small convolution operations, as illustrated in Figure 3B.

$$\mathcal{Y}_{t,w',h'} = \sum_{s=1}^{S}\sum_{j=1}^{D}\sum_{i=1}^{D}\sum_{r=1}^{R}\mathcal{K}^{S}_{r,s}\mathcal{K}^{DD}_{r,j,i}\mathcal{K}^{T}_{t,r}\mathcal{X}_{s,w_j,h_i}, \tag{4}$$

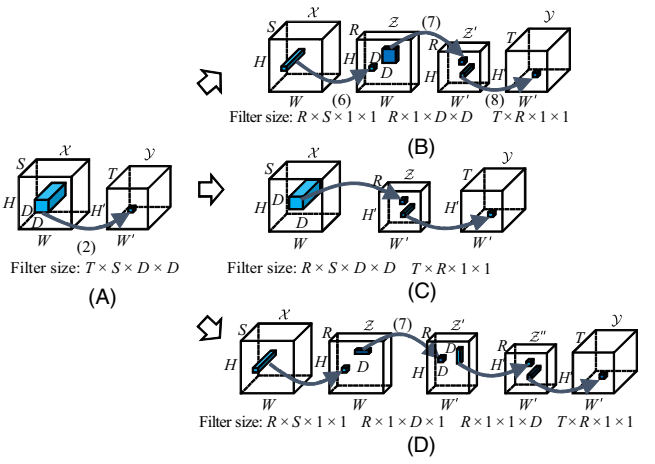

FIGURE 3 Convolutional layer compression as two-, three-, and four-way tensors: (A) original convolution, (B) 3-way, (c) 2-way, and (D) 4-way

$$\mathcal{Y}_{t,w',h'} = \sum_{r=1}^{R}\mathcal{K}^{T}_{t,r}\left(\sum_{j=1}^{D}\sum_{i=1}^{D}K^{DD}_{r,j,i}\left(\sum_{s=1}^{S}\mathcal{K}^{S}_{r,s}\mathcal{X}_{s,w_j,h_i}\right)\right), \tag{5}$$

$$\mathcal{Z}_{r,w,h} = \sum_{s=1}^{S}\mathcal{K}^{S}_{r,s}\mathcal{X}_{s,w,h} \tag{6}$$

where $\mathcal{K}^S$ is a tensor of size $R \times S \times 1 \times 1$, and $\mathcal{Z}$ of size $R \times W \times H$.

$$\mathcal{Z}'_{r,w',h'} = \sum_{j=1}^{D}\sum_{i=1}^{D}\mathcal{K}^{DD}_{r,j,i}\mathcal{Z}_{r,w_j,h_i} \tag{7}$$

where $\mathcal{K}^{DD}$ is a tensor of size $R \times 1 \times D \times D$, and $\mathcal{Z}'$ of size $R \times W' \times H'$.

$$\mathcal{Y}_{t,w',h'} = \sum_{r=1}^{R}\mathcal{K}^{T}_{t,r}\mathcal{Z}'_{r,w',h'} \tag{8}$$

where $\mathcal{K}^T$ is a tensor of size $T \times R \times 1 \times 1$.

In addition, as the convolution operation for a CNN input usually has a small filter depth S (it is typically $S = 3$ for RGB image channels), we combine the filter depth and spatial dimensions of the first convolutional layer into a two-way tensor $\mathcal{K}^{SDD}$ of size $T \times (S \times D \times D)$, as shown in Figure 3C, resulting in (9).

$$\mathcal{K}_{t,s,j,i} = \sum_{r=1}^{R}\mathcal{K}^{SDD}_{r,s,j,i}\mathcal{K}^{T}_{t,r}. \tag{9}$$

Substituting (9) into (2), we obtain (10), which is again separated into two sequential operations, as follows:

$$\mathcal{Y}_{t,w',h'} = \sum_{r=1}^{R}\mathcal{K}^{T}_{t,r}\left(\sum_{j=1}^{D}\sum_{i=1}^{D}\sum_{s=1}^{S}\mathcal{K}^{SDD}_{r,s,j,i}\mathcal{X}_{s,w_j,h_i}\right), \tag{10}$$

$$\mathcal{Z}_{r,w',h'} = \sum_{j=1}^{D}\sum_{i=1}^{D}\sum_{s=1}^{S}\mathcal{K}^{SDD}_{r,s,j,i}\mathcal{X}_{s,w_j,h_i} \tag{11}$$

where $\mathcal{Z}$ has size $R \times W' \times H'$, and $\mathcal{K}^{SDD}$ has size $R \times S \times D \times D$.

$$\mathcal{Y}_{t,w',h'} = \sum_{r=1}^{R}\mathcal{K}^{T}_{t,r}\mathcal{Z}_{r,w',h'} \tag{12}$$

where $\mathcal{K}^T$ has size $T \times R \times 1 \times 1$.

Note that the method used to decompose the tensors allows the decomposed results to be seamlessly supported by the most popular deep-learning toolkits such as PyTorch [14], Tensorflow [15], and Caffe [16], because the results are only composed of a series of convolution operations and densely decomposed tensors that are very well supported in the toolkits. However, if the results consisted of other kinds of non-convolutional operations or sparse tensors, it would not be well supported by the toolkits.

2.2 | Fully connected layer decomposition

As is typical with CNNs, the convolutional layers are followed by fully connected layers. An FC layer is described in 13), where $\mathbf{x}$ is an input vector of size M, $\mathbf{y}$ is an output vector of size N, and $\mathbf{W}$ is a weight matrix of size $M \times N$.

$$\mathbf{y}^{\mathrm{T}} = \mathbf{x}^{\mathrm{T}}\mathbf{W}. \tag{13}$$

Because the weights can be represented using a matrix, we propose the use of an SVD technique for fully connected layers instead of CP-decomposition. With SVD, a matrix $\mathbf{W}$ can be decomposed into three small matrices $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}^{\mathrm{T}}$ with sizes $M \times R$, $R \times R$, and $R \times N$, respectively, where $\mathbf{U}$ is a left singular matrix, $\mathbf{D}$ is a diagonal singular value matrix, and $\mathbf{V}$ is a right singular matrix.

$$\mathbf{W} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}} = (\mathbf{U}\mathbf{D})\mathbf{V}^{\mathrm{T}}. \tag{14}$$

Combining (14) into (13) gives us (15), which can then be separated into (16) and (17), meaning that one FC layer can be decomposed into two layers having sizes $M \times R$ and $R \times N$, with $\mathbf{z}$ as an intermediate layer of size R.

$$\mathbf{y}^{\mathrm{T}} = (\mathbf{x}^{\mathrm{T}}(\mathbf{U}\mathbf{D}))\mathbf{V}^{\mathrm{T}}, \tag{15}$$

$$\mathbf{z}^{\mathrm{T}} = \mathbf{x}^{\mathrm{T}}(\mathbf{U}\mathbf{D}), \tag{16}$$

$$\mathbf{y}^{\mathrm{T}} = \mathbf{z}^{\mathrm{T}}\mathbf{V}^{\mathrm{T}}. \tag{17}$$

2.3 | Theoretical complexity analysis

The original convolution operation (2) and FC layer (13) require TSD^2 and MN memory with computational costs of $O(T \times S \times D^2 \times W' \times H')$ and $O(MN)$, respectively. After decomposition, the memory compression ratio E and speed-up ratio C of the convolutional layer are given by

$$E = \frac{TSD^2}{RS + RD^2 + TR}, \tag{18}$$

$$C = \frac{TSD^2W'H'}{RSWH + RD^2W'H' + TRW'H'}. \tag{19}$$

For an FC layer, they are

$$E = C = \frac{MN}{MR + RN}. \tag{20}$$

2.4 | Computation of tensor decomposition

For tensor approximation, we require an optimization to minimize the difference between the approximated tensor (decomposed) and the target tensor (original). We used the TPM, which can generally approximate tensors better with a lower rank [18].

As explained in Algorithm 1, given a rank R, the TPM approximates a tensor $\mathcal{W}$ by iteratively adding components (one outer product of the vectors). Each component $\mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r$ approximates the residual of previous decomposition components $\widehat{\mathcal{W}}$ (Algorithm 1(2c)) by minimizing the error $\|\widehat{\mathcal{W}} \times \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r\|_2$ using a coordinate-descent algorithm (Algorithm 1 (2a loop)). During each iteration, the TPM updates one vector while keeping the other vectors unchanged, which is a variant of the coordinate descent that updates a vector (>1 element) instead of a coordinate (1 element).

2.5 | Rank selection

Thus far, we have assumed that the rank R for decomposition is given. In fact, the rank significantly affects the overall compression rate and performance. However, an algorithm that can find the optimal tensor rank has yet to be developed; rather, it is NP-hard [19]. In this subsection, we propose the use of a sensitivity-based rank-selection method.

First, we define the sensitivity of a layer as its responsiveness to decomposition. We measure the sensitivity based on the decrease in accuracy caused by the layer decomposition. If a layer has high sensitivity, it will not be desirable to set the rank too low, and vice versa.

To be more specific, we measured the accuracy drop of a layer by decomposing it while maintaining the other layers and fine-tuning the whole network for only one epoch in order to obtain the accuracy loss caused from the layer's decomposition. We repeated this process for each layer in a CNN with the same rank, allowing them to be compared with each other. Finally, the rank for each layer is determined proportionally to the accuracy loss ratio. However, because we only know the ratio, the average rank for all layers still needs to be arbitrarily chosen.

Algorithm 1. Tensor Power Method [18].

1) Initialize $\widehat{\mathcal{W}} = \mathcal{W}$.
2) For $k = 1, \ldots, R$
   a) Repeat for N iterations:
      i) $\mathbf{a}_k \leftarrow \widehat{\mathcal{W}} \times_2 \mathbf{b}_k \times_3 \mathbf{c}_k / \|\widehat{\mathcal{W}} \times_2 \mathbf{b}_k \times_3 \mathbf{c}_k\|_2$
      ii) $\mathbf{b}_k \leftarrow \widehat{\mathcal{W}} \times_1 \mathbf{a}_k \times_3 \mathbf{c}_k / \|\widehat{\mathcal{W}} \times_1 \mathbf{a}_k \times_3 \mathbf{c}_k\|_2$
      iii) $\mathbf{c}_k \leftarrow \widehat{\mathcal{W}} \times_1 \mathbf{a}_k \times_2 \mathbf{b}_k / \|\widehat{\mathcal{W}} \times_1 \mathbf{a}_k \times_2 \mathbf{b}_k\|_2$
   b) $d_k \leftarrow \widehat{\mathcal{W}} \times_1 \mathbf{a}_k \times_2 \mathbf{b}_k \times_3 \mathbf{c}_k$
   c) $\widehat{\mathcal{W}} \leftarrow \widehat{\mathcal{W}} - d_k \mathbf{a}_k \otimes \mathbf{b}_k \otimes \mathbf{c}_k$

2.6 | Hybrid fine-tuning

Decomposing multiple convolution layers using CP-decomposition will usually end in an unrecoverable drop in accuracy, which appears to be caused by CP instability, as

reported in [12] and [13]. We adopt iterative fine-tuning [20] to overcome the instability problem, but only to convolutional layers. With iterative fine-tuning, we fine-tune the whole network whenever each layer is decomposed. The idea behind the iteration is as follows: Decomposing a layer will inevitably result in an error or drop in accuracy arising from approximating the original tensor. When we decompose multiple layers, too many errors will be accumulated to be recovered by the so-called one-shot fine-tuning at the end of the process, which is a typical approach, as described in [12] and [21]. However, with iterative fine-tuning, we expect that the total number of errors will be kept from becoming too large to be recovered. The procedure is illustrated in Figure 4.

Note that the iterative fine-tuning process applies only to convolution layer decomposition. For FC layers, we propose the use of one-shot fine-tuning to speed up the training time and prevent a loss of accuracy. In one-shot fine-tuning, all FC layers are decomposed first, and then fine-tuned once.

2.7 | Method summary

A summary of the method is shown in Algorithm 2.

Algorithm 2. Overall method.

1) Read network *net* = *pretrained_weight*.
2) Rank selection
   a) *r_sensitivity* = a small rank (we use 10 for AlexNet and 20 for GoogleNet and VGG16)
   b) *r_ave* = arbitrary rank for average
      i) *net temp*$_l$ = decompose *layer*$_l$ with rank *r_sensitivity*
      ii) *net temp*$_l$ = 1 epoch fine-tuning of net *temp*$_l$
      iii) *sensitivity*$_l$ = top-5 loss of net *temp*$_l$
   c) For layer $l = 1 \ldots L$:
   d) Decide arbitrary average rank *r_ave*
   e) For layer $l = 1 \ldots L$:
      i) $rank_l = \frac{sensitivity_l}{\sum_{m=1}^{L} sensitivity_m} r\_ave * L$
3) All-layer decomposition
   a) For layer $l = 1 \ldots L$:
      i) If *layer*$_l$ is convolutional:
         A) *net* = decompose *layer*$_l$ with *rank*$_l$ from *net*
         B) *net* = fine-tune *net*
      ii) Else:
         A) While *layer*$_l$ is fully connected,
            I) *net* = decompose *layer*$_l$ with *rank*$_l$
            II) Go to next layer $l = l + 1$
         B) *net* = fine-tuning *net*
         C) Break if $l > L$

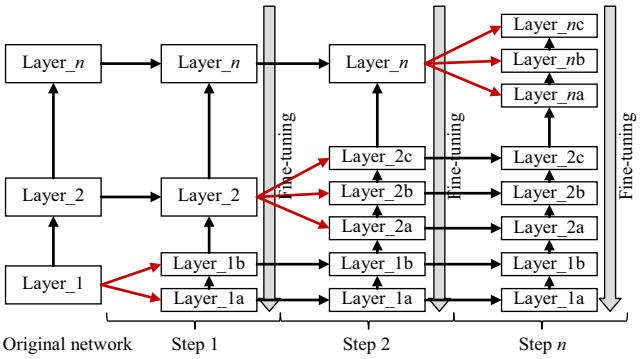

FIGURE 4 Iterative decomposition and fine-tuning

3 | EXPERIMENTS

Throughout the experiments, we first aim to answer the motivational question:

Q1: can CNNs be efficiently decomposed and implemented seamlessly in deep learning toolkits? We answered this question by applying the proposed method to representative CNNs such as AlexNet, GoogleNet, and VGG16, and then by deploying the decomposed network to an Android smartphone. In addition to the motivating question, we also try to answer several additional questions that arose during the development of the method:

Q2: Do the successful results (not yet shown) come from either the TPM or iterative fine-tuning, or from both? To answer this question, we conducted an ablation study instantiating each component separately (Section 3.4).

Q3: What is the best strategy for layer freezing when we apply iterative fine-tuning? In other words, do we need to freeze the already fine-tuned layers or fine-tune them altogether without freezing (Section 3.5)?

Q4: Do we need to decompose spatial dimensions and input channels despite the small dimensions and number of channels (Section 3.6)?

Q5: Instead of sensitivity-based rank selection, what would happen if we simply assign ranks proportional to the number of parameters (Section 3.7)?

Throughout the experiments, we used the Caffe [16] framework to read the pre-trained weights and for fine-tuning, and R software for decomposition. An accuracy evaluation and fine-tuning was conducted using the ImageNet 2012 dataset [7], which classifies 1,000 classes and has 1.2 million sets of training data and 50 thousand sets of validation data. Because there is no label in the testing data, the accuracy is measured using the validation data. A Linux machine with a 3.50 GHz Intel(R) Core(TM) i7-5930K CPU and 64 GB of RAM was used for the experiments.

3.1 | Models

We show the method's generalization by applying it to three of the most popular CNN architectures: AlexNet, GoogleNet, and VGG16. AlexNet [6] consists of five convolutional layers and three FC layers, which cover around 61 million weights and 724 million floating-point computations for each image inference. We applied three-way decomposition to the convolutional layers, except for the conv1 layer, which uses two-way decomposition.

With GoogleNet [5], we applied our method to 3×3 convolution layers inside the inception modules, and to convolutional layers outside the inception modules (conv1 and conv2). Of the several filters in the inception module, we only decomposed the 3×3 filters because they are the most computationally-intensive filters, accounting for more than 50% of the calculations. In addition, we also decomposed filters outside the inception modules: conv1 and conv2. Consequently, 3 million weights are decomposed out of almost 7 million weights in total, with a computational cost of 1 billion from 1.6 billion.

VGG16 [4] consists of 13 convolutional layers and three FC layers. Each convolutional layer has 3×3 filters with a stride of 1, and with additional padding in each input spatial dimension. Further, VGG16 has 138 million weights and 15 trillion computations for one image inference. VGG16 is fine-tuned iteratively for every four layers up to layer 8, that is, all four layers are fine-tuned together to speed up the training, and layers 9 through 12 are fine-tuned every two layers for accuracy. Refer to the Appendices for further details.

3.2 | All-layer decomposition

This section shows the results of the all-layer decomposition of AlexNet, GoogleNet, and VGG16, as summarized in Table 1, with the ranks selected based on sensitivity, as shown in Table 2, and training hyperparameters shown in Table 3. With AlexNet, our approach shows a $\times$ 7.06 parameter and $\times$5.13 theoretical cost reduction, with a $\times$ 3.48 speed up for one inference. Our method outperforms Tucker in terms of accuracy, cost, and CPU time. Note that the theoretical computational reduction ($\times$5.13) and CPU time ($\times$3.48) do not match because the CPU time contains all boilerplate processing steps for inference, for example, loading the input, transformation, pooling, and activation functions, whereas the theoretical cost only considers the convolution operation.

In the case of GoogleNet, the results are not very distinct. This is because GoogleNet has much fewer parameters such that although they are still improved, there is not much room for decomposition. In VGG16, our overall approach again outperforms Tucker [12], except that in this case, we have a relatively high accuracy drop. This is because the iterative fine-tuning was applied once every four or two layers to fasten the training time. However, it still demonstrates that our hybrid fine-tuning is successful in restoring the accuracy in very deep networks with much fewer weights.

3.3 | Fully connected layer decomposition

In contrast to a previous work [20] that applied iterative fine-tuning even to the FC layers, we demonstrate that one-shot fine-tuning shows better results with respect to FC layers, as illustrated in Figure 5.

3.4 | Importance of TPM and iterative fine-tuning

To answer question Q2, we performed an ablation study by instantiating them separately during AlexNet compression. More specifically, we decomposed AlexNet with either

TABLE 1 All-layer decomposition results

Network	Method	Top-1 acc.	Top-5 acc.	Weights	Theory cost	CPU time (ms)
AlexNet	Original	56.83	79.95	61.0M	724M	167.71
	Tucker [12]	N/A	78.33 (–1.70)	11.2M ($\times$5.46)	272M ($\times$2.67)	80.60 ($\times$2.08)
	CP-sensitivity (Ave rank: conv 150, FC 300)	**55.08** (–1.75)	**79.05** (–0.90)	8.7M ($\times$6.98)	205M ($\times$3.53)	53.79 ($\times$3.12)
	CP-sensitivity (Ave rank: conv 100, FC 300)	54.43 (–2.4)	78.15 (–1.80)	**8.6M** ($\times$7.06)	**141M** ($\times$5.13)	**48.21** ($\times$3.48)
GoogleNet	Original	68.93	89.14	7.0M	1,583M	211.73
	Tucker [12]	N/A	**88.66 (–0.24)**	4.7M ($\times$1.49)	760M ($\times$2.08)	183.17 ($\times$1.16)
	CP-sensitivity (Ave rank: conv 35)	**67.30** (–1.63)	88.08 (–1.06)	**4.0M** ($\times$1.74)	**659M** ($\times$2.40)	**155.62** ($\times$1.36)
VGG16	Original	68.36	88.44	138M	15,470M	1,568.33
	Tucker [12]	N/A	**89.40** (–0.50)	127M ($\times$1.09)	3,139M ($\times$4.93)	1,439.07 ($\times$1.09)
	CP-sensitivity (Ave rank: conv 80, FC 150)	**63.38** (–4.98)	85.45 (–2.99)	**7.4M** ($\times$18.80)	**1,600M** ($\times$9.67)	**576.74** ($\times$2.72)

TABLE 2 Sensitivity-based ranks

	Convolutional layers							
Alexnet	**Layer**		**Conv1**	**Conv2**	**Conv3**	**Conv4**	**Conv5**	**Total**
	Top-5 loss		5.38	11.89	11.86	13.68	15.17	57.98
	Rank	Ave = 150	69	154	153	178	196	750
		Ave = 100	47	102	103	118	130	500
	FC layers							
	Layer		**FC6**	**FC7**	**FC8**	**Total**		
	Top-5 loss		28.59	21.50	20.31	70.40		
	Rank	Ave = 300	365	275	260	900		
GoogleNet	**Layer**		**Conv1**	**Conv2**	**i3a**	**i3b**	**i4a**	**i4b**
	Top-5 loss		7.12	8.35	7.18	7.77	6.78	6.97
	Rank	Ave = 35	35	41	35	38	33	34
	Layer		**i4c**	**i4d**	**i4e**	**i5a**	**i5b**	**Total**
	Top-5 loss		7.08	6.63	6.59	7.41	7.10	78.98
	Rank	Ave = 35	34	32	32	36	35	385
	Convolutional layers							
VGG16	**Layer**		**Conv1_1**	**Conv1_2**	**Conv2_1**	**Conv2_2**	**Conv3_1**	**Conv3_2**	**Conv3_3**
	Top-5 loss		4.71	4.04	4.12	6.00	9.25	5.27	5.11
	Rank	Ave = 80	59	50	51	75	115	65	64
	Layer		**Conv4_1**	**Conv4_2**	**Conv4_3**	**Conv5_1**	**Conv5_2**	**Conv5_3**	**Total**
	Top-5 loss		6.25	6.39	7.37	7.98	8.82	8.22	83.53
	Rank	Ave = 80	78	80	92	99	110	102	1,040
	FC layers							
	Layer		**FC6**	**FC7**	**FC8**	**Total**		
	Top-5 loss		12.46	11.11	10.93	34.50		
	Rank	Ave = 150	162	145	143	450		

TABLE 3 Training hyperparameters

Network	**Layer**	**Init LR**	**Batch size**	**Weight decay**	**LR decay**	**#Epoch (each iteration)**	**Momentum**
AlexNet	Conv1&2	0.001	128	0.0005	LR*0.1 every 5 epochs	15	0.9
	Others	0.002	32				
GoogleNet	All	0.001	64				
VGG16	All		32				

TPM and one-shot fine-tuning (disabling iterative fine-tuning) or random and iterative fine-tuning (disabling TPM), under an average rank of 150.

3.4.1 | TPM and one-shot fine-tuning

To determine the importance of iterative fine-tuning, we performed one-shot fine-tuning instead of iterative fine-tuning. The hyperparameters for one-shot fine-tuning are the same as AlexNet-others (Table 3). Figure 6 shows that the TPM combined with one-shot fine-tuning cannot recover the lost accuracy, which reaches almost a random guess. In contrast, the TPM combined with iterative fine-tuning (in both the convolution and FC layers) is able to recover the lost accuracy.

3.4.2 | Random and iterative fine-tuning

The effectiveness of iterative fine-tuning raises another question regarding the actual effect of the TPM on the results. This question naturally arises when the accuracy of

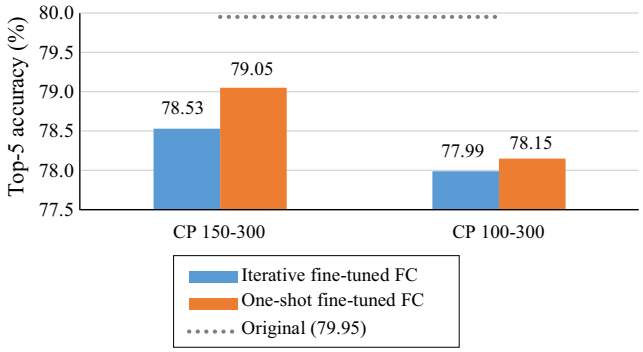

FIGURE 5 Comparison of iterative and one-shot FC training results

a decomposed layer drops close to a random guess (0.5% of the top-5 accuracy). To determine the importance of the TPM, we used random values for CP-decomposition instead of TPM values, with all other settings remaining the same.

As shown in Figure 7, until the conv4 layer, the random CP decomposition appears to rise above the random guess level (0.5% top-five accuracy), although the accuracy is obviously lower than the TPM. However, as the decomposition progresses to conv5 and beyond, a random decomposition fails to recover the accuracy. This means that although iterative fine-tuning helps to recover the accuracy, the optimization technique in CP also plays an important role in such a recovery.

3.5 | Freezing

One understated issue that accompanies fine-tuning is whether to freeze the layers, that is, whether to keep some layer weights fixed during fine-tuning. Without freezing,

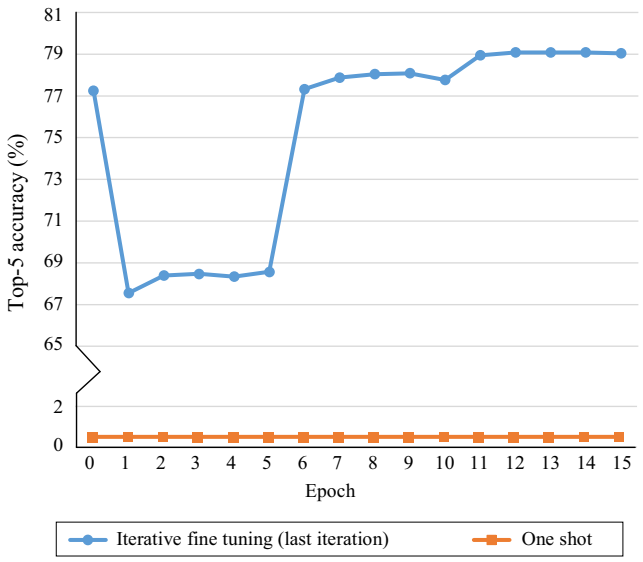

FIGURE 6 Accuracy of last iteration of iterative fine-tuning and one-shot fine-tuning

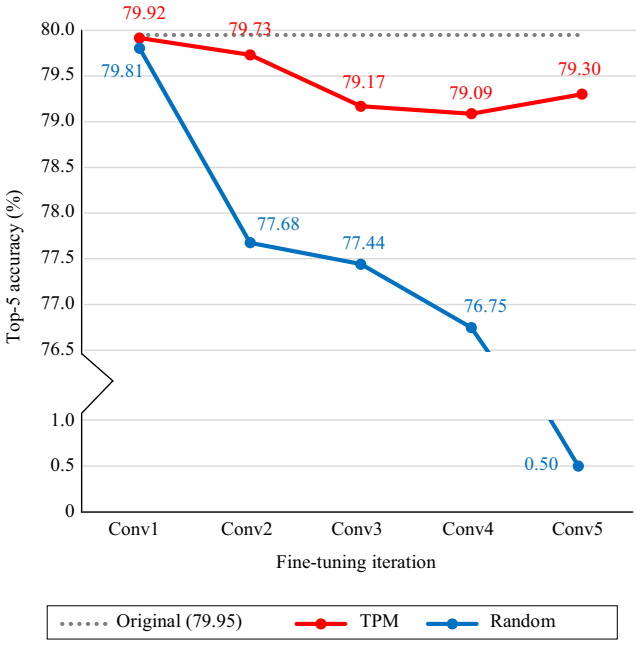

FIGURE 7 Accuracy of TPM and random initialization with iterative fine-tuning

all layers are adjusted together with the newly decomposed layer. In this work, we applied three different freezing schemes: (a) non-freezing, (b) semi-freezing, which freezes only the already-decomposed and fine-tuned layers, and (c) freezing all other layers except the currently decomposed layer. These freezing schemes are applied to decompose AlexNet. As shown in Figure 8, the non-freezing scheme provides the best accuracy compared to the others.

3.6 | Conv1 spatial and input channel compression

In this section, we address question Q4 to determine the need to decompose small dimensions, such as spatial dimensions and an input channel. We empirically show the effects of the dimensions on the accuracy and compression rate in conv1 ($D = 11$ and input channels $S = 3$) of AlexNet with three kinds of rank, $R = 25$, $R = 50$, and $R = 100$. Here, we tested three cases: (1) two-way decomposition without an input channel or spatial decomposition, (2) three-way decomposition without spatial decomposition, and (3) four-way decomposition with both an input channel and spatial decomposition, as shown in Figure 3.

As shown in Figure 9, all cases show almost the same level of accuracy regardless of the manner of decomposition, although three-way decomposition is observed to be the best. With respect to the computational cost and memory, as shown in Figure 10, two-way decomposition has the highest cost. However, two-way decomposition shows the best actual CPU time, with three-way decomposition

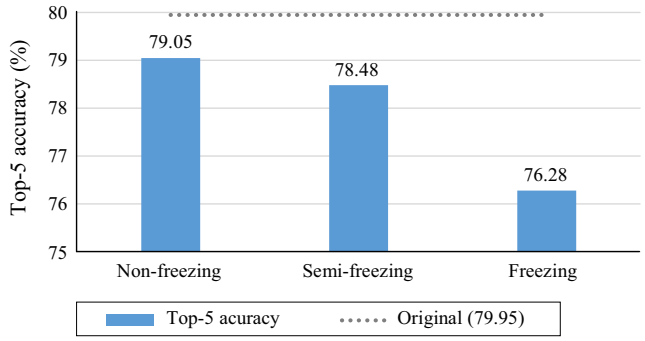

FIGURE 8 Top-5 accuracy with various freezing schemes

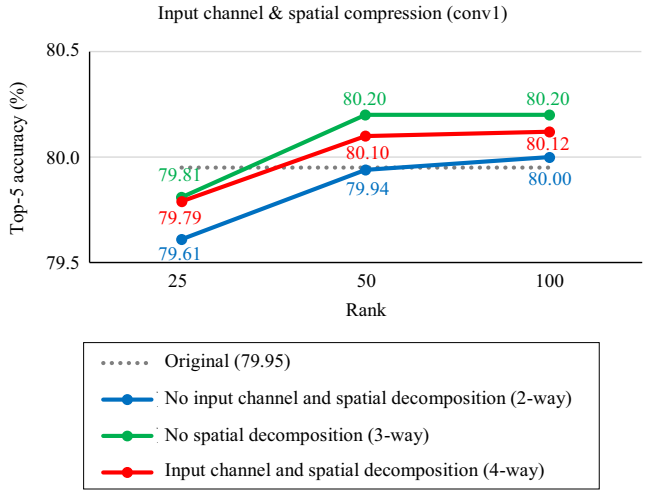

FIGURE 9 Comparison of accuracy of two-, three-, and four-way decomposition in conv1 layer

having an exceptionally high CPU time. One possible reason for this is the way that the deep-learning toolkit (in our case, Caffe) implements the convolution by first transforming the n-way tensor (namely, the input feature map) into a matrix. The transformation incurs a rearrangement of the tensor, and this overhead can cancel out the speed-up realized from the decomposition, particularly when the input channel size is much smaller than the rank, just as the case of conv1 shown in Figure 9 (also detailed in the Appendices).

3.7 | Rank-selection comparison

For question Q5, this section empirically demonstrates that the sensitivity-based rank-selection scheme outperforms the baseline for which we use the parameter-based rank-selection scheme. The parameter-based rank-selection method uses the parameter (weights) ratio, because intuitively, tensors with fewer parameters are expected to be decomposed with fewer parameters.

To compare the two schemes, we decomposed all convolutional layers of AlexNet with three-way decomposition. Figure 11 shows that sensitivity-based rank selection is

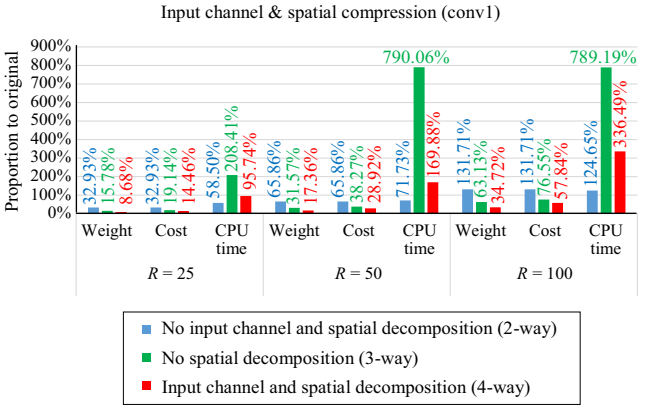

FIGURE 10 Weights, computational cost, and CPU time of two-, three-, and four-way decomposition

superior to parameter-based rank selection in terms of both compression and computation.

3.8 | Installation on android smartphones

The motivation for compression is to enable CNNs to operate on different devices, such as mobile phones. We installed the networks shown in Table 1 on a Nexus 5 phone with Android version 6.0.1 and 2 GB of RAM. We used Caffe-Android-Lib [22] for CNN instantiation in smartphones.

Table 4 shows the cost analysis results of an Android application. The load time indicates for how long the weight file is loaded into the application. The forward time indicates how long each feed forward is run, in milliseconds. The startup memory is the memory consumption before a feed forward, while the running memory is the memory consumption during a feed forward. Both are measured in MB. Overall, the decomposed networks outperform their respective original networks in terms of both weight and computational time. It should be noted that even the original VGG16 cannot be loaded into the device in contrast to the decomposed version.

4 | RELATED WORKS

Similar to our method, several low-rank-based tensor decomposition approaches have been proposed [12,13,20,21]. As with our method, Lebedev and others [13] uses CP-decomposition and a non-linear optimization technique, which is called nonlinear least squares (NLS). Our approach is different in that we used a greedy optimization, the TPM. However, they were not successful in compressing all layers in a deep network. Lebedev and others [13] does not combine any dimensions, and therefore a 4D layer is decomposed into four layers. Our

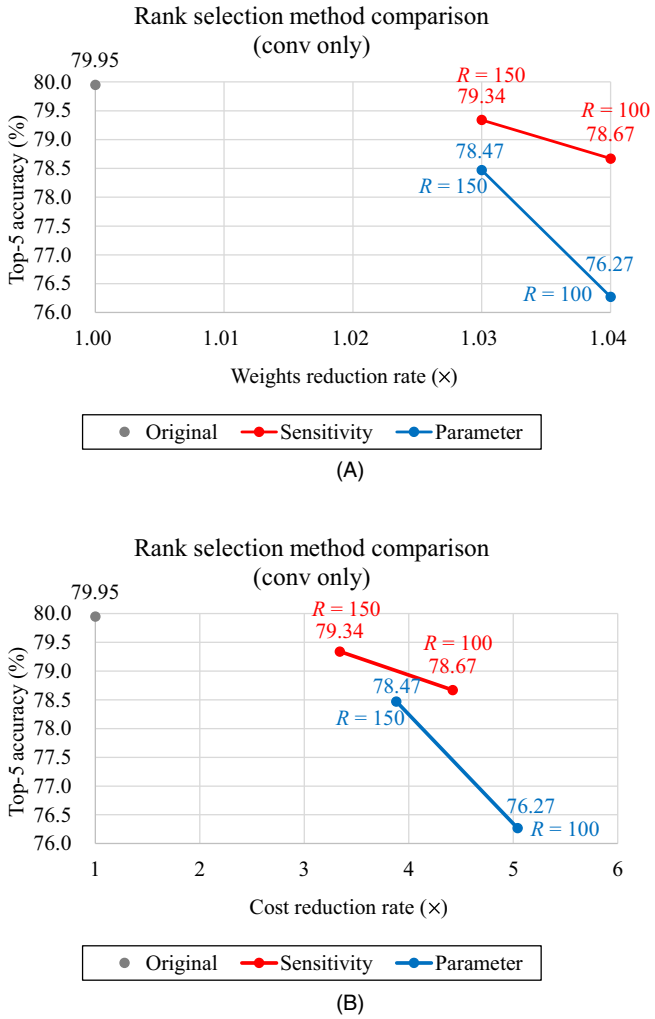

FIGURE 11 Top-five accuracy and (A) weight and (B) computation cost reduction rates of sensitivity and parameter rank-selection methods

method combines spatial dimensions because the spatial D dimensions in convolution kernels are typically very small (for example, 3×3 or 5×5), and thus maintaining the original does not make a noticeable difference. Kim and others [12] applied Tucker decomposition, and Wang and Cheng [21] applied block-term decomposition to decompose CNNs. However, because of the existence of a core tensor, Tucker-based approaches generally result in less compression than the CP approach.

CP-decomposition does not have a core tensor, and thus is expected to have a more compressed representation for the original tensors. However, as reported by Lebedev and others [13], it also has a critical drawback, namely the CP instability [12,13,23], which we believe prevents all-layer decomposition. Zhang and others [24] utilized matrix factorization, which is also a low-rank approximation, but considers the weights as a two-way tensor.

In addition to tensor decomposition approaches, there are various approaches to compressing CNNs. Pruning less-important weights is applied by Han and others [25]. Rastegari and others [26] uses a more extreme quantization by changing the weights to binary. Han and others [27] combines pruning, scalar quantization, and encoding in a single technique called deep compression. Yoon and Hwang [28] enforce sparsity on the filter through regularization. Chen and others also explored the frequency domain for compression [29]. However, all of these methods are more complex when implemented using standard deep-learning tools, which was actually the reason why we did not choose them as candidates for our approach.

Parallel to existing compression networks, several compact networks have been made from scratch. SqueezeNet [30] achieves AlexNet level accuracy on ImageNet with 50 times less parameters. MobileNets [31] and Xception [32] are used to create small networks based on a depth-wise separable convolution. Although the building of a new network from scratch in this manner has shown good results, it cannot be directly used for already existing and trained networks. Considering that researchers are currently reusing pre-existing networks at an increasing rate, decomposition approaches such as ours are expected to become more important.

5 | CONCLUSION

In this paper, we presented a method for decomposing DNNs. The method decomposes an entire network using a CP–SVD hybrid, optimizes the decomposition using the

TABLE 4 Compression results in Android smartphone deployment

Network	Method	Loading time	Forward time	Startup memory	Running memory
AlexNet	Original	3,904.40	2,309.70	259	314
	CP-sensitivity (Ave rank: conv 150, FC 300)	903.40 (×4.32)	1,041.92 (×2.22)	60 (×4.32)	128 (×2.45)
	CP-sensitivity (Ave rank: conv 100, FC 300)	886.81 (×4.40)	955.32 (×2.42)	60 (×4.32)	100 (×3.14)
GoogleNet	Original	1,595.32	3,333.15	45	157
	CP-sensitivity (Ave rank: conv 35)	512.49 (×3.11)	1,978.62 (×1.68)	42 (×1.07)	143 (×1.10)
VGG16	Original	Cannot be loaded			
	CP-sensitivity (Ave rank: conv 80, FC 150)	338.49	3,007.79	57	385

TPM, and fine-tunes the network using a hybrid of iterative and one-shot fine-tuning. By applying the approach to several representative CNNs such as AlexNet, GoogleNet, and VGG16, we showed that such an approach is generic. We also described extensive experiments conducted to answer several issues regarding rank selection, weight freezing, channel and spatial dimension decomposition, and the importance of both the TPM and iterative fine-tuning in the use of all-layer decomposition.

Our proposed method has several advantages. In contrast to other CP-based approaches, it can be used for all-layer decomposition. It also has a higher reduction in weights and less computational cost than other low-rank approximation methods, such as Tucker. The final point, which is one of practical importance, is that the proposed method can be seamlessly supported by all popular deep-learning toolkits such as PyTorch, Tensorflow, and Caffe.

In addition to such advantages, we hope that these extensive experiments can be used as guidelines for the various decision points that will also be encountered by practitioners or researchers who are interested in making large networks as compact as possible. As future work, we hope to explore a more systematic method for determining the ranks for decomposition. Although sensitivity-based rank selection has been empirically proven to work well, arbitrariness remains when determining the average rank.

ACKNOWLEDGEMENTS

This work was supported by the Institute for Information and Communications Technology Promotion (IITP) of the Korean government (MSIT) (No. 2017-0-00067, Development of ICT Core Technologies for Safe Unmanned Vehicles).

ORCID

Marcella Astrid http://orcid.org/0000-0003-1432-6661
Seung-Ik Lee http://orcid.org/0000-0003-2986-7540

REFERENCES

1. L. Zhang, Multi-view approach to specify and model aerospace cyber-physical systems, *Int. Conf. Autom. Comput. (ICAC)*, London, UK, Sept. 13–14, 2013, pp. 1–6.
2. S. A. Haque, S. M. Aziz, and M. Rahman, *Review of cyber-physical system in healthcare*, Int. J. Distrib. Sens. Netw. **10** (2014), no. 4, 217–415.
3. L. Monostori et al., *Cyber-physical systems in manufacturing*, CIRP Ann.-Manuf. Technol. **65** (2016), no. 2, 621–641.
4. K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv: 1409.1556.
5. C. Szegedy et al., Going deeper with convolutions, *IEEE Conf. Comput. Vision Pattern Recog.* Boston, MA, USA, June 7–12, 2015, pp. 1–9.
6. A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet classification with deep convolutional neural networks, *Adv. Neural Inform. Process. Syst.*, Lake Tahoe, NV, USA, Dec. 3–8, 2012, pp. 1097–1105.
7. O. Russakovsky et al., *ImageNet large scale visual recognition challenge*, Int. J. Comput. Vision **115** (2015), no. 3, 211–252.
8. J. Redmon and A. Farhadi, YOLO9000: Better, faster, stronger, *IEEE Conf. Comput. Vision Pattern Recog.*, Honolulu, HI, USA, July 21–26, 2017, pp. 6517–6525.
9. W. Liu et al., SSD: Single shot MultiBox detector, *Eur. Conf. Comput. Vision*, Amsterdam, Netherlands, Oct. 8–16, 2016, pp. 21–37.
10. C. Chen et al., DeepDriving: Learning affordance for direct perception in autonomous driving, *IEEE Int. Conf. Comput. Vision*, Santiago, Chile, Dec. 7–13, 2015, pp. 2722–2730.
11. M. Denil et al., Predicting parameters in deep learning, *Proc. Int. Conf. Neural Inform. Process. Syst.*, Lake Tahoe, NV, USA, Dec. 5–10, 2013, pp. 2148–2156.
12. Y.-D. Kim et al., Compression of deep convolutional neural networks for fast and low power mobile applications, 2015, arXiv preprint arXiv: 1511.06530.
13. V. Lebedev et al., Speeding-up convolutional neural networks using fine-tuned CP-decomposition, 2014, arXiv preprint arXiv: 1412.6553.
14. A. Paszke et al., Automatic differentiation in PyTorch, *Conf. Neural Inform. Process. Syst.*, Long Beach, CA, USA, Dec. 2017, pp. 1–4.
15. M. Abadi et al., TensorFlow: *Large-scale machine learning on heterogeneous systems*, 2015, available at http://tensorflow.org.
16. Y. Jia et al., Caffe: Convolutional architecture for fast feature embedding, *Proc. ACM Int. Conf. Multimed.*, Orlando, FL, USA, Nov. 3–7, 2014, pp. 675–678.
17. G. Golub and W. Kahan, *Calculating the singular values and pseudoinverse of a matrix*, J. Soc. Ind. Applicat. Math., Series B: Numer. Anal., **2** (1965), no. 2, 205–224.
18. G. Allen, *Sparse higher-order principal components analysis*, JISTATS, **15** (2012), 27–36.
19. C. J. Hillar and L.-H. Lim, *Most tensor problems are NP-hard*, J. ACM, **60** (2013), no. 6, 1–39.
20. M. Astrid and S.-I. Lee, CP-decomposition with tensor power method for convolutional neural networks compression, *IEEE Int. Conf. Big Data Smart Comput.*, Jeju Island, Rep of Korea, Feb. 13–16, 2017, pp. 115–118.
21. P. Wang and J. Cheng, Accelerating convolutional neural networks for mobile applications, *Proc. ACM Multimed. Conf.*, vAmsterdam, Netherlands, Oct. 15–19, 2016, pp. 541–545.
22. sh1r0, Caffe-android-lib, 2014. https://github.com/sh1r0/caffe-android-lib.
23. V. De Silva and L.-H. Lim, *Tensor rank and the ill-posedness of the best low-rank approximation problem*, SIAM J. Matrix Anal. Applicat. **30** (2008), no. 3, 1084–1127.
24. X. Zhang et al., *Accelerating very deep convolutional networks for classification and detection*, IEEE Trans. Pattern Anal. Mach. Intell. **38** (2016), no. 10, 1943–1955.
25. S. Han et al., Learning both weights and connections for efficient neural network, *Adv. Neural Inform. Process. Syst.*, Montreal, Canada, Dec. 7–12, 2015, pp. 1135–1143.

26. M. Rastegari et al., XNOR-net: ImageNet classification using binary convolutional neural networks, *Eur. Conf. Comput. Vision*, Amsterdam, Netherlands, Oct. 8–16, 2016, pp. 525–542.
27. S. Han, H. Mao, and W. J. Dally, Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding, *Int. Conf. Learning Representations*, San Diego, CA, USA, May 7–9, 2015.
28. J. Yoon and S. J. Hwang, Combined group and exclusive sparsity for deep neural networks, *Int. Conf. Mach. Learning*, Sydney, Australia, Aug. 6–11, 2017, pp. 3958–3966.
29. W. Chen et al., Compressing convolutional neural networks, 2015, arXiv preprint arXiv: 1506.04449.
30. F. N. Iandola et al., SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 1 MB model size, 2016, arXiv preprint arXiv: 1602.07360.
31. A. G. Howard et al., MobileNets: Efficient convolutional neural networks for mobile vision applications, 2017, arXiv preprint arXiv: 1704.04861.
32. F. Chollet, Xception: Deep learning with depthwise separable convolutions, *IEEE Conf. Comput. Vision Pattern Recog.*, Honolulu, HI, USA, July 21–26, 2017, pp. 1800–1807.

AUTHOR BIOGRAPHIES

Marcella Astrid received her BS in computer engineering from the Multimedia Nusantara University, Tangerang, Indonesia, in 2015, and the MEng in computer software from the University of Science and Technology (UST), Daejeon, Korea, in 2017. At the same university, she is currently working toward her PhD degree in computer science. Her recent interests include deep learning and computer vision.

Seung-Ik Lee received his BS, MS, and PhD degrees in computer science from Yonsei University, Seoul, Korea, in 1997 and 2001, respectively. He is currently working for the Electronics and Telecommunications Research Institute, Daejeon, South Korea. Since 2005, he has been with the Department of Computer Software, University of Science and Technology, Daejeon, Korea, where he is now a professor. His research interests include machine learning, deep learning, and reinforcement learning.

APPENDIX

PARAMETERS OF ALEXNET, GOOGLENET, AND VGG16

Table 5 shows the parameters of AlexNet, GoogleNet, and VGG16. Only the parameters to be compressed are displayed.

IMPLEMENTATION OF CONVOLUTION OPERATION IN CAFFE

To fasten the computation in convolution, Caffe implements the convolution by converting the feature map and filters (in tensors) into a two-dimensional matrix because the matrix multiplication is optimized using basic linear algebra subprograms (BLAS). More specifically, with an input feature map tensor of size ($\_S \times \_H \times \_W$), and a convolution filter of size ($\_S \times \_D \times \_D$), the feature map is converted into a matrix of size ($\_S \times \_D \times \_D$) $\times$ ($\_H' \times \_W'$), and the filter is converted into a vector of size ($\_S \times \_D \times \_D$), as shown in Figure 12. They are then multiplied and give us a single output feature map of size ($\_H' \times \_W'$). Thus, a feature map conversion requires ($\_S \times \_D \times \_D$) $\times$ ($\_H' \times \_W'$) rearrangements for a single filter. If we have _T filters, then the number of rearrangements will be multiplied by $\_T$. Note that there is no need to so when a 1×1 filter is used because the feature map can be directly interpreted as a matrix without further rearrangements. However, if a filter other than a 1×1 filter is used, then a rearrangement is inevitable because of the overlapping that occurs when the convolution filter slides.

This rearrangement overhead becomes obvious, especially when the convolution operation is conducted using a network input layer that typically has three channels in the case of images.

For example, Table 5 shows that conv1 has $T = 96$, $S = 3$, $D = 11$, and $H' = W' = 55$. When we decompose conv1 with rank R, the input channels of a non-1×1 filter differ between two-way and three- or four-way decomposition. This is so because in two-way, the input channel of

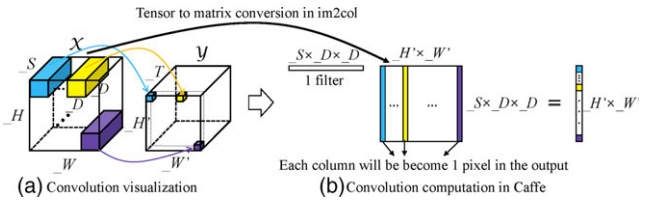

FIGURE 12 Convolution operation in Caffe

TABLE 5 Parameters of AlexNet, GoogleNet, and VGG16

	Group	*T*	*S*	*D*	*H*	*H′*	#Param	#Cost
AlexNet								
conv1	1	96	3	11	227	55	34,848	105,415,200
conv2	2	256	48	5	55	27	307,200	223,948,800
conv3	1	384	256	3	27	13	884,736	149,520,384
conv4	2	384	192	3	13	13	663,552	112,140,288
conv5	2	256	192	3	13	13	442,368	74,760,192
fc6	1	4,096	9,216	1	1	1	37,748,736	37,748,736
fc7	1	4,096	4,096	1	1	1	16,777,216	16,777,216
fc8	1	1,000	4,096	1	1	1	4,096,000	4,096,000
TOTAL							**60,954,656**	**724,406,816**
GoogleNet								
conv1	1	64	3	7	224	112	9,408	118,013,952
conv2	1	192	64	3	56	56	110,592	346,816,512
i3a	1	128	96	3	28	28	110,592	86,704,128
i3b	1	192	128	3	28	28	221,184	173,408,256
i4a	1	208	96	3	14	14	179,712	35,223,552
i4b	1	224	112	3	14	14	225,792	44,255,232
i4c	1	256	128	3	14	14	294,912	57,802,752
i4d	1	288	144	3	14	14	373,248	73,156,608
i4e	1	320	160	3	14	14	460,800	90,316,800
i5a	1	320	160	3	7	7	460,800	22,579,200
i5b	1	384	192	3	7	7	663,552	32,514,048
TOTAL decomposed layer only							**3,110,592**	**1,080,791,040**
TOTAL with other layers							**6,990,272**	**1,582,671,872**
VGG16								
conv1_1	1	64	3	3	224	224	1,728	86,704,128
conv1_2	1	64	64	3	224	224	36,864	1,849,688,064
conv2_1	1	128	64	3	112	112	73,728	924,844,032
conv2_2	1	128	128	3	112	112	147,456	1,849,688,064
conv3_1	1	256	128	3	56	56	294,912	924,844,032
conv3_2	1	256	256	3	56	56	589,824	1,849,688,064
conv3_3	1	256	256	3	56	56	589,824	1,849,688,064
conv4_1	1	512	256	3	28	28	1,179,648	924,844,032
conv4_2	1	512	512	3	28	28	2,359,296	1,849,688,064
conv4_3	1	512	512	3	28	28	2,359,296	1,849,688,064
conv5_1	1	512	512	3	14	14	2,359,296	462,422,016
conv5_2	1	512	512	3	14	14	2,359,296	462,422,016
conv5_3	1	512	512	3	14	14	2,359,296	462,422,016
fc6	1	4,096	2,5088	1	1	1	102,760,448	102,760,448
fc7	1	4,096	4,096	1	1	1	16,777,216	16,777,216
fc8	1	1,000	4,096	1	1	1	4,096,000	4,096,000
TOTAL							**138,344,128**	**15,470,264,320**

the non-1 × 1 filter, as shown in the first arrow in Figure 3C, is S (namely, $\_S = S = 3$). In three-way decomposition, there is one non-1 × 1 convolution operation, as shown in the second arrow in Figure 3B, and its number of input channels is R. In four-way decomposition, there are two non-1 × 1 convolution operations, as shown in the second and third arrows in Figure 3D, and their number of input channels is R. In other words, for example, if we decompose using $R = 50$, the two-way decomposition has $(3 \times 11 \times 11) \times (55 \times 55) \times 50 = 54.9$ M operations, the three-way decomposition has $(50 \times 11 \times 11) \times (55 \times 55) \times 50 = 915$ M operations, and the four-way decomposition has $(50 \times 1 \times 11) \times (227 \times 55) \times 50 + (50 \times 11 \times 1) \times (55 \times 55) \times 50 = 426$ M operations. Therefore, the two-way decomposition performs better in this case. This disadvantage of Caffe in a depth-wise convolution (convolution with a filter depth of 1) has also been addressed through Github issue number 5649.