IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 20, 2019, accepted March 9, 2019, date of publication March 14, 2019, date of current version April 5, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2905138

Sensitivity-Oriented Layer-Wise Acceleration and Compression for Convolutional Neural Network

WEI ZHOU, YUE NIU, AND GUANWEN ZHANG
School of Electronics and Information, Northwestern Polytechnical University, Xi'an 710072, China

Corresponding author: Wei Zhou (zhouwei@nwpu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61602383, Grant 61772424, and Grant 61702418, and in part by the Fundamental Natural Science Research Funds of Shaanxi Province under Grant 2017JQ6019 and Grant 2018JQ6016.

ABSTRACT Convolutional neural networks (CNNs) have achieved excellent performance improvement in image processing and other machine learning tasks. However, tremendous computation and memory consumption for most classical CNN models pose a great challenge to the deployment in portable and power-limited devices. In this paper, by analyzing the sensitivity of the rank in each layer of the network accuracy, we propose a sensitivity-oriented layer-wise low-rank approximation algorithm. With specific compression and acceleration requirement, the convolutional layer with higher *sensitivity* keeps more kernels than that with lower *sensitivity*. In addition, we also demonstrated that global optimization can obtain a better classification performance than layer-wise fine-tuning. The experimental results show that the proposed method can achieve 20% acceleration ratio gaining compared with the traditional rank-reducing methods. When deployed on the VGGNet-16 model, the proposed method can achieve 2.7× compression/acceleration ratio on convolutional layers and 10.9× compression/acceleration ratio on fully connected (FC) layers with 0.05% top-1 accuracy loss and 0.01% top-5 accuracy loss.

INDEX TERMS CNN, layer-wise sensitivity, compression, acceleration.

I. INTRODUCTION

Convolutional neural networks(CNNs) have been the most striking approach in many machine learning tasks, e.g., image classification and detection, with extraordinary accuracy and generalization ability compared to traditional algorithm, like SVM [1]. In 2012, the famous AlexNet CNN model [2] was proposed which improved classification accuracy by nearly 10 percent. Furthermore, classification performance continues to improve in recent years. From 2014 to 2016, 93.2%, 96.3% and 96.9% top-5 accuracy was achieved by VGGNet [3], ResNet [4] and Inception-v4 [5] respectively.

Compared to traditional algorithm, CNNs possess powerful ability of feature extraction and combination. Various levels of features can be extracted by multiple convolutional layers. CNNs also have high generalization ability to handle different oriented task, while the traditional algorithms can only mange some specific task under specific circumstances. Therefore, CNNs have been widely used in many applications, such as image classification, detection [6]–[8], semantic segmentation [9], [10], speech emotion recognition [11], [13], feature learning and face recognition [14], and image assessment [12], [15].

At the same time, CNNs introduce tremendous computational complexity and memory requirement. Generally, the convolution is the most complex operation in CNNs which takes up much of computation resource. For one convolutional layer with input image of $m \times L \times L$ (m, input channel size) and kernels of $m \times l \times l \times n$ (n, output channel size), the convolutional operation will carry out $m \times l^2 \times L^2 \times n$ float additions and multiplications in total. In addition, most of parameters exist in Fully-Connected (FC) layer, which dominate the whole network size. As an example, VGGNet-16 (16 parametric layers) consists of 138 million parameters with 124 million fully-connected parameters and 15.6 GFLOPS with 15.5 convolutional GFLOPS during one single-image forward processing. Computations involved in *forward* process and parameters of several classical network are shown in Table 1, where M indicates million while B indicates Billion.

The associate editor coordinating the review of this manuscript and approving it for publication was Hong-Mei Zhang.

TABLE 1. Computations and parameters of several classical networks.

	AlexNet	VGGNet-16	ResNet-152	Inception-v4
Comp.	650M	15.6B	15.3B	5.3B
Param.	60M	138M	118M	13M

Due to high computational complexity and memory storage, power consumption is also much exorbitant. Statistically, the average power consumption is 180W during running VGGNet-16 by Caffe [35] in GPU platform. Things becomes much worse when it comes to portable devices where power and computation resources are very limited. In some embedded system, in order to reduce power consumption when processing *forward* computation in CNNs, computation per second was limited at a very low level.

Our experiments revealed that, the sensitivity of the rank of kernel matrix to the network accuracy is a layer-wise variable. In some networks, the final classification accuracy varies greatly when certain amount of kernels are removed in each convolutional layer. In consequence, the rank of kernel matrices can be further reduced for accuracy insensitive layer. In addition, fine-tuning network is necessary to restore the ability of feature extraction. In our experiment, layer-wise optimization shows that different convolutional layer has different restoration ability after fine-tuning. According to this observation, we first fine-tune the layer with least restoration ability, then the layer with stronger restoration ability. The proposed method is evaluated on several classical models (e.g., AlexNet [2], VGGNet-16/19 [3], OverFeat [32]) on ImageNet.

The rest of this paper is organized as follows. Related works are briefly summarized in Sec. II. The proposed sensitivity oriented compression method is introduced in Sec. III. Sec. IV presents experiment results. Finally, conclusion is drawn in Sec. V.

II. RELATED WORKS

In the literature, many studies on accelerating and downsizing CNNs have been done. These methods can be divided into two groups: network pruning and low-rank approximation. Network pruning [18]–[21] remove connections or neurons with small values which are supposed to have no distinct effects to the whole network performance. By this way, up to 95.4% parameters in convolutional and FC layers can be reduced with about 1% top-5 error increasing. This method is effective in reducing memory requirement for large CNN model. For example, the model size of VGGNet-16 can be reduced by 92%, with 10× acceleration ratio on FC layers [19]. Furthermore, weights in all connections are grouped into harsh buckets, which can also greatly reduce storage overhead [21].

On the other hand, Low-rank approximation has been successfully applied to many tasks, like dimensionality reduction [22], image restoration [23]. Reference [23] exploits low-dimensional structure in images with high-dimensional data. While, for CNNs, low-rank characteristics for convolutional kernels were exploited to reduce redundancies among convolutional kernels [24]–[27]. In many networks, convolutional kernels in the same layer often extract similar features of inputs. Algorithms based on low-rank aim to remove the redundancy between similar kernels. In these papers, original convolutional kernels was approximated with fewer *uncorrelated basis* which was obtained by principle component analysis or iterative method. Normally, different from network pruning, the ratio of computation reduction is directly proportional to ratio of parameter compression in low-rank approximation. The whole network can still operate with high parallelism, which makes it friendly to VLSI accelerator design [28]–[30]. However, the compression ratio of low rank approximation is inferior to network pruning. For example, the compression ratio in literature [19] is about 2×, while top-5 error would increase about 0.9%.

Besides the above works, there are some other algorithms were also proposed to compress and accelerate CNN. Reference [31] replaced the original convolutional kernels with a perforated one, in which connections with small weights were masked. During computation, the proposed method can skip the evaluation in these spatial positions. As a result, AlexNet and VGGNet-16 can be accelerate by a factor of $2\times \sim 4\times$.

III. ALGORITHM

A. LOW-RANK CHARACTERISTIC REVIEW

According to [33], CNN models are overparameterized and contain redundancies in convolutional kernels and FC neurons. Taking VGGNet-16 as an example, output feature maps in 1th convolutional layer only consists of background features for input image extracted by some convolutional kernels, which are barely useful to final classification, as shown in Fig. 1(a), and what we truly need is the information of foreground, like cat features in image.

On the other side, as shown in Fig. 1(b), there are strong resemblance among output feature maps corresponding to kernels in one convolutional layer, and features extracted by each convolutional kernels are similar, such as eyes and other edge information. This similarity indicates the closely correlation among convolutional kernels.

The following is a method to evaluate the correlation between two output feature maps p and q:

$$corr(p,q) = \frac{\sum_{i,j}(f_{p_{i,j}} - \bar{f}_p)(f_{q_{i,j}} - \bar{f}_q)}{\sqrt{\sum_{i,j}(f_{p_{i,j}} - \bar{f}_p)^2 \sum_{i,j}(f_{q_{i,j}} - \bar{f}_q)^2}}, \quad (1)$$

where f_p and f_q indicate the feature maps in the same convolutional layer, $\bar{f}_p$ and $\bar{f}_q$ are the average of f_p and f_q respectively, i and j are pixel indexes.

Fig. 1(c) shows the correlation statistics from the output feature maps in the 1st layer of VGGNet-16. It can be seen that the ratio of feature maps with $corr > 0.8$ is larger than 60%. For one convolutional layer, suppose there are n convolutional kernels with dimension $m \times l \times l$. m indicates channels for input feature map and l indicates kernel size. Then, two-dimensional kernel matrix K is constructed by expanding $m \times l \times l$ into one column vector, and matrix

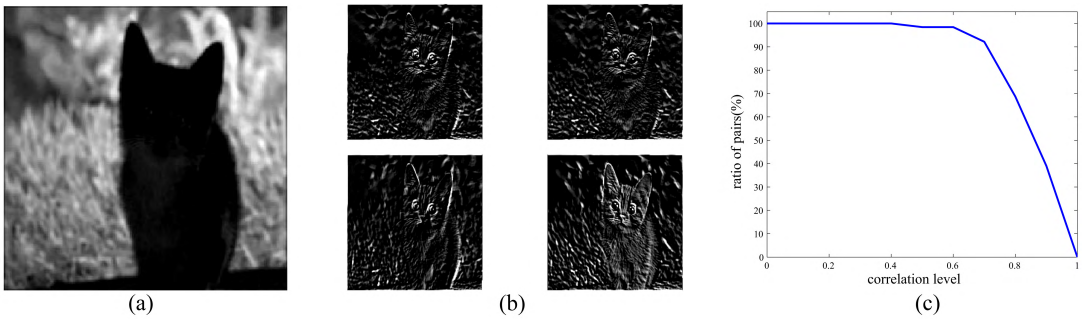

FIGURE 1. From the left are (a) the feature map only contains background information, (b) the similar feature maps, and (c) the correlation among convolutional kernels. The result is obtained by using 1th convolutional layer of VGGNet-16.

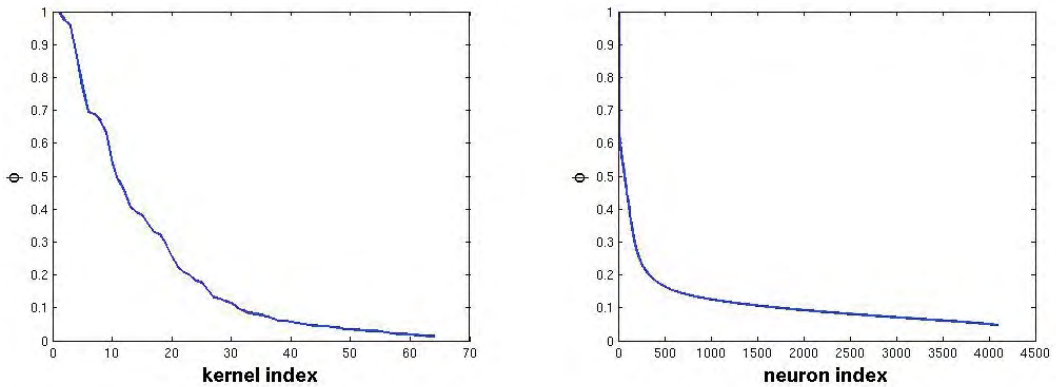

FIGURE 2. Importance for convolutional kernel of 2th convolutional (a) and importance for fully-connected neurons of FC6 (b) in VGGNet-16. We can see that both convolutional kernels and fully-connected neurons have much redundancies.

columns indicate the number of output channels. The metric importance for kernels is calculated as Eq. (2):

$$K = U \times S \times V^T, \tag{2}$$

and together as Eq. (3) :

$$\phi(j) = \frac{S(j,j)}{\sum_i S(i,i)}, \tag{3}$$

where, S indicates variance of principal component for kernel matrix K in row-wise. $\phi(j)$ is the quantitative *importance* for jth component. The similar operation can be applied to FC layer to obtain *importance* for connection neurons.

As show in Fig. 2(a), we can observe that just fewer important convolutional kernels can extract requisite features for final classification in 2th convolutional layer, and original output feature maps can be combined with these principal components due to high correlation among original kernels. The importance curve for neurons contained in FC6 layer for VGGNet-16 is shown in Fig. 2(b). We can see that there are a few essential neurons predominate in FC layer. Original neural outputs can be similarly reconstructed with linear combination of these essential neurons as convolutional layer.

B. NETWORK DECOMPOSITION AND RECONSTRUCTION

Based on the low-rank property elaborated above, the CNNs can be decomposed into kernels/neurons with low ranks, and can be reconstructed with linear combination of these "basis", which can speed up network significantly. This procedure can be expressed as following Eq. (4):

$$\Phi_{orig} = C \times \Phi_{basis}, \tag{4}$$

where Φ_{orig} is original convolutional kernels in convolutional layer or original neurons in FC layer, Φ_{basis} is principal kernels in convolutional layer and principal neurons in FC layer, and C is coefficient for linear combination. For VGGNet-16, up to 85% computation time is consumed on convolutional layer, and FC layer dominates the model size (about 90%). So, decomposition to convolutional layer and FC layer can notably speed up and compress the whole network respectively.

1) KERNEL DECOMPOSITION AND RECONSTRUCTION

The convolutional layer is the most important component of CNNs, which consists of small quantity of kernel parameters, but majority of computations on the contrary. It is necessary to exploit the potential of low-rank property to speed up CNNs. Normally, the weight of convolutional kernel is a 4-D tensor $K_{l,l,m,n}$, where l is kernel size, m is the number of

input channel, and n is the number of output channel. We first expand kernel matrix on one output channel into column vector $\boldsymbol{k}$. Correspondingly, original 4-D tensor $K_{l,l,m,n}$ is converted to a 2-D matrix $W = (\boldsymbol{k}_1, \boldsymbol{k}_2, ...\boldsymbol{k}_n)$.

Then we decompose W according to Eq. (2), where we consider U as principal kernel matrix, in which different column vector is a *basis* kernel with different *importance*. So, we can approximate original W by removing less important "basis" as follows:

$$\widetilde{W} = U_{:,1:p} \times S_{1:p,1:p} \times V^T_{:,1:p}. \tag{5}$$

In Eq. 5, $\widetilde{W}$ is an approximated form of W and p is the number of reserved convolutional kernels. Furthermore, we slightly modify this equation as in Eq. (6):

$$\begin{aligned}\widetilde{W} &= (U_{:,1:p} \times \sqrt{S_{1:p,1:p}}) \times (\sqrt{S_{1:p,1:p}} \times V^T_{:,1:p}) \\ &= P \times Q^T. \end{aligned} \tag{6}$$

The original convolutional operation with matrix production $I \times W$ will be transformed into $I \times P \times Q^T$ approximately. And the ratio of computation complexity reduction is $nl^2m/(pl^2m + pn)$. The whole convolutional layer will be accelerated significantly if $p \ll n$.

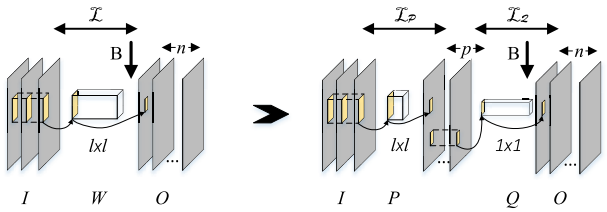

FIGURE 3. Illustration for the convolutional layer decomposition. *I* and *Q* are input and output feature maps, while *P* and *Q* are corresponding to kernels of first and second decomposed layers.

Problems will occur if we further do optimization to $\widetilde{W}$, like fine-tuning. Since it is possible that $\widetilde{W}$ can revive its original form W after fine-tuning. Thus, we decompose single original convolutional layer $\mathscr{L}$ into two new convolutional layer $\mathscr{L}_\mathscr{P}$ and $\mathscr{L}_\mathscr{Q}$, parameterized with P and Q respectively. The P is a matrix with size of $l^2m \times p$ and Q is a matrix with size of $p \times n$. Therefore, the convolutional layer $\mathscr{L}_\mathscr{P}$ consists of p filters and each filter has the size of $l \times l \times m$ (i.e., the same size as that of the original filters of W), while the convolutional layer $\mathscr{L}_\mathscr{Q}$ consist of n filters and each filter has the size of $1 \times 1 \times p$. As shown in Fig. 3, after decomposition, $\mathscr{L}_\mathscr{P}$ produces a feature map with p channels. And then $\mathscr{L}_\mathscr{Q}$ performs 1×1 convolution on basis of the output feature map and produces a feature map with n channel.

Furthermore, we can apply optimization to $\mathscr{L}_\mathscr{P}$ and $\mathscr{L}_\mathscr{Q}$ without concerned problem mentioned above. Since the combination of fine-tuned layer $\mathscr{L}_\mathscr{P}$ and $\mathscr{L}_\mathscr{Q}$ still have the low-rank characteristic compared to original layer $\mathscr{L}$.

2) NEURON DECOMPOSITION AND RECONSTRUCTION

Most CNNs are designed to be a compact form with as many parameters as possible to represent probability distribution of input image, nevertheless, most connection weights are marginally useful or highly correlated. Therefore, neurons and connections in FC layer also have many redundancies in most classical networks, like VGGNet. Since most of the model parameters exist in FC layer, it is important to reduce network model by exploiting the redundancy of neurons in FC layer.

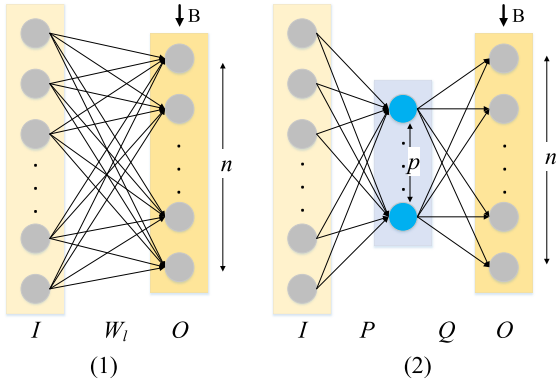

FIGURE 4. Illustration for the FC layer decomposition. *I* and *O* are input and output data, while *P* and *Q* are responding to the matrix for first and second decomposed layers.

The weight matrix in fully-connected layer is a 2-D dimensional matrix, represented by W. Each column vector $\boldsymbol{l}$ of W, represents a connected neuron. Similarly, W can also be decomposed according to Eq. (6), and here, we consider U as principal neurons, in which different column vector is a *basis* neuron with different *importance*. Inner production in current layer will be changed from $I \times W$ to $I \times P \times Q^T$. The ratio of parameter reduction is $mn/(mp + pn)$, and it will dramatically reduce the model size if $p \ll n$. Similarly, current FC layer $\mathscr{L}$ is decomposed into two new layer $\mathscr{L}_\mathscr{P}$ and $\mathscr{L}_\mathscr{Q}$. As shown in Fig. 4, the first layer $\mathscr{L}_\mathscr{P}$ has p output neurons without bias term, the second layer $\mathscr{L}_\mathscr{Q}$ is also a FC layer with n output neurons corresponding to original bias of current layer. Compared to network pruning method, which remove inessential connections (normally connection weights with small value), redundant neurons are removed in this manner. Though the ratio of model parameter reduction in this design cannot be comparable with method in network pruning, parallel computation can be easier to realize in low-rank design. In network pruning method, index of weights must be stored for indexing and computation, which is not necessary in low-rank design.

C. SENSITIVITY-ORIENTED RANK DECISION

It is unwise to roughly determine rank for convolutional layer and FC layer according to *importance* curve, since we cannot discriminate the relative redundancy among different layers only by *importance curve*. Furthermore, we should decide the tradeoffs between accuracy and computation complexity (convolutional layer), as well as model size (FC layer) according to redundancy of different layers. We should be careful that the degree of redundancy varies from different layers.

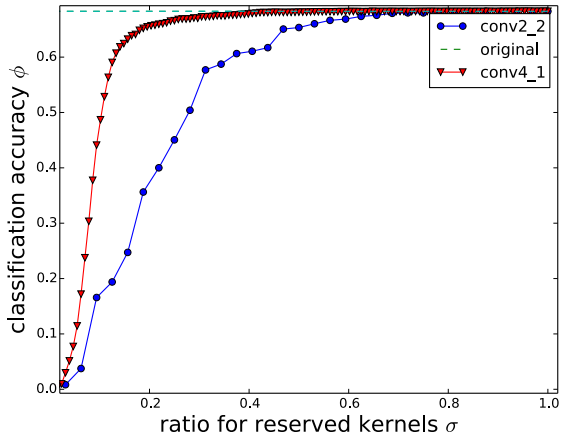

FIGURE 5. Sensitivity curve for conv2_2 (blue line) and conv4_1 (red line) layer in VGGNet-16. The dash line is the original classification accuracy as a benchmark.

In this paper, we propose a sensitivity metric ζ to explore the relationship between the rank p and the network performance. To define sensitivity ζ, accuracy-rank curve is used to explore the relationship between the ratio σ of the reserved kernels and classification accuracy ϕ in one convolutional layer. Fig. 5 shows the relationship between classification accuracy and the number of reserved kernels for conv2_2 and conv4_1 layers in VGGNet-16. From the result, the classification accuracy degrading of conv2_2 is much bigger than that of conv4_1, when the same proportion of kernels are removed. Therefore, conv2_2 is definitely more sensitive than conv4_1 in the sense of accuracy loss. To improve compression/acceleration performance, we should reserve more kernels in conv2_2 than that in conv4_1 layer.

To better describe convolutional layer sensitivity, ζ for ith layer is defined as:

$$\zeta_i = \frac{1}{\int_0^1 \phi_i(\sigma) d\sigma}, \tag{7}$$

where $\phi_i(\sigma)$ is the classification accuracy and σ is the ratio of reserved kernels/neurons. Since ζ cannot be analytically resolved, we transform it into numerical form as:

$$\widetilde{\zeta}_i = \frac{1}{\sum_j \phi_i(\sigma_j)\Delta\sigma}. \tag{8}$$

When *sensitivity* $\widetilde{\zeta}_i$ has been solved, the average *sensitivity* $\bar{\zeta}$ over network layers is then calculated. And normalization will be applied to $\widetilde{\zeta}_i$ according to $\bar{\zeta}$. If the required acceleration ratio is $\eta\times$ ($\eta > 1$) times as original, the rank p for each layer is decided as follows:

$$p = \frac{\widetilde{\zeta}}{\eta\bar{\zeta}_i} n_i, \tag{9}$$

where n_i is the number of original kernels for ith layer.

D. GLOBAL NETWORK OPTIMIZATION

The main problem of low-rank approximation is that it may be not able to represent features of principal kernels and to combine features of principal FC neurons efficiently. Hence, it is necessary to drive the convolutional kernels and FC neurons to improve feature representation and combination ability. Network optimization has been studied by many researches, such as [24] and [25]. These researches mainly tried to minimize reconstructed error of output channels layer by layer, while high correlation of convolutional kernel or FC neurons among layers were not taken into consideration. Global networks optimization was tried in [26]. However, due to the inserted layers involved in the fine-tuning, the gradient explosion issue makes the training trapped in ill-conditions. Different from the method in [26], only two new layers are introduced for each layer during the optimization process. And as a result, the kernels and neurons in different layers are able to be optimized jointly. Benefiting from our proposed method, the overall network optimization can then be successfully carried out.

IV. EXPERIMENTS

This section experimentally shows the practicability of the proposed method, as well as comparison with other acceleration/compression algorithms. Four network models with relatively large amount of parameters and computations are chosen in these experiments which are AlexNet, OverFeat, VGGNet-16, VGGNet-19 respectively. These networks are suitable for compressing and accelerating due to their high redundancy.

We employ top-1 and top-5 error as comparison metric based on the dataset of ImageNet [34]. As the reference, the original classification top-1 accuracy for AlexNet, OverFeat, VGGNet-16, VGGNet-19, on the single-view of test dataset with 50000 image is 58%, 60.02%, 68.28%, 68.47% respectively. The framework we used for training and testing is Caffe [35] and Torch [36] toolkit. Detailed codes are published in Github (https://github.com/Yue-Niu/CNN.CompAcc).

Layer-wise performance analysis is implemented first to evaluate the feasibility for algorithms on a single layer. Then, the proposed method is applied on the whole network based on the sensitivity of each layer, and the performance is compared with other algorithms.

A. LAYER-WISE PERFORMANCE

Layer-wise performance analysis shows influences after decomposing a single layer (convolutional layer or FC layer). We first decompose and reconstruct networks directly by removing specified number of kernels or neurons according to Sec. III-B, and fine-tune the whole reconstructed net. By analyzing the single-layer fine-tuning, we would like to know: how much performance of feature extraction can be restored after decomposing single layer? and how does the distribution of weights change after fine-tuning a single layer?

We fine-tune each specific single layer-cluster (i.e., all convolutional layers in same stage) of conv1 (C1), conv2 (C2), conv3 (C3), conv4 (C4), and conv5 (C5) on VGGNet-16 and VGGNet-19. We evaluate the performance with certain

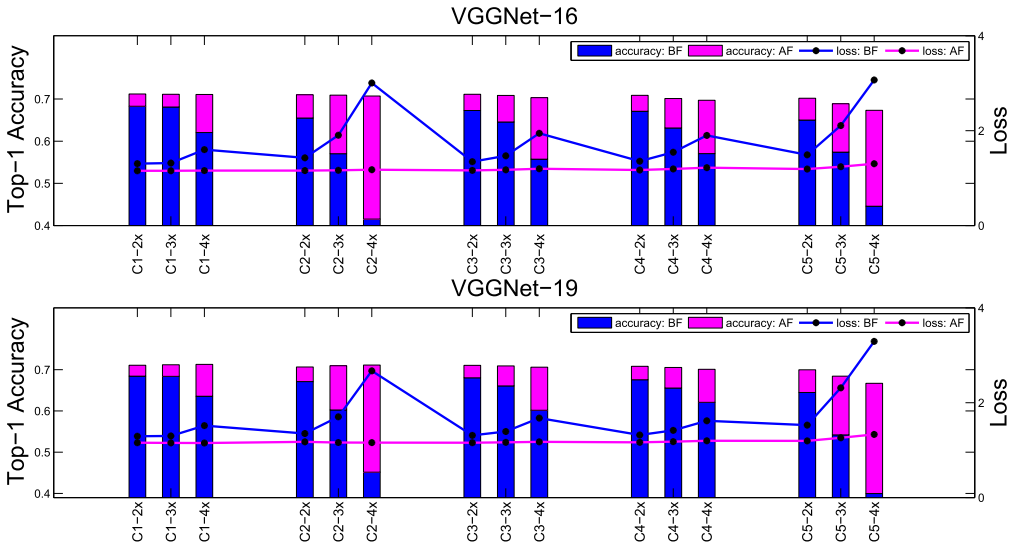

FIGURE 6. Layer-wise fine-tuning for VGGNet-16 and VGGNet-19. The blue bars and lines indicate classification accuracy and loss before fine-tuning respectively, and the pink indicates accuracy and loss after fine-tuning. Before fine-tuning, classification capability degrades little in 2× and 3×, while in 4×, the capability downgrades drastically. However, after fine-tuning, classification capability can be restored in all these three situations. We can see that both classification accuracy and loss are almost the same as the original model.

quantity of kernels and reserve half, third, and quarter *principal kernels* for each single convolutional respectively, i.e., with the ideal acceleration ratio as 2×, 3×, and 4×. The results are depicted in Fig. 6. In the Fig. 6, the blue bar and purple bar indicate the classification accuracy before fine-tuning (BF) and after fine-tuning (AF) respectively, while the blue line and purple line indicate the loss before fine-tuning (BF) and after-tuning (AF) respectively. As shown in Fig. 6, the classification accuracy can be significantly improved after fine-tuning. Different layer varies obviously in classification improvement due to different redundancy. Classification capability can be restored significantly in each of the three situations. One interesting fact is that the capability of high level layers (such as C5) is more difficult to be restored than that of the low lever layers (such as C1). An interesting thing is that capability of high level (such as C5) is a little more difficult to be restored than the low level (such as C1. Since the kernels in low level layers mainly extracts general features of input images, which are more robust during compression, while kernels in high level layers extracts high level features, which are likely more sensitive to compression. In addition, for most layers in VGGNet-16 and VGGNet-19, more than half kernels in convolutional layers and up to 90% neurons in FC layers can be removed without obvious accuracy decline after fine-tuning.

By comparing the distribution of network before and after global fine-tuning and partial fine-tuning (i.e., only fine-tuning the current decomposed layer), we can obtain the distribution difference curve. Fig. 7 shows the distribution difference of weights after global fine-tuning (Global Opt.) and partial fine-tuning (Partial Opt.) for FC6 in VGGNet-16. We can see that the distribution of weights after global fine-tuning changes slightly in current decomposed layer, meanwhile, the classification performance is better after global fine-tuning. The reason for significant accuracy improvement is that the other layers were optimized jointly during fine-tuning. Therefore, the kernels and neurons in all layers can be optimized. However, in paper [26], optimization is just applied on single layer without considering joint optimization on all layers. The other layers in VGGNet-16/19 share similar characteristics, as well as other network models.

TABLE 2. Different acceleration/compression configurations for VGGNet-16. ζ indicates sensitivity for each layer.

	ζ	number of reserved kernels		
		2×	2.5×	3×
conv1_1	*	*	*	*
conv1_2	1.649	31	25	21
conv2_1	1.626	62	50	42
conv2_2	1.885	72	58	50
conv3_1	1.684	128	103	86
conv3_2	1.651	126	101	84
conv3_3	1.715	131	105	88
conv4_1	1.619	247	198	165
conv4_2	1.619	247	198	165
conv4_3	1.653	252	202	169
conv5_1	1.711	261	209	175
conv5_2	1.666	254	203	170
conv5_3	1.661	253	203	170

B. WHOLE NETWORK PERFORMANCE

The whole network performance analysis is carried out on VGGNet-16, VGGNet-19 models. The Fig. 8 (a) and Fig. 8 (b) show the sensitivity curves for VGGNet-16 and VGGNet-19 respectively. We focus on the reduction factors with 2×, 2.5× and 3×. The number of reserved kernels are obtained according to Eq. (9), and results are summarized in Table 2. Since there are great amount of redundancies in FC layers, we only keep 256 *principal neurons* for FC6 and FC7 layers.

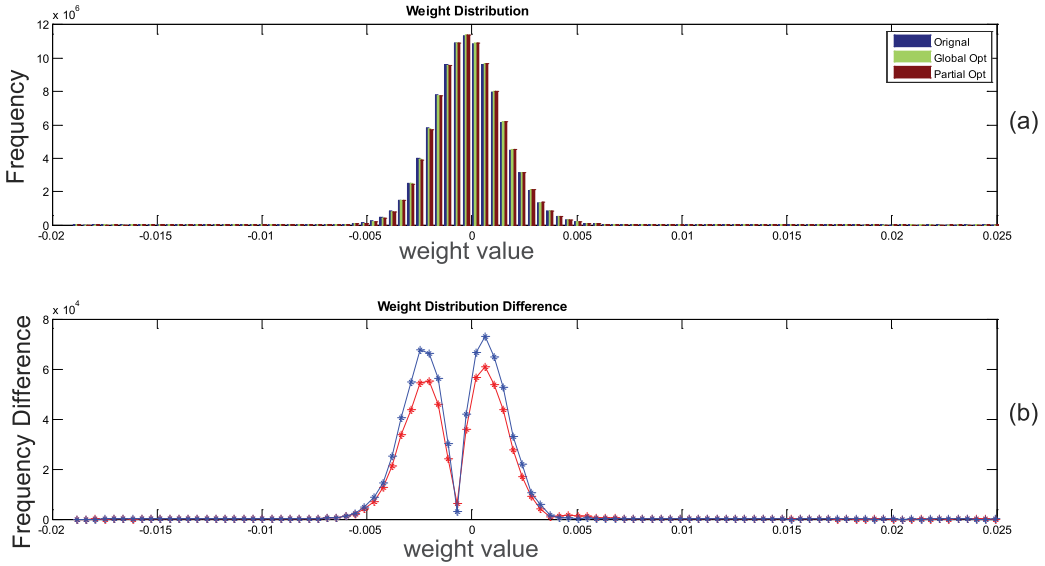

FIGURE 7. (a) Weight distribution after globa and partial fine-tuning. (b) Weight distribution difference. The red line is the weight distribution difference of the global fine-tuning, while the blue one is weight distribution difference of the partial fine-tuning.

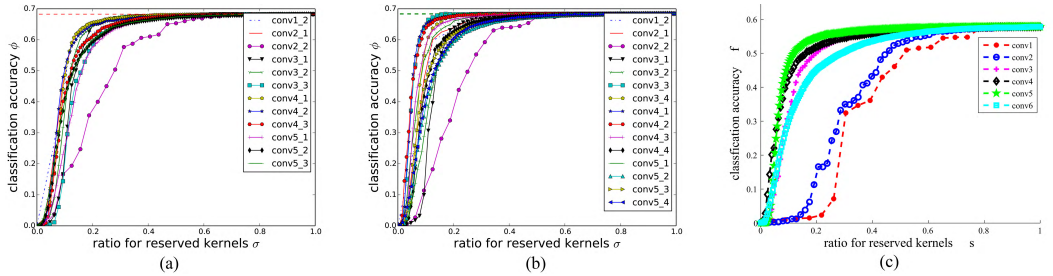

FIGURE 8. From the left to the right are sensitivity curves for VGGNet-16, VGGNet-19, and OverFeat respectively. Due to large size of kernel in conv1 and conv2, they are more sensitive when certain ratio of kernels were removed.

TABLE 3. The compression and acceleration on AlexNet with different parameter reserved ratio (PR) and acceleration ratio (AR) configuration.

	[19]		[26]		this paper	
	PR	AR	PR	AR	PR	AR
conv1	0.84	*	1	1	1	1
conv2	0.38	*	0.24	1.5	0.55	1.81
conv3	0.35	*	0.25	1.42	0.58	1.71
conv4	0.37	*	0.24	1.49	0.56	1.80
conv5	0.37	*	0.22	1.56	0.53	1.89
FC6	0.09	2	1	1	0.18	5.55
FC7	0.09	5	1	1	0.25	4
FC8	0.25	1	1	1	1	1

TABLE 4. The compression and acceleration on VGGNet-16 with different parameter reserved ratio (PR) and acceleration ratio (AR) configuration.

	[19]		this paper	
	PR	AR	PR	AR
conv1_1	0.58	*	1	1
conv1_2	0.22	*	0.37	2.7
conv2_1	0.34	*	0.40	2.5
conv2_2	0.36	*	0.40	2.5
conv3_1	0.53	*	0.41	2.44
conv3_2	0.24	*	0.44	2.27
conv3_3	0.42	*	0.45	2.22
conv4_1	0.32	*	0.33	3.03
conv4_2	0.27	*	0.36	2.78
conv4_3	0.34	*	0.37	2.71
conv5_1	0.35	*	0.38	2.63
conv5_2	0.29	*	0.37	2.71
conv5_3	0.36	*	0.37	2.71
FC6	0.04	9	0.07	13.75
FC7	0.04	10	0.125	8
FC8	0.23	1	1	1

During the experiments, we observed that the performance by fine-tuning multiple layers in one layer-cluster is better that of fine-tuning each single layer. And as shown in Fig. 6, the conv5 layers has the worst restoration ability, while the conv1 and conv2 layers have the best restoration ability. Therefore, we perform layer decomposition fine-tuning progressively as follows: we first decompose FC6 layer and keep the left layers as same as that of the original model, reconstruct network model as discussed in Sec. III, and perform fine-tuning on the new reconstructed model; we then repeat the same procedure on the layer-clusters (e.g. conv5 and conv4) successively until we reach the top layer-cluster conv1; we finally perform decomposition and reconstruction on FC7 layer. The top-1 accuracy and corresponding loss obtained after fine-tuning each FC layer and layer-cluster are shown in Fig. 9. As indicated in Fig. 9, we can achieve about 3× time speed-up without obvious accuracy decreasing or loss increasing. Furthermore, the proposed method

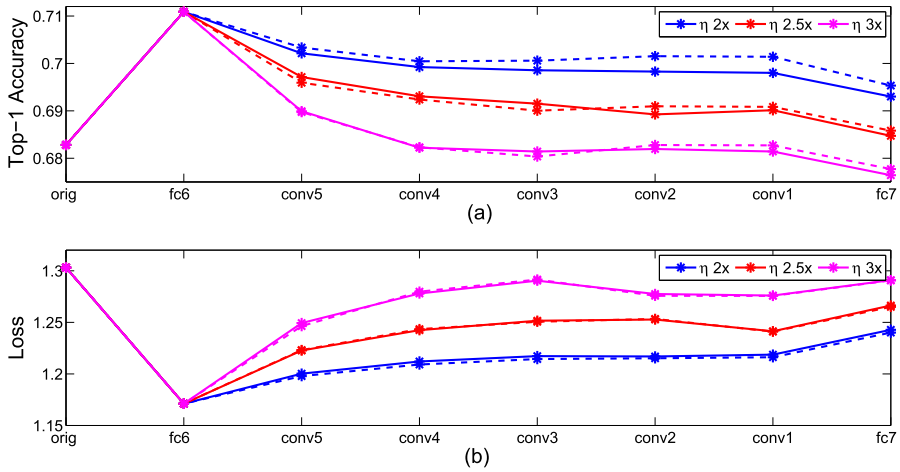

FIGURE 9. The top-1 accuracy and corresponding loss after fine-tuning each FC layer and layer-cluster. From the left to right are results obtained from the original model (orig), and after fine-tuning FC6 (fc6), conv5, conv4, conv3, conv2, conv1, and FC7 (fc7) respectively. The dash line indicates the accuracy and loss curve of proposed method, while the solid line indicates the results obtained by roughly selecting principle components. (a) Accuracy curve. (b) Loss curve.

TABLE 5. Acceleration/compression configuration for OverFeat.

AR		conv1	conv2	conv3	conv4	conv5	conv6	top-1%	top-5%
2x	ζ	2.760	2.517	1.974	1.916	1.888	2.036	0.4	0.2
	p	61	148	232	225	443	478		

is able to improve the accuracy with 0.34%, 0.15%, and 0.20% according to 2×, 2.5× and 3× configuration. The experiment on VGGNet-19 model shows similar result as that of VGGNet-16 model.

C. COMPARISONS WITH OTHER WORKS

In this section, we compare the our proposed method with the methods proposed in [19] and [26]. The compression and acceleration results obtained on AlexNet model with different parameter reserved ratio (PR) and acceleration ratio (AR) are summarized in Table 3. As shown in Table 3, only FC layer can be accelerated by [19]. And as for FC6 layer, the compression ratio is 11, but the acceleration ratio is only 2. However, our proposed method can achieve 5× compression and acceleration ratios. The method proposed in [26] employ the global optimization on the whole network model rather than one single layer, and it is effective for small network models. While our proposed method is able to perform compress and accelerate more efficiently on large scale networks model, such as VGGNet and AlexNet. The compression and acceleration results obtained on VGGNet-16 by our proposed method and [19] are shown in Table 4. It can be seen that 2.7× compression and acceleration ratio can be achieved by our proposed method on convolutional layers for large network models.

Our proposed method is similar as that in [25]. However, our proposed method use different strategy to decide rank for each layer. In order to perform fair comparison with [25], we evaluate performance on OverFeat [32]network model, and only perform decomposition and reconstruction on convolutional layers. We compare the classification accuracy on the basis of top-5 error. According to sensitivity curve as shown in Fig. 8 (c), reserved rank for each convolutional layer is summarized as Table 5 for each layer. After decomposition and fine-tuning with 2× acceleration and compression configuration, our proposed method increase top-5 error only with 0.2%, while [25] increase top-5 error with about 0.9%.

V. CONCLUSION

In this paper, we proposed a sensitivity-oriented layer-wise acceleration and compression method for CNN, which is able to facilitate network model deployment on portable devices. On the one hand, we analyze the sensitivity of the rank in each layer to the network accuracy; on the other hand, we remove the kernels with lower sensitivity than that with higher sensitivity. Furthermore, we show that our global optimization method can achieve better performance than layer-wise optimization. Experimental results demonstrate that, the proposed method is able to reduce 91% model size and speed up 2.7 times inference computation with 0.05% accuracy loss for VGGNet-16/19.

REFERENCES

[1] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intell. Syst. Appl.*, vol. 13, no. 4, pp. 18–28, Jul. 1998.

[2] G. Hinton, A. Krizhevsky, and I. Sutskever, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1–9.

[3] S. Karen and A. Zisserman. (2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: https://arxiv.org/abs/1409.1556

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[5] S. Christian, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proc. 31st AAAI Conf. Artif. Intell.*, Feb. 2017, pp. 1–7.
[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.
[7] S. Q. Ren, K. M. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.
[8] W. Liu *et al.*, "Ssd: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.*, Sep. 2016, pp. 21–37.
[9] L. Jonathan, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 3431–3440.
[10] F. Clement, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915–1929, Aug. 2013.
[11] Q. Mao, M. Dong, Z. Huang, and Y. Zhan, "Learning salient features for speech emotion recognition using convolutional neural networks," *IEEE Trans. Multimedia*, vol. 16, no. 8, pp. 2203–2213, Dec. 2014.
[12] G. Zhang, J. Kato, Y. Wang, and K. Mase, "A novel approach for annotation-based image retrieval using deep architecture," *J. Multiple-Valued Logic Soft Comput.*, vol. 30, nos. 4–6, pp. 541–558, 2018.
[13] S. Zhang, S. Zhang, T. Huang, and W. Gao, "Speech emotion recognition using deep convolutional neural network and discriminant temporal pyramid matching," *IEEE Trans. Multimedia*, vol. 20, no. 6, pp. 1576–1590, Oct. 2017.
[14] T. Zhang, W. Zheng, Z. Cui, Y. Zong, J. Yan, and K. Yan, "A deep neural network-driven feature learning method for multi-view facial expression recognition," *IEEE Trans. Multimedia*, vol. 18, no. 12, pp. 2528–2536, Dec. 2016.
[15] X. Lu, Z. Lin, H. Jin, J. Yang, and J. Z. Wang, "Rating image aesthetics using deep learning," *IEEE Trans. Multimedia*, vol. 17, no. 11, pp. 2021–2034, Nov. 2015.
[16] K. Yoon. (2014). "Convolutional neural networks for sentence classification." [Online]. Available: https://arxiv.org/abs/1408.5882
[17] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, Feb. 2016, pp. 2094–2100.
[18] H. Babak and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Proc. Adv. Neural Inf. Process. Syst.*, 1993, pp. 164–171.
[19] S. Han and H. Mao, W. J. Dally. (2015). "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." [Online]. Available: https://arxiv.org/abs/1510.00149
[20] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. (2017). "Incremental network quantization: Towards lossless CNNs with low-precision weights." [Online]. Available: https://arxiv.org/abs/1702.03044
[21] C. Wenlin, J. Wilson, S. Tyree, K. Weinberger, and Y. X. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2015, pp. 2285–2294.
[22] I. Khan, "Non-rigid structure-from-motion with uniqueness constraint and low rank matrix fitting factorization," *IEEE Trans. Multimedia*, vol. 16, no. 5, pp. 1350–1357, Aug. 2014.
[23] H. Wang, Y. Cen, Z. He, R. Zhao, Y. Cen, and F. Zhang, "Robust generalized low-rank decomposition of multimatrices for image recovery," *IEEE Trans. Multimedia*, vol. 19, no. 5, pp. 969–983, May 2017.
[24] J. Max, V. Andrea, and Z. Andrew. (2014). "Speeding up convolutional neural networks with low rank expansions." [Online]. Available: https://arxiv.org/abs/1405.3866
[25] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1984–1992.
[26] L. Vadim, G. Yaroslav, R. Maksim, O. Ivan, and L. Victor. (2014). "Speeding-up convolutional neural networks using fine-tuned CP-decomposition." [Online]. Available: https://arxiv.org/abs/1412.6553
[27] P. Wang and J. Cheng, "Accelerating convolutional neural networks for mobile applications," in *Proc. ACM Multimedia Conf.*, Oct. 2016, pp. 541–545.
[28] J. T. Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2016, pp. 25–36.
[29] Z. Chen *et al.*, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2015, pp. 161–170.
[30] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *Proc. IEEE 31st Int. Conf. Comput. Des. (ICCD)*, Oct. 2013, pp. 13–19.
[31] M. Figurnov, A. Ibraimova, D. Vetrov, and P. Kohli, "PerforatedCNNs: Acceleration through elimination of redundant convolutions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 947–955.
[32] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. (2013). "OverFeat: Integrated recognition, localization and detection using convolutional networks." [Online]. Available: https://arxiv.org/abs/1312.6229
[33] M. Denil, B. Shakibi, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2148–2156.
[34] B. Alex, J. Deng, and F. F. Li. *Large Scale Visual Recognition Challenge 2010*. Accessed: Nov. 2017. [Online]. Available: www.imagenet.org/challenges
[35] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, Nov. 2014, pp. 675–678.
[36] R. Collobert, C. Farabet, and K. Kavukcuoglu. *Torch: A Scientific Computing Framework for Luajit*. Accessed: Nov. 2017. [Online]. Available: http://torch.ch
[37] S. Pierre, D. Eigen, and X. Zhang. *OverFeat: Object Recognizer, Feature Extractor*. Accessed: Nov. 2017. [Online]. Available: https://cilvr.nyu.edu/doku.php?id=software:overfeat:start

WEI ZHOU received the B.E., M.S., and Ph.D. degrees from Northwestern Polytechnical University, Xi'an, China, in 2001, 2004, and 2007, respectively, where he is currently a Professor. His research interests include video coding and associated VLSI architecture design.

YUE NIU is currently pursuing the M.S. degree with the School of Electronic and Information, Northwestern Polytechnical University, Xi'an, China. His research interests include image and video processing, machine learning, and embedded systems.

GUANWEN ZHANG received the B.S. and M.S. degrees in computer science from Northwestern Polytechnical University, in 2007 and 2010, respectively, and the Ph.D. degree in engineering from Nagoya University, in 2014. He was a Postdoctoral Researcher with the Graduate School of Information Science, Nagoya University, from 2014 to 2016. He is currently an Assistant Professor with the School of Electronics and Information, Northwestern Polytechnical University. His research interests include computer vision, pattern recognition, and machine learning.