A Unified Approximation Framework for Compressing and Accelerating Deep Neural Networks

Yuzhe Ma*, Ran Chen*, Wei Li*, Fanhua Shang†, Wenjian Yu‡, Minsik Cho§, Bei Yu*,
*CSE Department, Chinese University of Hong Kong,
†School of Artificial Intelligence, Xidian University,
‡BNRist, Dept. Computer Science & Tech., Tsinghua University, §IBM T. J. Watson

***Abstract*—Deep neural networks (DNNs) have achieved significant success in a variety of real world applications, i.e., image classification. However, tons of parameters in the networks restrict the efficiency of neural networks due to the large model size and the intensive computation. To address this issue, various approximation techniques have been investigated, which seek for a light weighted network with little performance degradation in exchange of smaller model size or faster inference. Both low-rankness and sparsity are appealing properties for the network approximation. In this paper we propose a unified framework to compress the convolutional neural networks (CNNs) by combining these two properties, while taking the nonlinear activation into consideration. Each layer in the network is approximated by the sum of a structured sparse component and a low-rank component, which is formulated as an optimization problem. Then, an extended version of alternating direction method of multipliers (ADMM) with guaranteed convergence is presented to solve the relaxed optimization problem. Experiments are carried out on *VGG-16*, *AlexNet* and *GoogLeNet* with large image classification datasets. The results outperform previous work in terms of accuracy degradation, compression rate and speedup ratio. The proposed method is able to remarkably compress the model (with up to $4.9\times$ reduction of parameters) at a cost of little loss or without loss on accuracy.**

I. Introduction

As neural networks become deeper and deeper, the representation ability of neural network keeps improving, leading to significant performance promotion in a variety of tasks. However, the model size and the computation cost of neural networks are also increasing due to the huge amount of weights learned, which results in low throughput in inference stage and restrains the deployment on resource-limited systems. For example, embedded devices may lack enough storage and computation power to execute the giant networks. Meanwhile, deep neural networks are demonstrated to be over-parameterized [1], which motivates researchers to explore efficient approaches to make the deep models compact.

Approximating the deep models involves removing the redundancy and seeking for simplified structures such that the network after approximation may retain the performance on original tasks. Low-rankness and sparse connection are the most commonly applied assumptions when approximating a model. The illustration is presented in Fig. 1. Sparse connection can be realized by pruning a pre-trained network, which is the most straightforward approach. A hard thresholding approach is proposed in [2], which achieves high sparsity by removing the weights with less importance. Structured sparse connections can be learned by imposing group sparse regularizations during the training [3], as shown in Fig. 1(c). It also favors computation acceleration. In additional to sparse connection, low-rankness is another desired structure for model approximation. The weight matrices in the neural networks of low-rank can be further decomposed into smaller matrices, so as to reduce the amount of parameters as well as the computation cost [4], [5], as shown in Fig. 1(b). Tensor decomposition is applied in [6], where the weight tensor in fully-connected (FC) layer is approximated by a series of smaller kernels. Similar to sparse connection networks, the low-rank filter of neural network can also be learned by imposing regularizations [7], [8]. An intuitive extension of these work is to consider both low-rank structure and sparse structure simultaneously. In [9], a layer in the pre-trained neural network is decomposed into a low-rank component and a sparse component by a greedy algorithm. The obtained networks are compressed but not accelerated due to the non-structured sparse weights.

Performing approximation or pruning to a pre-trained network may inevitably result in performance loss. In order to retain the accuracy of a pre-trained model, some approaches aim to minimize the reconstruction error of the feature maps in each layer through solving an optimization problem with specified constraints on the rank or sparsity of the filters. The reconstruction error is measured between the linear response in original network and approximated one [9], [10]. Since the non-linearity such as Rectified Linear Units (ReLU) [11] follows the linear filters in most neural networks, only the error of positive response is accumulated and the error of negative response is omitted, which makes the accuracy more dependent on the positive response reconstruction. In [5], a method for reconstructing non-linear response is proposed.

In this work, we propose a unified approximation framework for CNNs which approximates the convolutional layers with two components, including a structured sparse component and a low-rank component. In contrast to [9], our

Corresponding authors: Wenjian Yu (yu-wj@tsinghua.edu.cn) and Bei Yu (byu@cse.cuhk.edu.hk).

2375-0197/19/$31.00 ©2019 IEEE
DOI 10.1109/ICTAI.2019.00060

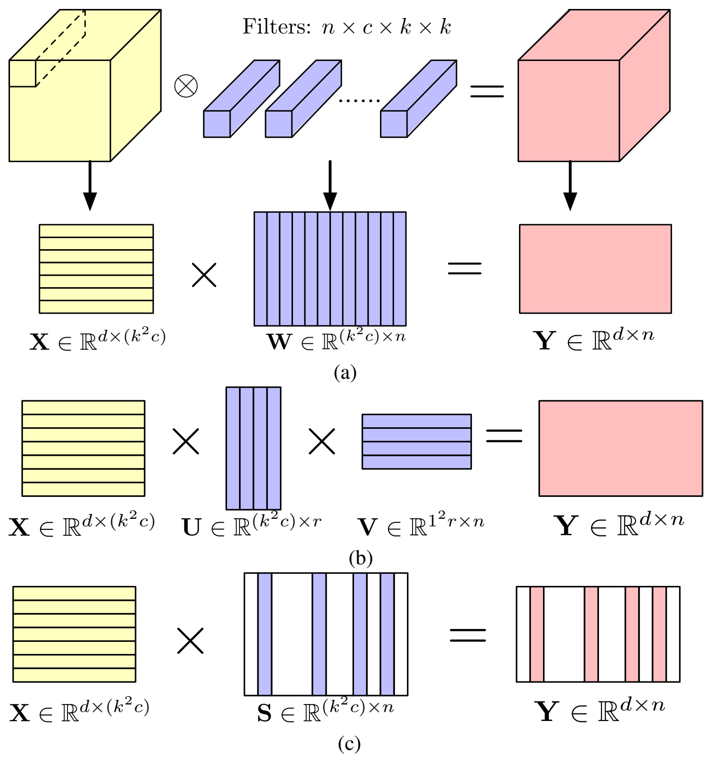
Filters: $n \times c \times k \times k$
$\mathbf{X} \in \mathbb{R}^{d\times(k^2c)}$
$\mathbf{W} \in \mathbb{R}^{(k^2c)\times n}$
$\mathbf{Y} \in \mathbb{R}^{d\times n}$
(a)
$\mathbf{X} \in \mathbb{R}^{d\times(k^2c)}$
$\mathbf{U} \in \mathbb{R}^{(k^2c)\times r}$
$\mathbf{V} \in \mathbb{R}^{1^2 r\times n}$
$\mathbf{Y} \in \mathbb{R}^{d\times n}$
(b)
$\mathbf{X} \in \mathbb{R}^{d\times(k^2c)}$
$\mathbf{S} \in \mathbb{R}^{(k^2c)\times n}$
$\mathbf{Y} \in \mathbb{R}^{d\times n}$
(c)

Fig. 1: (a) Transform convolution to matrix multiplication; (b) Approximate weight matrix $\mathbf{W}$ using two matrices with lower-rank; (c) Impose structured sparsity on weight matrix $\mathbf{W}$.

constraints not only facilitate model compression but also favors acceleration because of the structured sparse weights [3]. We retain the accuracy of the model by approximating the nonlinear response after activation. The layer-wise network approximation problem is formulated as minimizing the reconstruction error of the response after non-linear ReLU. To overcome the resulted difficulty of non-convex optimization, we propose a convex relaxation scheme which considers the constraints for structured sparsity and low-rankness, and then solve it with an extension of alternating direction method of multipliers (ADMM) [12]. Moreover, we prove that the extended ADMM algorithm converges to the optimal solution of the relaxed problem.

The proposed method is evaluated on well-known DNN architectures, including *VGG-16* [13], *NIN* [14], *AlexNet* [15] and *GoogLeNet* [16]. For *VGG-16* with the CIFAR-10 dataset, we achieve $4.4\times$ model compression with only 0.4% accuracy drop. Meanwhile, with the compressed model the inference is accelerated by $2.2\times$. For *AlexNet* with the ImageNet dataset, we achieve $4.9\times$ model compression at the cost that the top-5 accuracy drops slightly from 81.3% to 80%. For *GoogLeNet* with the ImageNet dataset, the proposed method also brings $2.9\times$ reduction of the model parameters without any degradation on the accuracy of inference. These experimental results reveal that the proposed approximation framework is able to remarkably compress the CNN models while keeping high accuracy.

The rest of this paper is organized as follows. In Section II, related literature on DNN compression and acceleration is summarized. The problem formulation of the proposed methodology is given in Section III. Section IV presents a numerical optimization algorithm for solving the problem. The experimental results are reported in Section V, and Section VI concludes the paper.

II. Related Work

Neural Network Sparsification. Despite the appealing performance of the deep neural network, it has been demonstrated that there is much redundancy which leads to computation overhead and large model size. Therefore, sparsifying some over-parameterized layers in neural network is a straightforward method to eliminate the redundancy while preserving the performance. The majority of the parameters in a sparse layer are zeros, thus the parameters can be stored with compressed representation, e.g., compressed sparse row (CSR) format, for size reduction. A three-stage pipeline is proposed in [2]. The parameters that are smaller than a threshold are considered as less important and are set to zeros. Then retraining is performed on the sparse structure to restore the accuracy. However, the sparse pattern is non-structured which has limited benefit for speedup during inference due to the poor weight locality. [17] proposes to prune the entire convolution kernel rather than single element based on the intensity. A structured sparse learning algorithm is proposed in [3], which enables to learn a network with structured sparse network by applying group sparse regularizations during training. Since structured sparsity leads to zero-columns and zero-rows in the lowered matrices, [3] further proposes to reduce the dimension of lowered matrices by removing these zero-columns and zero-rows, which reduces the dimension of the lowered weight matrix when applying General Matrix-Matrix Multiplication (GEMM) function and accelerates inference. A channel pruning method is proposed in [10], which can be considered as a special case of structured sparsity. The difference is that channel pruning is performed on a pre-trained model rather than training the model from scratch. [10] formulated the problem as l_0-norm minimization problem, trying to find the "informative" channels of the feature map and the corresponding weights. Instead of trying to minimize the reconstruction error layer by layer, [18] targets at a unified goal which is to minimize the reconstruction error of important response in the final response layer. In this paper, we are more interested in exploring structured sparsity since it not only facilitates compressing the networks but also acceleration.

Low-Rank Approximation. In addition to sparsifying a network, low-rank approximation is another sort of approach which can be applied for both network compression and acceleration. In modern convolutional neural networks (CNNs) structure, filters are usually a 4-D tensor. Some tensor decomposition techniques are leveraged for acceleration and compression. A straightforward idea is to replace the 4-D tensor with two consecutive tensors with lower-rank [19]. In addition, other kinds of tensor decomposition can also be applied. In [6], fully-connected layers are converted to the Tensor Train format, resulting in compression by a huge factor. CP-decomposition of the filter tensors is proposed in [20]. A relevant approach to low-rank approximation is tensor sketching [21]. The difference is that low-rank approximation will increase the network depth since an original layer will be decomposed into multiple layers. In order to conduct low-rank approximation more efficiently, methods for training neural networks with low-rank filters are investigated [4], [7], [8], [22].

Most of those low-rank approximation-based methods for a pre-trained model like [19], [23] consider reconstructing the response of linear block of a network, while ignoring the following non-linear activation function like ReLU [11] which is widely applied in DNNs. A method for low-rank approximation of non-linear response in convolutional networks is proposed in [5], which is demonstrated to have more speedup than approximating linear-response only.

Although both low-rank approximation and network sparsification are appealing, there are not so much work that consider how to combine them. Some previous work aims to train a network with both group-sparse and low-rank regularizations [8], [24], which impose the constraints of low-rank and sparse on the filters at the same time. Considering that filters tend to be both low-rank and sparse, a layer in a pre-trained DNN is approximated by the sum of a non-structured sparse component and a low-rank component for compression in [9]. Similar to [19], it relies on reconstructing the linear response of a layer to constrain accuracy loss. A potential problem is that non-structured sparsity may not favor computation acceleration well.

Although there are existing work combining the low-rank approximation and network sparsification and the work approximating the non-linear response of network, no one has combined all the ideas together. In this work, we propose to approximate the layers in CNN with the sum of a structured sparse component and a low-rank component, and minimize the reconstruction error, while taking the non-linear activation into account. It results in a unified approximation framework for compressing and accelerating DNN models.

III. Problem Formulation

In this section, we introduce our mathematical formulation for network approximation using structured sparse and low-rank decomposition, while taking non-linearity into account. To this end, we propose to formulate the problem into a unified optimization model. In the following context, we focus on CNNs which involve a large model size.

In an FC layer of a CNN, the output feature map can be computed as

$$\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{X}, \tag{1}$$

where $\boldsymbol{X} \in \mathbb{R}^m$ and $\boldsymbol{Y} \in \mathbb{R}^n$ represent the input feature vector and output response, respectively. $\boldsymbol{W} \in \mathbb{R}^{n\times m}$ denotes the weight matrix. For a convolutional layer the convolution operation can also be represented as Equation (1). The illustration is shown in Fig. 1(a). The convolution filter is $\mathcal{W} \in \mathbb{R}^{n\times c\times k\times k}$, where k is spatial size, c is the number of input channels and n is the number of filters. The filter can be reshaped to a matrix with size n-by-k^2c. The input $\boldsymbol{X}$ is lowered to a matrix such that each k^2c volume involved in a convolution forms a column. Then the convolution operation is converted to matrix multiplication.

The information loss is inevitable when we approximate the original filters by low-rank or sparse filters, which may cause performance degradation. In order to compress the network and accelerate the computation, we perform low-rank approximation and network sparsification simultaneously. The output feature maps of a layer is generated by the sum of convolving with each filter. In order to preserve the performance, we aim at minimizing the reconstruction error of the response generated by the approximated filters in each layer after activation. An example block structure in the network is demonstrated in Fig. 2. Then, the problem is formulated as follows:

$$\begin{aligned} \min_{\boldsymbol{A},\boldsymbol{B}} \; & \sum_{i=1}^{N} \left\| \boldsymbol{Y}_i - r((\boldsymbol{A}+\boldsymbol{B})\boldsymbol{X}_i) \right\|_F^2, \\ \text{s.t.} \; & \left\|\boldsymbol{A}\right\|_0 \le S, \quad \operatorname{rank}(\boldsymbol{B}) \le L. \end{aligned} \tag{2}$$

Here $\boldsymbol{Y}_i$ and $\boldsymbol{X}_i$ represent the output feature map and the input feature map of a layer, respectively. Structured sparse component $\boldsymbol{A}$ and low-rank component $\boldsymbol{B}$ are two weight matrices we are looking for, each of which is the lowered matrices of a 4-D tensor. N is the total number of samples used for approximation. $\left\|\cdot\right\|_F$ is Frobenius norm. $r(\cdot)$ is the activation function in the network, i.e., ReLU$(\cdot)$. S and L are user-defined target sparsity level and target rank for the filters.

IV. Optimization Methodology

A. Problem Relaxation

Solving Problem (2) directly involves both l_0 minimization and rank minimization, which is NP-hard. Besides, we want $\boldsymbol{A}$ to be structured sparse, which leads to extra difficulty. To tackle this challenge, we apply convex relaxation to the constraints. The rank constraint on $\boldsymbol{B}$ is relaxed by nuclear norm of $\boldsymbol{B}$, which is the sum of the singular values of $\boldsymbol{B}$. As for the l_0 norm constraints, a general way is to relax it by l_1 norm which is convex and has good performance in imposing sparsity. However, as we discussed above, structured sparse patterns can be more easily used for computation acceleration. Therefore, here we relax l_0 constraint by $l_{2,1}$ norm (the sum of the Euclidean norms of the columns) such that the zero elements in $\boldsymbol{A}$ appear column-wise. Then, the original problem is reformulated as

$$\min_{\boldsymbol{A},\boldsymbol{B}} \sum_{i=1}^{N} \left\| \boldsymbol{Y}_i - r((\boldsymbol{A}+\boldsymbol{B})\boldsymbol{X}_i) \right\|_F^2 + \lambda_1 \left\|\boldsymbol{A}\right\|_{2,1} + \lambda_2 \left\|\boldsymbol{B}\right\|_*, \tag{3}$$

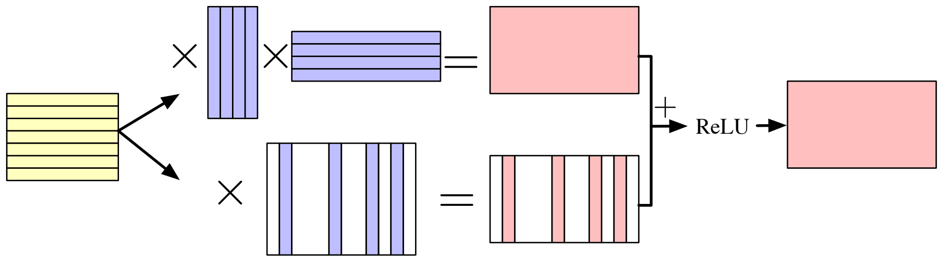

Fig. 2: Structure combining sparse and low-rank decomposition.

where $\|\cdot\|_{2,1}$ is $l_{2,1}$ norm and $\|\cdot\|_*$ is nuclear norm. λ_1 and λ_2 are coefficients of the relaxed terms. The problem (3) now is a convex optimization problem. To solve it, we make use of the alternating direction method of multipliers (ADMM), which is widely used in large-scale problems arising in statistics [12]. Especially, the optimal solution of the sub-problems involving $l_{2,1}$-norm and nuclear norm can be obtained in closed-form as in subspace learning [25] and the singular value thresholding (SVT) operator [26], respectively.

By introducing an auxiliary variable $\boldsymbol{M}$, the Problem (3) can be rewritten as

$$\begin{aligned}\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{M}} &\sum_{i=1}^{N} \|\boldsymbol{Y}_i - r(\boldsymbol{M}\boldsymbol{X}_i)\|_F^2 + \lambda_1 \|\boldsymbol{A}\|_{2,1} + \lambda_2 \|\boldsymbol{B}\|_*, \\ \text{s.t.} \quad &\boldsymbol{A}+\boldsymbol{B} = \boldsymbol{M}.\end{aligned} \tag{4}$$

Then the augmented Lagrangian function of Problem (4) is

$$\begin{aligned}L_t(\boldsymbol{A},\boldsymbol{B},\boldsymbol{M},\boldsymbol{\Lambda}) &= \sum_{i=1}^{N} \|\boldsymbol{Y}_i - r(\boldsymbol{M}\boldsymbol{X}_i)\|_F^2 + \lambda_1 \|\boldsymbol{A}\|_{2,1} \\ &+ \lambda_2 \|\boldsymbol{B}\|_* + \langle \boldsymbol{\Lambda}, \boldsymbol{A}+\boldsymbol{B}-\boldsymbol{M}\rangle + \frac{t}{2}\|\boldsymbol{A}+\boldsymbol{B}-\boldsymbol{M}\|_F^2,\end{aligned} \tag{5}$$

where $t > 0$ is the penalty parameter and $\boldsymbol{\Lambda}$ is Lagrange multiplier. $\langle\cdot,\cdot\rangle$ represents the inner product operator.

B. Variables Update

ADMM solves the minimization problem of $L_t(\boldsymbol{A},\boldsymbol{B},\boldsymbol{M},\boldsymbol{\Lambda})$ iteratively. The variables are alternatively updated in each iteration. To update $\boldsymbol{A},\boldsymbol{B},\boldsymbol{M}$ in iteration $k+1$, our algorithm takes two steps. Firstly, we consider the following three sub-problems.

$$\min_{\boldsymbol{A}} \lambda_1 \|\boldsymbol{A}\|_{2,1} + \frac{t}{2}\left\|\boldsymbol{A}+\boldsymbol{B}_k-\boldsymbol{M}_k+\frac{\boldsymbol{\Lambda}_k}{t}\right\|_F^2, \tag{6}$$

$$\min_{\boldsymbol{B}} \lambda_2 \|\boldsymbol{B}\|_* + \frac{t}{2}\left\|\boldsymbol{B}+\hat{\boldsymbol{A}}_k-\boldsymbol{M}_k+\frac{\boldsymbol{\Lambda}_k}{t}\right\|_F^2, \tag{7}$$

$$\begin{aligned}\min_{\boldsymbol{M}} &\sum_{i=1}^{N} \|\boldsymbol{Y}_i - r(\boldsymbol{M}\boldsymbol{X}_i)\|_F^2 + \langle \boldsymbol{\Lambda}_k, \hat{\boldsymbol{A}}_k+\hat{\boldsymbol{B}}_k-\boldsymbol{M}\rangle \\ &+\frac{t}{2}\left\|\hat{\boldsymbol{A}}_k+\hat{\boldsymbol{B}}_k-\boldsymbol{M}\right\|_F^2.\end{aligned} \tag{8}$$

All these three problems are proximal mapping problems. For Problem (6), the optimal solution is given by

$$\hat{\boldsymbol{A}}_k = \text{prox}_{\frac{\lambda_1}{t}\|\cdot\|_{2,1}}(\boldsymbol{M}_k - \boldsymbol{B}_k - \frac{\boldsymbol{\Lambda}_k}{t}). \tag{9}$$

The explicit representation of Equation (9) can be derived based on [25]. Let $\boldsymbol{C} = \boldsymbol{M}_k - \boldsymbol{B}_k - \frac{\boldsymbol{\Lambda}_k}{t}$, then the column i in $\hat{\boldsymbol{A}}_k$ is given as

$$[\hat{\boldsymbol{A}}_k]_{:,i} = \begin{cases} \dfrac{\|[\boldsymbol{C}]_{:,i}\|_2 - \frac{\lambda_1}{t}}{\|[\boldsymbol{C}]_{:,i}\|_2}[\boldsymbol{C}]_{:,i}, & \text{if } \|[\boldsymbol{C}]_{:,i}\|_2 > \dfrac{\lambda_1}{t}; \\ 0, & \text{otherwise.}\end{cases} \tag{10}$$

For Problem (7), the optimal solution is given by

$$\hat{\boldsymbol{B}}_k = \text{prox}_{\frac{\lambda_2}{t}\|\cdot\|_*}(\boldsymbol{M}_k - \hat{\boldsymbol{A}}_k - \frac{\boldsymbol{\Lambda}_k}{t}). \tag{11}$$

The explicit representation of Equation (11) can be obtained based on SVT operator $\mathcal{D}_\tau$ [26]. Let $\boldsymbol{D} = \boldsymbol{M}_k - \hat{\boldsymbol{A}}_k - \frac{\boldsymbol{\Lambda}_k}{t}$. We perform singular value decomposition on $\boldsymbol{D}$ such that $\boldsymbol{D} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}$, where $\boldsymbol{\Sigma} = \text{diag}(\{\sigma_i\}_{1\le i\le r})$ and σ_i is the i-th largest singular value. Then $\hat{\boldsymbol{B}}_k$ is given by

$$\hat{\boldsymbol{B}}_k = \boldsymbol{U}\mathcal{D}_{\frac{\lambda_2}{t}}(\boldsymbol{\Sigma})\boldsymbol{V} \tag{12}$$

where $\mathcal{D}_{\frac{\lambda_2}{t}}(\boldsymbol{\Sigma}) = \text{diag}(\{(\sigma_i - \frac{\lambda_2}{t})_+\})$.

For Problem (8), it is non-trivial to derive the closed-form of the optimal solution of the sub-problem with respect to $\boldsymbol{M}$ since $r(\cdot)$ is a piecewise linear function. However, the function is continuous and convex so that we can approach the optimal solution of $\boldsymbol{M}$ iteratively by applying gradient-based method. In our implementation, we apply stochastic gradient descent (SGD) to solve it, and set learning rate as 10^{-3} and momentum as 0.9.

Up to now we are extending the classical ADMM to a three-block separable convex programming. This direct extension, however, is not necessarily convergent, as shown in the previous works [27], [28]. To address this issue, a simple correction step was proposed in [27], shown as follows.

$$\begin{pmatrix}\boldsymbol{B}_{k+1}\\ \boldsymbol{M}_{k+1}\\ \boldsymbol{\Lambda}_{k+1}\end{pmatrix} = \begin{pmatrix}\boldsymbol{B}_{k}\\ \boldsymbol{M}_{k}\\ \boldsymbol{\Lambda}_{k}\end{pmatrix} - \alpha\begin{pmatrix}\boldsymbol{I} & (\tau-1)\boldsymbol{I} & \boldsymbol{O}\\ \tau\boldsymbol{I} & \boldsymbol{I} & \boldsymbol{O}\\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{I}\end{pmatrix}\begin{pmatrix}\boldsymbol{B}_k - \hat{\boldsymbol{B}}_k\\ \boldsymbol{M}_k - \hat{\boldsymbol{M}}_k\\ \boldsymbol{\Lambda}_k - \hat{\boldsymbol{\Lambda}}_k\end{pmatrix}, \tag{13}$$

where $\boldsymbol{O}$ denotes zero matrix. τ is set to $\frac{1}{2}$. α is set to $\frac{3}{4}$. With this correction step, the extended ADMM can ensure the global convergence.

The overall optimization procedure is summarized in Algorithm 1. It starts with an initialization for all the variables and hyper-parameters (line 1). Then these variables are updated alternatively in each iteration based on the equations or SGD algorithm, as described above (line 3 – line 6). Each iteration ends up with a correction step presented as Equation (13) (line 7). The entire optimization procedure exits when the pre-defined condition is satisfied.

Algorithm 1 ADMM for solving Problem (4)

```
Input: Feature maps $\boldsymbol{Y}_i, \boldsymbol{X}_i$, $i = 1 \cdots N$, given $\lambda_1$, $\lambda_2$.
Output: Structured sparse matrix $\boldsymbol{A}$ & low-rank matrix $\boldsymbol{B}$.
 1: Initialize $k \leftarrow 0$, $\boldsymbol{\Lambda}_0$, $\boldsymbol{A}_0$, $\boldsymbol{B}_0$, $\boldsymbol{M}_0$, error tolerance $\epsilon$, $t$;
 2: while not converged do
 3:     Calculate $\hat{\boldsymbol{A}}_k$ by Equation (10);
 4:     Calculate $\hat{\boldsymbol{B}}_k$ by Section IV-B;
 5:     Calculate $\hat{\boldsymbol{M}}_k$ by solving Problem (8) with SGD method;
 6:     $\hat{\boldsymbol{\Lambda}}_k \leftarrow \boldsymbol{\Lambda}_k + t(\hat{\boldsymbol{A}}_k + \hat{\boldsymbol{B}}_k - \hat{\boldsymbol{M}}_k)$;
 7:     Perform correction step by Equation (13);
 8:     $k \leftarrow k + 1$;
 9: end while
10: return $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$;
```

C. Convergence Analysis

In this subsection, we prove the convergence of Algorithm 1. Let $f_1(\boldsymbol{M}) = \sum_{i=1}^{N} \|\boldsymbol{Y}_i - r(\boldsymbol{M}\boldsymbol{X}_i)\|_F^2$, $f_2(\boldsymbol{A}) = \lambda_1 \|\boldsymbol{A}\|_{2,1}$, and $f_3(\boldsymbol{B}) = \lambda_2 \|\boldsymbol{B}\|_*$. Let $\mathbf{m}$ denote the vectorization of $\boldsymbol{M}$, i.e., $\mathbf{m} = \text{vec}(\boldsymbol{M})$, and similarly, let $\mathbf{a} = \text{vec}(\boldsymbol{A})$, and $\mathbf{b} = \text{vec}(\boldsymbol{B})$.

Using these notations, the problem in Equation (4) takes the following generic form

$$\begin{aligned} \min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{M}} \ & f_1(\boldsymbol{M}) + f_2(\boldsymbol{A}) + f_3(\boldsymbol{B}), \\ \text{s.t.} \ & \boldsymbol{C}_1\mathbf{a} + \boldsymbol{C}_2\mathbf{b} - \boldsymbol{C}_3\mathbf{m} = \mathbf{c}, \end{aligned} \tag{14}$$

where $\boldsymbol{C}_1$, $\boldsymbol{C}_2$, and $\boldsymbol{C}_3$ are the identity matrices, and $\boldsymbol{c} = \mathbf{0}$. The convergence of ADMM for solving the standard form (14) was studied in [27], [28]. We establish the convergence of our algorithm by transforming the problem in Equation (4) into a standard form (14). Note that our algorithm alternates between three blocks of variables, $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{M}$. According to the definitions of $f_1(\boldsymbol{M})$, $f_2(\boldsymbol{A})$, and $f_3(\boldsymbol{B})$, it is easy to verify the problem in Equation (4) and our algorithm satisfy the convergence conditions of the problem in Equation (14), as stated in [27]. Thus, we have the following theorem.

Theorem 1. *Consider the problem in Equation* (4)*, where* $f_1(\boldsymbol{M})$*,* $f_2(\boldsymbol{A})$*, and* $f_3(\boldsymbol{B})$*, are convex functions, and* $\boldsymbol{C}_1$*,* $\boldsymbol{C}_2$*, and* $\boldsymbol{C}_3$ *are the identity matrices, and have full column rank. The sequence* $\{\boldsymbol{A}_k, \boldsymbol{B}_k, \boldsymbol{M}_k\}$ *generated by Algorithm 1 converges to the optimal solution* $\{\boldsymbol{A}^*, \boldsymbol{B}^*, \boldsymbol{M}^*\}$ *of the problem in Equation* (4).

TABLE I: Results on *VGG-16* with CIFAR-10

Layer	$CR(\boldsymbol{A})(\%)$	$CR(\boldsymbol{B})(\%)$	$CR(\boldsymbol{A}+\boldsymbol{B})(\%)$
conv1-1	0.0	100.0	100
conv1-2	5.4	52.1	57.5
conv2-1	5.4	38.2	43.6
conv2-2	2.2	28.6	30.8
conv3-1	2.8	47.7	50.5
conv3-2	4.0	54.3	58.3
conv3-3	10.0	59.0	69.0
conv4-1	2.0	16.0	18.0
conv4-2	2.0	22.4	24.4
conv4-3	4.0	16.3	24.3
conv5-1	2.0	9.8	11.8
conv5-2	2.0	8.7	10.7
conv5-3	2.0	6.7	8.7
fc1	44.2	0.0	44.2
fc2	36.2	0.0	36.2
fc3	24.0	0.0	24.0
CR	22.5%	(4.44× reduction of model size)	
Speed-up		2.2×	
Accu. ↓		0.40%	

V. Experimental Results

A. Experimental Setup

Algorithm 1 takes in the input and output feature maps generated from the inference on some sample data, and outputs the approximated network layers. The tested CNNs include *VGG-16* [13], *NIN* [14], *AlexNet* [15] and *GoogLeNet* [16]. For each network, we first obtain its approximation, and then fine-tune the network based on the obtained structures to restore the accuracy. During the approximation, different layers use different weight coefficients λ_1 and λ_2. In our experiments, we find out setting λ_2 to be $2.5 \sim 3$ times larger than λ_1 gives good trade-off between accuracy and model compression rate. And we let λ_1 ranges from $0.08 \sim 0.3$. The penalty parameter t in (5) is set to 10^{-3}. The runtime of Algorithm 1 varies from layer to layer, ranging from 10 minutes to half an hour.

The inference is conducted on Caffe [29] using CIFAR-10 and ILSVRC-2012, i.e., ImageNet [30]. After the network approximation, a small initial learning rate of 10^{-5} is used in the fine-tuning step. We use three metrics for evaluation, including accuracy loss, compression rate (CR) and speedup ratio. The CR is calculated as

$$\text{CR} = \frac{\text{Approximated layer size}}{\text{Original layer size}} \times 100\%. \tag{15}$$

The accuracy loss is the degradation on accuracy after approximation, denoted by "*accu.* ↓" in the table. The "speed-up" ratio indicates the acceleration for inference.

B. Experiments on CIFAR-10

1) VGG-16: VGG-16 [13] network is a convolutional neural network consisting of 13 convolution layers and 3 FC layers. All the convolutional filters have the same spatial size of 3×3. We test the proposed method with experiments on the CIFAR-10 dataset which consists of 50K training images

TABLE II: Results on *NIN* with CIFAR-10

Layer	$CR(\boldsymbol{A})(\%)$	$CR(\boldsymbol{B})(\%)$	$CR(\boldsymbol{A}+\boldsymbol{B})(\%)$
conv1	0.0	18.4	18.4
cccp1	0.0	100.0	100
cccp2	0.0	100.0	100
conv2	0.0	16.9	16.9
cccp3	0.0	100.0	100
cccp4	0.0	100.0	100
conv3	0.0	38.2	38.2
cccp5	0.0	100.0	100
cccp6	0.0	100.0	100
CR	36.0%	(2.77× reduction of model size)	
Speed-up		2.2×	
Accu. ↓		0.41%	

TABLE III: Comparison on CIFAR-10

Model	Method	Accu. ↓	CR	Speed-up
VGG-16	Original	0.00%	1.00	1.00
	ICLR'17 [17]	**0.06%**	2.70	1.80
	Ours	0.40%	**4.44**	**2.20**
NIN	Original	0.00%	1.00	1.00
	ICLR'16 [7]	1.43%	1.54	1.50
	IJCAI'18 [21]	1.43%	1.45	-
	Ours	**0.41%**	**2.77**	**1.70**

and 10K test images. We first train a *VGG-16* network from scratch to obtain the baseline, which has an accuracy of 92.05%. To make the approximation, 1000 images are selected from training set for inference and the input and output feature maps are collected for Algorithm 1.

The approximation is performed on each layer sequentially. The layer-wise approximation results are shown in TABLE I. In our experiment, we find out approximating the first convolutional layer may lead to significant accuracy drop. Therefore, the first layer is not approximated. The sparse component $\boldsymbol{A}$ is stored in CSR format. Moreover, we constrain the sparse component $\boldsymbol{A}$ to be structured sparse to accelerate the computation as in [3]. The low-rank component is represented by the product of two smaller matrices. For FC layers, we only use the sparse component for approximation to reduce accuracy drop.

The performance comparison with other previous work [17] is presented in TABLE III. With the approximation, the model size is reduced by $4.44\times$, which corresponds to $2.2\times$ speedup on inference. Both compression rate and speedup ratio outperform [17]. Without fine-tuning, there is some classification accuracy drop. In order to restore the accuracy of the compressed model, we retrain the compressed network with the training set for 5 epochs. With this fine-tuning step the accuracy loss reduces from 1.8% to only 0.40%, which becomes very close to the accuracy of the original VGG-16.

If a shallow layer is approximated, the approximation error may be accumulated when deeper layers are approximated. In order to handle this issue, we take the 'asymmetric' strategy used in [5]. We approximate the layers from shallow to deep. When approximating a deep layer, use the response produced by all previous layers instead of the non-approximate response as the input feature map $\boldsymbol{X}_i$. Fig. 3 shows the comparison of classification error increase. We can observe that with more layers being approximated, the performance becomes worse for both strategies. However, the asymmetric version loses less accuracy.

We further compare the performance between reconstructing non-linear response and reconstructing linear response. We perform the comparison on a single layer each time, while the remaining layers are kept unchanged. In Fig. 4, we plot the relation between the CR and the accuracy degradation

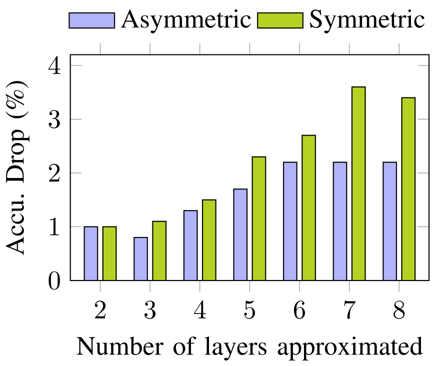

Fig. 3: Accuracy on CIFAR-10 using symmetric reconstruction and asymmetric reconstruction.

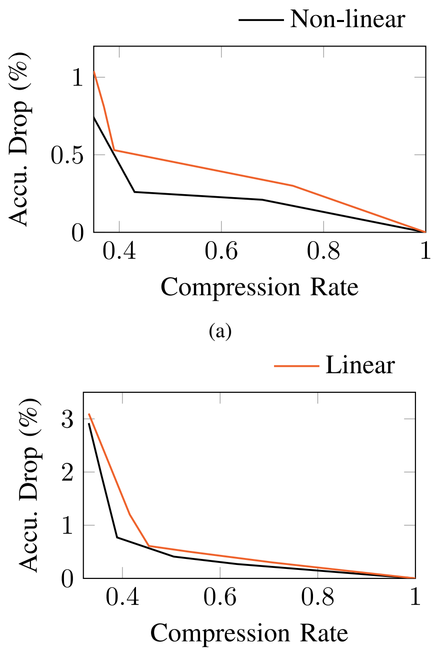

Fig. 4: Comparison of reconstructing linear response and non-linear response: (a) layer conv2-1; (b) layer conv3-1.

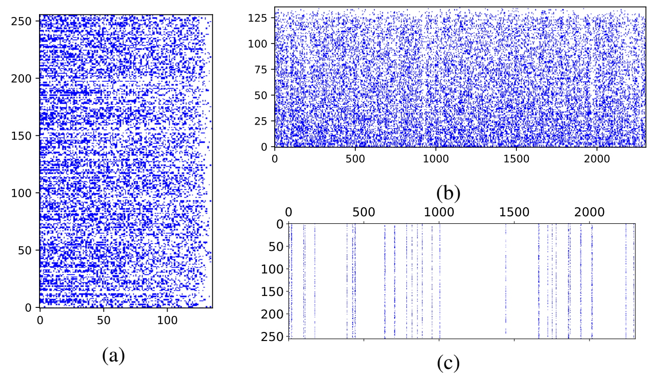

Fig. 5: Approximated filters of `conv3-1`. Blue dots have non-zero values. Low-rank filter $\boldsymbol{B}$ with rank 136 is decomposed into $\boldsymbol{UV}$, both of which have rank 136. (a) Matrix $\boldsymbol{U}$; (b) Matrix $\boldsymbol{V}$. (c) Column-wise sparse filter $\boldsymbol{A}$.

of two approaches of different layers. The performance is evaluated by the accuracy drop compared with original model. We take two convolutional layers in two different stages of the *VGG-16*, including `conv2-1` and `conv3-1`. Fig. 4 shows that under the same CR, reconstructing non-linear response achieves lower accuracy drop than reconstructing linear response, which verifies the advantage of reconstructing the non-linear response. In Fig. 5, we visualize the sparse filter and low-rank filter after the approximation of layer `conv3-1`. $\boldsymbol{B}$ has rank 136 and it can be further decomposed by $\boldsymbol{B} = \boldsymbol{UV}$, where both $\boldsymbol{U}$ and $\boldsymbol{V}$ have rank 136.

2) NIN: Network-in-network (*NIN*) [14] has 9 convolutional layers among which 6 layers have a spatial size of 1×1. Considering that these 1×1 convolutional layers have less contribution to the overall model size and computation, we focus on remaining three layers which have spatial size of 3×3 or 5×5. We present the layer-wise approximation results in TABLE II. It can be observed that for all the approximated convolutional layers, only low-rank component is used and structured sparse didn't show up, which means approximating *NIN* using CIFAR10 dataset reduces to low-rank approximation and sparse components are not beneficial to the objectives. It indicates that the proposed unified framework is flexible to find good solutions and does not rely on prior assumptions to achieve good results.

Experimental results using the same network (i.e., *NIN*) and CIFAR10 are reported in previous work [7], [21]. The comparison of accuracy loss, compression rates and the accuracy is shown in TABLE III. We can see that the number of parameters is reduced by 2.77× and the inference time is accelerated by 1.70×, with only 0.41% accuracy loss compared with original model. All these three metrics are significantly better than previous work [7], [21].

C. *Experiments on* ImageNet

1) AlexNet: AlexNet [15] has 5 convolutional layers and 3 FC layers. It is tested for the ImageNet classification task. We evaluate the top-5 accuracy with single-view. The ILSVRC-2012 dataset consists of 1.2 million training images and 50 thousand test images. Images are resized with 256 pixels on the shorter side. The testing image is on the center crop of 224 × 224 pixels. We use the pre-trained model provided by Caffe Model Zoo as the baseline. In our experiment, we first select 1500 images from the training set and collect their responses for building the approximate network. The layer-wise approximation results are demonstrated in TABLE IV. The first two convolutional layers of *AlexNet* are not approximated, in order to preserve good accuracy. For FC layers, again we only use the structured sparse component for approximation.

TABLE IV: Results on *AlexNet* with ILSVRC-2012

Layer	$CR(\boldsymbol{A})(\%)$	$CR(\boldsymbol{B})(\%)$	$CR(\boldsymbol{A}+\boldsymbol{B})(\%)$
`conv1`	0.0	100.0	100.0
`conv2`	21.4	23.6	45.0
`conv3`	30.0	22.9	52.9
`conv4`	0.0	32.6	32.6
`conv5`	0.0	26.0	26.0
`fc1`	12.8	0.0	12.8
`fc2`	26.2	0.0	26.2
`fc3`	18.8	0.0	18.8
CR	18.0%	(5.56× reduction of model size)	
Speed-up		1.1×	
Top-5 accu. ↓		1.27%	

The compression rates and the accuracy comparison are shown in TABLE VI. From the table we see that the network is compressed by more than 5×, which outperforms [7], [31], and [18], while the top-5 accuracy drop is only 1.3%. This reveals that the proposed approximation framework can remarkably compress *AlexNet* while keeping good accuracy.

2) GoogLeNet: GoogLeNet [16] is another widely used network in image recognition and classification. Different from *AlexNet*, *GoogLeNet* combines two spatial sizes of convolutional filters, 3×3 and 5×5, in each inception block. In order to collect the input samples for optimization, we use a pre-trained model provided by Caffe Model Zoo to perform inference and dump the input and output feature maps of each convolutional layer. After performing approximation on *GoogLeNet*, both model size and inference time are reduced. The layer-wise approximation results are shown in TABLE V. The comparison of accuracy loss, compression rates and accuracy are shown in TABLE VI. We can see that the model size is reduced by 2.87× and the inference time is accelerated by 1.35×, without loss on accuracy. All these three metrics are significantly better than previous works using the same network model and dataset [7], [18], [31].

VI. Conclusion

In this paper, we have proposed a unified approximation model for deep neural networks with simultaneous low-rank approximation and structured sparsification. It also considers the non-linear activation to retain the accuracy. To obtain this model, a layer-wise optimization problem is presented, relaxed, and solved with an extended ADMM algorithm

TABLE V: Results on *GoogLeNet* with ILSVRC-2012

Layer	$CR(\boldsymbol{A})(\%)$	$CR(\boldsymbol{B})(\%)$	$CR(\boldsymbol{A}+\boldsymbol{B})(\%)$
`conv1`	0.0	100.0	100.0
`conv2`	0.0	100.0	100.0
`inception-3a`	2.1	31.1	33.2
`inception-3b`	5.4	39.8	45.3
`inception-4a`	3.8	28.6	32.4
`inception-4b`	2.3	23.7	26.1
`inception-4c`	8.9	29.9	38.9
`inception-4e`	2.7	23.7	26.5
`inception-5a`	2.4	28.8	31.2
`inception-5b`	1.6	31.6	33.3
`fc`	35.0	0.0	35.0
CR	34.8%	(2.87× reduction of model size)	
Speed-up		1.35×	
Top-5 accu. ↓		0.00%	

TABLE VI: Comparison on ILSVRC-2012

Model	Method	Top-5 Accu.↓	CR	Speed-up
AlexNet	Original	0.00%	1.00	1.00
	ICLR'16 [7]	**0.37%**	5.00	**1.82**
	ICLR'16 [31]	1.70%	5.46	1.81
	CVPR'18 [18]	1.43%	1.50	-
	Ours	1.27%	**5.56**	1.10
GoogleNet	Original	0.00%	1.00	1.00
	ICLR'16 [7]	0.42%	2.84	1.20
	ICLR'16 [31]	0.24%	1.28	1.23
	CVPR'18 [18]	0.21%	1.50	-
	Ours	**0.00%**	**2.87**	**1.35**

whose convergence is provably guaranteed. The effectiveness of the proposed approximation framework is verified on *VGG-16*, *NIN*, *GoogLeNet* and *AlexNet*. By sacrificing little accuracy, *VGG-16* and *AlexNet* are compressed by up to 5.56×. *GoogLeNet* is compressed by nearly 3× without loss of accuracy. What's more, since structured sparse filters and low-rank filters are independent to each other, more inference speedup may be expected if taking actual architecture and parallel computing into account.

VII. Acknowledgment

This work is partially supported by Tencent Technology, The Research Grants Council of Hong Kong SAR (Project No. CUHK24209017), Beijing National Research Center for Information Science and Technology (BNR2019ZS01001), and NSFC under grant (No. 61872206, No. 61976164).

References

[1] M. Denil, B. Shakibi, L. Dinh, N. De Freitas *et al.*, "Predicting parameters in deep learning," in *Proc. NIPS*, 2013, pp. 2148–2156.

[2] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proc. ICLR*, 2016.

[3] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. NIPS*, 2016, pp. 2074–2082.

[4] R. Dai, L. Li, and W. Yu, "Fast training and model compression of gated RNNs via singular value decomposition," in *Proc. IJCNN*, 2018.

[5] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *Proc. CVPR*, 2015, pp. 1984–1992.

[6] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, "Tensorizing neural networks," in *Proc. NIPS*, 2015, pp. 442–450.

[7] C. Tai, T. Xiao, Y. Zhang, X. Wang *et al.*, "Convolutional neural networks with low-rank regularization," in *Proc. ICLR*, 2016.

[8] J. M. Alvarez and M. Salzmann, "Compression-aware training of deep networks," in *Proc. NIPS*, 2017, pp. 856–867.

[9] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," in *Proc. CVPR*, 2017, pp. 7370–7379.

[10] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. ICCV*, 2017.

[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. ICML*, 2010, pp. 807–814.

[12] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. ICLR*, 2015, pp. 1–14.

[14] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012, pp. 1097–1105.

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

[17] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. ICLR*, 2017.

[18] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "NISP: Pruning networks using neuron importance score propagation," in *Proc. CVPR*, 2018.

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

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

[21] S. P. Kasiviswanathan, N. Narodytska, and H. Jin, "Network approximation using tensor sketching," in *Proc. IJCAI*, 2018, pp. 2319–2325.

[22] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li, "Coordinating filters for faster deep neural networks," in *Proc. ICCV*, 2017, pp. 658–666.

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

[24] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact cnns," in *Proc. ECCV*, 2016, pp. 662–677.

[25] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Yu, and Y. Ma, "Robust recovery of subspace structures by low-rank representation," *IEEE TPAMI*, vol. 35, no. 1, pp. 171–184, 2013.

[26] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on Optimization (SIOPT)*, vol. 20, no. 4, pp. 1956–1982, 2010.

[27] B.-S. He and X. Yuan, "A class of ADMM-based algorithms for three-block separable convex programming," *Computational Optimization and Applications*, 2018.

[28] C. Chen, B. He, Y. Ye, and X. Yuan, "The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent," *Mathematical Programming*, vol. 155, no. 1-2, pp. 57–79, 2016.

[29] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. Multimedia*, 2014, pp. 675–678.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. CVPR*, 2009, pp. 248–255.

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