Toward Compact ConvNets via Structure-Sparsity Regularized Filter Pruning

Shaohui Lin, Student Member, IEEE, Rongrong Ji, Senior Member, IEEE, Yuchao Li, Cheng Deng, Member, IEEE, and Xuelong Li, Fellow, IEEE

Abstract—The success of convolutional neural networks (CNNs) in computer vision applications has been accompanied by a significant increase of computation and memory costs, which prohibits their usage on resource-limited environments, such as mobile systems or embedded devices. This to, the research of CNN compression has recently become emerging. In this paper, we propose a novel filter pruning scheme, termed structured sparsity regularization (SSR), to simultaneously speed up the computation and reduce the memory overhead of CNNs, which can be well supported by various off-the-shelf deep learning libraries. Concretely, the proposed scheme incorporates two different regularizers of structured sparsity into the original objective function of filter pruning, which fully coordinates the global output and local pruning operations to adaptively prune filters. We further propose an alternative updating with Lagrange multipliers (AULM) scheme to efficiently solve its optimization. AULM follows the principle of alternating direction method of multipliers (ADMM) and alternates between promoting the structured sparsity of CNNs and optimizing the recognition loss, which leads to a very efficient solver ((2.5 x) to the most recent work that directly solves the group sparsity-based regularization). Moreover, by imposing the structured sparsity, the online inference is extremely memory-light since the number of filters and the output feature maps are simultaneously reduced. The proposed scheme has been deployed to a variety of state-of-the-art CNN structures, including LeNet, AlexNet, VGGNet, ResNet, and GoogLeNet, over different data sets. Quantitative results demonstrate that the proposed scheme achieves superior performance over the state-of-the-art methods. We further demonstrate the proposed compression scheme for the task of transfer learning, including domain adaptation and object detection, which also show exciting performance gains over the state-of-the-art filter pruning methods.

Index Terms—Convolutional neural networks (CNNs), CNN acceleration, CNN compression, structured sparsity.

I. INTRODUCTION

IN RECENT years, convolutional neural networks (CNNs) have achieved great success on a variety of computer vision applications, ranging from action recognition [1], object recognition [2]–[8], to object detection [9]–[12]. One essential foundation of such success lies in the gigantic amount of model parameters to accompany with the large-scale training data. For instance, ResNet-152 [3] has 57 million parameters, costs 230-MB storage, and requires 11.3 billion FLOPs\(^1\) to classify one image with a resolution of 224 \(\times\) 224. Under such a circumstance, these models cannot be directly deployed to scenarios that require fast processing or compact storage, such as mobile systems or embedded devices.

Substantial efforts have been devoted to the speedup and compression of CNNs for efficient online inference. Among the existing methods, network pruning has attracted increasing attention recently due to its ability to reduce an overwhelming amount of parameters. As one of the earliest works, LeCun et al. [13] proposed an optimal brain damage algorithm to prune the network by reducing the number of connections with a theoretically justified saliency measurement. Later, Hassibi and Stork [14] proposed an optimal brain surgeon algorithm to remove unimportant parameters, which are determined by the second-order derivatives of their weights. Recently, Han et al. [15], [16] proposed to prune the parameters with small magnitudes to reduce the network size. However, the above-mentioned pruning schemes typically produce sparse CNNs with nonstructured random connections, which will cause irregular memory access, i.e., a complex storage structure that adversely impacts the efficiency of accessing CNN models in memory. Moreover, such nonstructured sparse CNNs cannot be supported by off-the-shelf libraries, which thus need specialized hardware [17] or software [18] designs to improve their efficiency in online inference.

To address the shortcoming of such nonstructured connections, filter pruning is regarded as a promising solution, which has shown significant speed up in online inference as well as good independence to software/hardware platforms [19]–[23]. Differing from the previous works in parameter pruning, filter pruning can be

\(^1\)FLOPs: the number of floating-point operations.
further integrated by various CNN compression or acceleration methods, e.g., low-rank decomposition [24]–[31], discrete cosine transform (DCT)-based [32] or fast Fourier transform (FFT)-based [33], [34] frequency-domain acceleration, and parameter quantization [31], [35]–[38]. Its another advantage lies in reducing the energy consumption [39], which is not only influenced by the parameter amounts but also influenced by the FLOPs and memory access of input/output feature maps. From this perspective, filter pruning can directly remove FLOPs and the intermediate activations of filters (e.g., output feature maps and their corresponding filter channels in the next layer), which substantially reduces the energy consumption.

However, there are still open issues in the existing filter pruning schemes. Concretely, how to adaptively and efficiently select important filters to reconstruct the global output, i.e., the probabilistic “softmax” output after local filter pruning, is still an open problem, with only a few works in the literature. For instance, the work in [20] proposed a magnitude-based criterion to prune convolutional filters with small $\ell_1$-norms. It has resulted in structured sparse patterns\(^2\) that can accelerate the online inference. However, such a magnitude-based measurement (e.g., $\ell_1$-norm) is too simple and ineffective to determine the importance of each filter due to the existence of nonlinear activation functions [e.g., rectifier linear unit (ReLU) [40]] and other complex operations (e.g., pooling and batch normalization [41]). To explain, filters with small $\ell_1$-norm values may have large responses in the output. For example, given two filter vectors $a = (0.1, 0.5, 1)\top$ and $b = (0.2, 0.1, 0)\top$ with an input $c = (1, 0, 0)\top$, we have $\|a\|_1 > \|b\|_1$, while $\text{ReLU}(c \ast a) < \text{ReLU}(c \ast b)$ after convolutional operator and ReLU activation. Very recently, Luo et al. [19] implicitly associated the convolutional filter of each layer to its input channel of the next layer, upon which filter pruning is done by selecting input channels that have minimal local reconstruction error. The small reconstruction error, however, might be magnified and propagated in the deep networks, leading to large reconstruction error in the global output. A Taylor expansion (TE)-based criterion [42] was further proposed to iteratively prune one feature map and its associated filter. The pruned network is then fine-tuned to reduce the accuracy drop. However, such a scheme is unadaptive and costly in pruning the entire network. Group sparsity [21]–[23], [43], [44] was introduced to select unimportant filters by the stochastic gradient descent (SGD). However, SGD is less efficient in convergence, which is also less efficient to generate the structured output of the pruned filters.

In this paper, we propose an efficient filter pruning scheme, termed structured sparsity regularization (SSR), which can efficiently and adaptively prune a group of convolutional filters to minimize the classification error of the global output. Compared with the existing works of sequential filter pruning [19], [20], [23], [42], [45], we incorporate the structured sparsity constraint into the objective function of global output to model the correlation between the global output loss and the local filter removal, which produces a structured network with fast computing and light memory consumption.\(^3\) In particular, we propose two different kinds of structured sparse regularizer for adaptive filter pruning, i.e., $\ell_{2,1}$-norm [46] and $\ell_{2,0}$-norm. The $\ell_{2,0}$-norm of a matrix $A$ is defined as $\|A\|_{2,0} = \sum_i \|\sum_j A_{ij}\|_0^2$, where for a scalar $a$, $\|a\|_0 = 0$ if $a = 0$, and $\|a\|_0 = 1$, otherwise. The $\ell_{2,1}$-norm is a convex norm to approximate the cardinality in filter selection, while the $\ell_{2,0}$-norm selects a filter with the constraint of explicit cardinality, which is the most natural constraint. As for group sparsity with $\ell_{2,0}$-regularization, SGD in the previous works [21]–[23], [43] cannot solve the NP-hard problem for effective filter pruning. To this end, we propose a novel alternative updating with Lagrange multipliers (AULM), which handles the convergence difficulty with $\ell_{2,1}$-norm using SGD and the NP-hard problem with $\ell_{2,0}$-norm. AULM follows the principle of alternating direction method of multipliers (ADMM) [47] by splitting the optimization problem into tractable subproblems that can be solved efficiently. In addition, AULM has a faster convergence than ADMM by adding Nesterov’s optimal method [48], can effectively identify the importance of filters, and then updates the parameters by alternating between promoting the structured sparsity and optimizing the recognition loss. In particular, the proposed solver circumvents the sparse constraint evaluations during the standard backpropagation step, which makes the implementation very practical. Moreover, compared with structured sparsity learning (SSL) [23], the proposed solver is much faster to the existing solvers in off-line pruning, i.e., almost $2.5 \times$ than directly solving the regularizer with $\ell_{2,1}$-norm by using SGD. Fig. 1 shows the proposed filter pruning framework.

Quantitatively, we demonstrate the advantage of the proposed SSR scheme using five widely used models (i.e., LeNet, AlexNet, VGGNet-16, ResNet-50, and GoogLeNet) on two data sets (i.e., MNIST and ImageNet 2012). Compared with several state-of-the-art filter pruning methods [19], [20], [23], [42], [43], [45], [49], the proposed scheme performs drastically better, i.e., $5.5 \times$ CPU speedup and $39.1 \times$ compression with a negligible classification accuracy loss for LeNet on MNIST, $2.1 \times$ CPU speedup with an increase of $1.28\%$ Top-5 classification error for AlexNet, $2.4 \times$ GPU speedup with a decrease (even better) of $1.65\%$ Top-1 classification error for VGG-16, $1.9 \times$ CPU speedup and $2.13 \times$ compression with an increase of $3.65\%$ Top-1 classification error for ResNet-50, and $1.6 \times$ CPU speedup and $1.7 \times$ compression with an increase of $1.05\%$ Top-5 classification error for GoogLeNet in ImageNet 2012. Moreover, the pruned AlexNet and VGG-16 can be further compressed by replacing the original fully connected layers with global average pooling (GAP) [50], leading to $24.96 \times$ and $15 \times$ compression rates only with an increase of $4.62\%$ and $0.27\%$ Top-5 classification error, respectively.

In addition, we also explore the generalization ability of the SSR-based compressed model in more complex tasks.

\(^2\)Structured sparsity is to directly prune the filter/block (i.e., set the values of filter/block to zeros), while nonstructured sparsity only justifies whether each element in the filter is zero or not.

\(^3\)To make a fair comparison, we only evaluate the computational cost of the model for online inference, without including the training
Fig. 1. Illustration of an SSR scheme. (a) Complete process includes adaptively pruning the unimportant filters via the AULM solver, removing the corresponding output feature maps, removing the corresponding channels of filters in the next layer, as well as updating filters and fine-tuning the pruned network. (b) Process to select and prune unimportant filters, the corresponding output feature maps, and channels in the next layer (highlighted in purple). In particular, the tensor-based convolutional operator can be replaced by a matrix-by-matrix multiplication using BLAS library to accelerate the computation of CNNs. (Best viewed in color.)

i.e., domain adaptation and object detection. The experimental results demonstrate that such a compressed model has achieved $3.5 \times$ FLOPs reduction and $15.4 \times$ compression only with an increase of $1.95\%$ Top-1 error on the task of domain adaptation, as well as $0.3\%$ mean average precision (mAP) drops with a factor of $2.45 \times$ GPU speedup on the task of object detection. These two results are highly competitive comparing to the state-of-the-art filter pruning methods.

II. RELATED WORK

Early works in network compression mainly focus on compressing the fully connected layers [13]–[16], [51]. For instance, LeCun et al. [13] and Hassibi et al. [14] proposed a saliency measurement by computing the Hessian matrix of the loss function with respect to the parameters, based on which network parameters with low saliency values are pruned. Srinivas and Babu [51] explored the redundancy among neurons to remove a subset of neurons without retraining. Han et al. [15], [16] proposed a pruning scheme based on low-weight connections to reduce the total amount of parameters in CNNs. However, these methods only reduce the memory footprint and do not guarantee to reduce the computation time since the time consumption is mostly dominated by the convolutional layers. Moreover, the above-mentioned pruning schemes typically produce nonstructured sparse CNNs that lack flexibility to be applied across different platforms or libraries. For example, the compressed sparse column (CSC)-based weight formation has to change the original format of weight storing in Caffe [52] after pruning, which cannot be well supported across different platforms.

To reduce the computation cost of convolutional layers, a popular solution is to decompose the convolutional filters into a sequence of tensors with fewer parameters [25], [26], [28], [30], [31], [53]. The convolution can also be conducted in the frequency domain using DCT [32] and FFT [33], [34], or approximated by the balanced decoupled spatial convolution [54] to reduce the redundancy of spatial and channel information. Besides, binarization of weights [36], [37], [55] and low-complexity weights [56] can also be employed in the convolutional layers to reduce the computation overheads with multiplication-free operations. Designing a compact filter is also able to accelerate the convolutional computation by replacing the overparametric filters with a compact block, such as inception module in GoogLeNet [5], bottleneck module in ResNet [3], fire module in SqueezeNet [57], group convolution [4], [58], [59], and depthwise separable convolution [6], [60], [61]. Without incurring additional overheads, our scheme can be integrated with the above-mentioned schemes to further speed up the computation since the above-mentioned schemes are orthogonal to the core contribution of this paper.

In line with our work, some recent works have investigated structured pruning to remove redundant filters or feature maps, which can be categorized into either greedy-based pruning [19], [20], [42], [62], [63] or sparsity regularization-based pruning [21]–[23], [43], [49], [64]. For the former group, the work in [20] proposed a magnitude-based pruning to prune filters with their corresponding feature maps by measuring the $\ell_1$-norm of filters, which is, however, inefficient in determining the importance of filters. He et al. [62] proposed an $\ell_2$-norm criterion to prune unsalient filters in a soft manner. Luo et al. [19] explored the importance of the input channel from the next convolutional layer, based on which conducted a local channel selection to prune unimportant input channels and the corresponding filters in the current layer. However, the small local reconstruction error might lead to a large error in the global output by propagating through the deep network. A TE-based criterion was proposed in [42] to iteratively prune one filter and then fine-tune the pruned network, which is, however, prohibitively costly for deep networks.
Lin et al. [63] proposed a global and dynamic pruning scheme to reduce redundant filters by greedy alternative updating. Alternatively, group sparsity-based regularization was proposed in [21]–[23], [43], and [44] to penalize unimportant parameters and prune redundant filters directly by using SGD, which is also very slow in convergence for filter selection. To reduce redundancies in the model parameters, the combination of group and exclusive sparsity was proposed in [49] to promote sharing and competition for features, respectively. Different from these group sparsity-based regularizations, we investigate the structured sparsity of the filters instead, which is also very slow in convergence for filter selection. To reduce redundancies in the model parameters, the combination of group and exclusive sparsity was proposed in [49] to promote sharing and competition for features, respectively.

In addition, we consider several norms of filter matrix $\mathbf{K}^l$, which are used in the regularization term. For example, the Frobenius norm of the filter matrix $\mathbf{K}^l$ is defined as $\|\mathbf{K}^l\|_F := (\sum_{i,j} |\mathbf{K}^l_{ij}|^2)^{1/2}$. The sparsity inducing $\ell_1$-norm is defined as $\|\mathbf{K}^l\|_{1,1} := \sum_{i} |\mathbf{K}^l_{i,1}|$. In this paper, we introduce two different structured sparsity norms to adaptively select the unimportant filters to be pruned, i.e., $\ell_2,1$-norm and $\ell_2,0$-norm, which are denoted as $\|\mathbf{K}^l\|_{2,1} := \sum_{i} |\mathbf{K}^l_{i,1}|$ and $\|\mathbf{K}^l\|_{2,0} := \sum_{i} (\sum_{j} |\mathbf{K}^l_{i,j}|)^{1/2}$. Note that $\ell_2,0$-norm is not a valid norm because it does not satisfy the positive scalability: $\|a\mathbf{K}^l\|_{2,0} = |a| \|\mathbf{K}^l\|_{2,0}$ for any scalar $a$. The term “norm” here is for convenience.

### B. Framework of SSR

SSR prunes the least important filters from a trained convolutional network to reduce the computation and memory costs. Its procedure consists of three basic operations, that is: 1) evaluate the importance of each filter; 2) prune unimportant filters; and 3) fine-tune the whole network. Differing from the previous filter pruning, we adaptively select unimportant filters to be pruned by using AULM. As shown in Fig. 1, we focus on the blue dotted boxes that perform the AULM solver to adaptively prune unimportant filters and then remove the corresponding output feature maps and the filter channels in the next layer. We present the principle steps of SSR as follows.

1. **Automatic Filter Selection:** We design a novel objective function, which incorporates the structured sparsity constraint into the data error term, e.g., cross entropy between the probability of data inference and the ground truth. The optimization problem can be solved by the proposed AULM solver. Thus, the unimportant filters are adaptively identified during the training.
2. **Pruning:** We prune the unimportant filters and their corresponding feature maps, together with the channels of filters in the next layer.
3. **Updating:** We update the rest filters and feature maps in the current layer, as well as the channels of filters in the next layer.
4. **Iteration of Step 1 to prune the next layer.**
5. **Global Fine-Tuning:** We globally fine-tune the pruned networks, which recovers the discriminability and generalization ability for the pruned networks.

### C. Objective Function of SSR

Instead of directly pruning the filters by calculating their corresponding magnitudes [20], [45], SSR utilizes the struc-
tured sparsity to seek the best tradeoff between loss minimization and filter selection. We consider the following objective function:

$$\min_{K} \mathcal{L}(Z, f(X; K)) + \lambda g(K). \quad (2)$$

Here, $K$ represents the collection of all weights in CNNs. $\mathcal{L}(Z, f(X; K))$ is the cross-entropy loss for classification or mean-squared error for regression between the labels of ground truth $Z$ and the output of the last layer in CNNs $f(X; K)^4$ where $D = \{X, Z\} = \{X_i, Z_i\}_{i=1}^N$ is a training data set with $N$ instances. We denote the first loss term in (2) as $L_D(K)$ for simplicity. The term $g(K)$ is certain SSR on the total size of the remaining filters in each iteration. In this paper, we consider two different kinds of structured sparsity regularizer, i.e., $\ell_{2,1}$-norm and $\ell_{2,0}$-norm.

Parameter $\lambda$ is the penalty term of structured sparsity. As $\lambda$ varies, the solution of (2) traces a tradeoff path between the performance and structured sparsity. Note that the $\ell_{2,0}$- or $\ell_{2,1}$-norm-based structured sparse regularizer $g(K)$ and its efficient solver in (2) are nontrivial.

### D. AULM Solver

The proposed AULM solver aims at solving the problem of SSR, which is inspired by ADMM in the field of distributed optimization. Different from ADMM, we focus on selecting and pruning unimportant filters by solving a nonconvex optimization problem with the $\ell_{2,1}$-regularization and an NP-hard problem with the $\ell_{2,0}$-regularization. In particular, to handle the nontrivial regularizer in (2), we introduce a slack variable and an equality constraint as follows:

$$\min_{K, F} L_D(K) + \lambda g(F) \quad \text{s.t.} \quad K = F. \quad (3)$$

AULM is an iterative method that augments the Lagrangian function with quadratic penalty terms. The augmented Lagrangian associated with the constrained problem of (3) is given by

$$L(K, F, Y) = L_D(K) + \sum_{l=1}^{L} \lambda g(F^l) + \sum_{l=1}^{L} \text{trace}(Y^l^T(K^l - F^l)) + \sum_{l=1}^{L} \frac{\rho}{2} \|K^l - F^l\|_F^2 \quad (4)$$

where $K^l$, $F^l$, and $Y^l$ are the filter kernel, the intermediate filter with structured sparsity, and the dual variables (i.e., the Lagrange multipliers) at the $l$th layer, respectively, $\rho > 0$ is a penalty parameter. To minimize (4), AULM solves for each variable via a sequence of iterative computations as follows:

1) Employ the gradient descent to minimize the loss of

$$K^{[l+1]} = \min_{K} L(K, \hat{F}^{[l]}, \hat{Y}^{[l]}). \quad (5)$$

The above-mentioned four steps are applied in an alternating manner. In the following, we describe the details of steps 1 and 2 to obtain $K$ and $F$. The proposed alternative optimization is summarized in Algorithm 1. By overrelaxing the Lagrange multiplier variables after each iteration, AULM not only has a faster convergence but also obtains a more effective solution compared with ADMM.

**Algorithm 1: AULM for Structured Pruning CNN**

**Input:** Training data points $D$, pre-trained CNN weights $K$, a set of regularization factors $S$.

**Output:** The structured pruning filters $K$.

1: Initialize: dual variables $\hat{Y} = Y = 0$, $\hat{F} = F = K$, and $\rho = 1$.
2: for each $\lambda$ in $S$ do
3:  for each $l$ in $[1, L]$ do
4:      repeat
5:         Step 1: Find the estimation of $K^{[l+1]}$ by solving the problem in Eq. (11) using SGD;
6:         Step 2: Find the structured sparsity estimation of $F^{[l+1]}$ with $\ell_{2,1}$-norm or $\ell_{2,0}$-norm from Eq. (14) or Eq. (15), respectively;
7:         Step 3: Update dual variables $Y^{[l+1]}$ by Eq. (7);
8:         Step 4: Update accelerated variables $\hat{Y}^{[l+1]}$ and $\hat{F}^{[l+1]}$ by Eq. (8) and Eq. (9), respectively;
9:      until $\|K^{[l+1]} - K^{[l]}\|_F \leq \epsilon$, or $\|F^{[l+1]} - F^{[l]}\|_F \leq \epsilon$;
10:     Prune filters $K^l$ corresponding to the row-index of $F^l$ with zeros and their corresponding feature maps.
11:   end for
12:  the pruned network.
13: end for

2) Find the closed-form solution of the structured sparsity

$$F^{[k+1]} = \min_{F} \mathcal{L}(K^{[k+1]}, F, \hat{Y}^{[k]}). \quad (6)$$

3) Update the dual variables $Y^l$ using gradient ascent with a step-size equal to $\rho$, that is

$$Y^{[k+1]} = Y^{[k]} + \rho (K^{[k+1]} - F^{[k+1]}). \quad (7)$$

4) Conduct an overrelaxation step for accelerated variables $\hat{F}$ and $\hat{Y}$ with a step-size equal to $\gamma$, that is

$$\hat{Y}^{[k+1]} = \gamma Y^{[k+1]} + (1 - \gamma) (Y^{[k+1]} - Y^{[k]}) \quad (8)$$

$$\hat{F}^{[k+1]} = \gamma F^{[k+1]} + (1 - \gamma) (F^{[k+1]} - F^{[k]}) \quad (9)$$

where $\gamma = k/(k + r)$, with $r \geq 3$ ($r = 3$ is the standard choice).

The following steps are applied in an alternating manner. In the following, we describe the details of steps 1 and 2 to obtain $K$ and $F$. The proposed alternative optimization is summarized in Algorithm 1. By overrelaxing the Lagrange multiplier variables after each iteration, AULM not only has a faster convergence but also obtains a more effective solution compared with ADMM.

1) **Updating of AULM:** We describe the details of steps 1 and 2 to obtain $K$ and $F$ in the following.

**Step 1:** By removing the penalty terms of $g(F^l)$ and completing the squares with respect to $K$ in (4), we obtain the following equivalent problem to (5):

$$\min_{K} L_D(K) + \sum_{l=1}^{L} \frac{\rho}{2} \|K^l - T_l^1\|_F^2 \quad (10)$$

For simplicity, the term of weight decay (i.e., nonstructured regularization applying on every weight, e.g., $\ell_2$-norm) is omitted, since it can be directly incorporated into the loss function and does not affect the result of SSR.
where \( T_1^l = F^l - (1/\rho)Y^l \). To obtain the suboptimal filters \( K^l \) in the layerwise pruning framework, we separately update \( K^l \) in the \( l \)th layer with the following optimization problem:

\[
\min_{K^l} \mathcal{L}_D(K^l) + \frac{\rho}{2} \|K^l - T_1^l\|_F^2.
\]  

(11)

We use SGD to optimize the filters \( K^l \), which is a reasonable choice to handle such a high-dimensional optimization. The entire procedure relies mainly on the standard forward-backward pass.

Step 2: By removing the first term \( \mathcal{L}_D(K) \) and completing the squares with respect to \( F \) in (4), we obtain the following equivalent problem to (6):

\[
\min_{F^l} \lambda \left( \sum_{i=1}^{L} \|F^l - T_2^l\|_F^2 \right) + \frac{\rho}{2} \left\| \sum_{j} F_{ij}^l \right\|_0.
\]  

(12)

where \( T_2^l = K^l + (1/\rho)Y^l \). We update \( F^l \) layer-by-layer instead of directly updating the whole layers. Hence, we get the following optimization problem at the \( l \)th layer:

\[
\min_{F^l} \lambda \|g(F^l)\|_2 + \frac{\rho}{2} \|F^l - T_2^l\|_F^2.
\]  

(13)

Based on (13), we can obtain a closed-form solution by considering the following regularizers \( g(F^l) \).

1) \( \ell_{2,1} \)-Norm: A closed-form solution of (13) can be derived, which is evaluated row-by-row on \( T_2^l \) [66], [67]. The \( i \)th row is calculated via

\[
F^l_{i} = T_{2i}^l \max \left\{ \left\| T_{2i}^l \right\|_2 - \frac{\lambda}{\rho}, 0 \right\}.
\]  

(14)

2) \( \ell_{2,0} \)-Norm: A closed-form solution of (13) with this regularizer can also be derived, which is evaluated row-by-row on \( T_2^l \) in Theorem 1. The \( i \)th row is calculated via

\[
F^l_{i} = \begin{cases} 
0, & \lambda \geq \frac{\rho}{2} \left\| T_{2i}^l \right\|_2 \\
T_{2i}^l, & \lambda < \frac{\rho}{2} \left\| T_{2i}^l \right\|_2
\end{cases}
\]  

(15)

3) \( \ell_1 \)-Norm: The norm is not a structural constraint, which leads to unstructured sparsity by solving the problem in (13).\(^5\) Specifically, we obtain a closed-form solution of (13) with this constraint by evaluating on each entry of \( T_2^l \). The optimal solution \( F^l_{ij} \) is obtained via

\[
F^l_{ij} = \text{sign}(T^l_{2ij}) \max \left\{ T^l_{2ij} - \frac{\lambda}{\rho}, 0 \right\}
\]  

(16)

where \( \text{sign}(\cdot) \) is an indicator function, that is

\[
\text{sign}(T^l_{2ij}) = \begin{cases} 
1, & T^l_{2ij} \geq 0 \\
-1, & \text{otherwise}
\end{cases}
\]

Proof: Since \( g(F^l) = \|F^l\|_{2,0} = \sum_j \|\sum_i F^l_{ij}\|_0^{1/2} \), we are interested in solving the following problem:

\[
\min_{F^l} \lambda \sum_i \left\| \sum_j F^l_{ij} \right\|_0 + \frac{\rho}{2} \|F^l - T_2^l\|_F^2.
\]  

(17)

We rewrite \( F^l \) and \( T_2^l \) as \( F^l = (F^l_1, F^l_2, \ldots, F^l_{C_{l+1}})^\top \) and \( T_2 = (T_{21}^l, T_{22}^l, \ldots, T_{2C_{l+1}})^\top \), where \( F^l \) and \( T_2^l \) are the \( i \)th row of \( F^l \) and \( T_2^l \), respectively. Then, we solve the following equivalent problem for each row independently to (13):

\[
\min_{F^l_i} \mathcal{L}(F^l_i) = \lambda \|F^l_i\|_2 + \frac{\rho}{2} \|F^l_i - T_{2i}\|_2^2.
\]  

(18)

On one hand, if \( \forall F^l_i \neq 0, \lambda \|F^l_i\|_2 = \lambda \), we obtain the optimal value \( \mathcal{L}(F^l_i) = \lambda \) when \( \|F^l_i - T_{2i}\|_2^2 = 0 \), i.e., \( F^l_i = T_{2i} \). On the other hand, if \( F^l_i = 0, \lambda \|F^l_i\|_2 = 0 \) and \( \mathcal{L}(F^l_i) = (\rho/2)\|T_{2i}\|_2^2 \). Therefore, when \( \lambda \geq (\rho/2)\|T_{2i}\|_2^2 \), the optimal solution is \( F^l_i = 0 \), while \( \lambda < (\rho/2)\|T_{2i}\|_2^2 \) the optimal solution is \( F^l_i = T_{2i} \). Thus, we can obtain the optimal solution of (13), which is given by (15) by considering the rowwise decoupling property of (13). \( \square \)

2) Convergence of AULM: Since the nonlinear transformation, normalization, and pooling commonly occur in CNNs, the objective function of (4) is highly nonconvex, which lacks a theoretical proof to guarantee its convergence to the global optimal. However, it is empirically shown that AULM works well when the penalty parameter \( \rho \) is sufficiently large. This is related to the quadratic term that tends to be locally convex by giving a sufficiently large \( \rho \) value. However, if \( \rho \) is too large, it is difficult for the iterative solver to take effects. As a tradeoff, we set \( \rho = 1 \) in our implementation.

Since the objective function is highly nonconvex, there is a risk to be trapped into a local optimum. In our implementation, we circumvent this difficulty by using the pretrained weights as the initialized weights, which performs quite well in practice.

E. Pruning on ResNet

Unlike VGG-16 and AlexNet, there are some restrictions to prune ResNet due to the special residual blocks. In general, each residual block with bottleneck structure contains three convolutional layers (followed by both batch normalization and ReLU) and shortcut connections. In order to perform the sum operator, the number of output feature maps in the last convolutional layer needs to be consistent with that of the projection shortcut layer. In particular, when the dimensions of input/output channels are mismatched in a residual block, a linear projection is performed by the shortcut connections (see [3] for more details).

In this paper, we focus on pruning the first two layers in each residual block, as shown in Fig. 2. And we do not prune the last convolutional layer of each residual block directly. In fact, the parameters (e.g., filter channel) in the last convolutional layer are much fewer, as a large proportion of filters in the second layer have been pruned.
We implement the proposed SSR scheme with Tensorflow 6, ResNet-50 [3], and GoogLeNet [5] on ImageNet [69]. We set LeNet on MNIST [68], AlexNet [4], VGG-16 [2], and 128-G RAM. The number of pruned filters is directly controlled by the hyperparameters $\lambda$, i.e., the regularization factor of the structured sparsity. In our experiments, we vary $\lambda$ in the set of $\{0.1, 0.2, \ldots, 0.8\}$ with eight values to select the best tradeoff between the compression/speedup rate and the accuracy. For $r$ in the overrelaxation step, we set it to be 3.

3) Evaluation Protocols: For evaluation protocols, we quantize the performance by using FLOPs, the number of parameters, and the Top-1/5 classification error. To make a fair comparison, the speedup rate is measured in a single-thread Intel Xeon E5-2620 CPU and an NVIDIA GTX TITAN X GPU.

4) Alternative and State-of-the-Art Approaches: We first compare our filter selection criterion with three alternative criteria, which are briefly summarized as follows.

1) Random: Randomly prune filters of each layer.

2) L1-Norm (Filter Norm) [20]: Filters with smaller magnitude tend to be unimportant filters. Therefore, the $\ell_1$-norm of each filter $s_i = |\mathbf{K}(i, \cdot)|_1$ is chosen as its importance score. We then sort these scores to prune filters correspondingly.

3) Average Percentage of Zeros [45]: The sparsity of each channel in output after ReLU activation can be chosen as the importance score of the corresponding filter. Then, the sparsity is calculated as $s_i = 1 - (1/\mathbb{I}(\cdot,i)) \sum \mathbb{I}(\cdot,i) = 0$, where $\mathbb{I}(\cdot,i)$ is the entry number of the $i$th channel in the tensor $\mathbb{I}$. The smaller $s_i$ is, the less important the corresponding filter is.

We also compare the proposed SSR scheme to the state-of-the-art filter pruning methods, including SSL [23], ThiNet [19], TE [42], combined group and exclusive sparsity (CGES) [49], and guided structured sparsity (GSS) [43]. Furthermore, we make a comparison with our alternative schemes with different regularizers, i.e., SSR with $\ell_2,1$-norm (termed SSR-L2,1), SSR with $\ell_2,0$-norm (termed SSR-L2,0), and SSR with $\ell_1$-norm (termed SSR-L1).

IV. EXPERIMENTS

A. Experimental Setups

1) Models and Data Sets: We conduct comprehensive experiments using five convolutional networks on two data sets, i.e., LeNet on MNIST [68], AlexNet [4], VGG-16 [2], ResNet-50 [3], and GoogLeNet [5] on ImageNet [69]. We implement the proposed SSR scheme with Tensorflow [65]. All pretrained CNNs except LeNet are taken from the Caffe model zoo. We make use of an open-source tool to convert the pretrained models to Tensorflow format and then fine-tune them to restore the accuracy. We train LeNet from scratch and report the results in Table I.

2) Implementations: To train the proposed SSR scheme, we use a learning rate of 0.001 with a constant dropping factor of 10 throughout 10 epochs. The weight decay is set to be 0.0005, and the momentum is set to be 0.9. To train SSR on both LeNet and AlexNet, the mini-batch size is set to be 256. To train VGG-16, ResNet-50, and GoogLeNet, the mini-batch size is all set to be 32. After pruning, the pruned network is fine-tuned for 30 epochs, in which a learning rate starts at 10$^{-3}$ and is scaled by 0.1 throughout 10 epochs. All the experiments are run on NVIDIA GTX 1080Ti graphics card with 11-GB and 128-G RAM. The number of pruned filters is directly controlled by the hyperparameters $\lambda$, i.e., the regularization factor of the structured sparsity. In our experiments, we vary $\lambda$ in the set of $\{0.1, 0.2, \ldots, 0.8\}$ with eight values to select the best tradeoff between the compression/speedup rate and the accuracy. For $r$ in the overrelaxation step, we set it to be 3.

B. LeNet on MNIST

MNIST is a small-scale data set, which contains a training set of 60000 images and a test set of 10000 images from ten classes. Each image is a 28 $\times$ 28 gray-scale handwritten digit image. LeNet on MNIST consists of 2 convolutional layers and 2 fully connected layers, which achieves an error rate of 0.88% on MNIST. The detailed structure of LeNet is

\begin{equation}
(20)C5 - MP2 - (50)C5 - MP2 - 500FC - 10FC - S
\end{equation}

where $(20)C5$ is a 5 $\times$ 5 convolutional layer with 20 filters, $MP2$ is a max-pooling layer with kernel size 2, 10FC is a fully connected layer with ten nodes, and $S$ is a softmax loss layer. Since the node number of the last layer is directly related to the number of classes, we prune the remaining layers except for the last layer. The regularization factor $\lambda$ is fixed for AlexNet, VGG-16, and ResNet-50, we also keep the number nodes of the final layer unchanged and prune other remaining layers.

---

**TABLE I**

**PRUNING RESULTS OF LE.NET ON MNIST. "NUM-NUM-NUM" IS THE NUMBER OF REMAINING FILTERS IN EACH LAYER. K/M/B MEANS THOUSAND/MILLION/BILLION IN THIS PAPER, RESPECTIVELY.**

<table>
<thead>
<tr>
<th>Method</th>
<th>#Filter/Node</th>
<th>FLOPs</th>
<th>#Param.</th>
<th>CPU (ms)</th>
<th>Speedup</th>
<th>Top-1 Err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet</td>
<td>20-50-500</td>
<td>7.3M</td>
<td>0.43M</td>
<td>26.4</td>
<td>1x</td>
<td>0%</td>
</tr>
<tr>
<td>SSL [23]</td>
<td>3-15-175</td>
<td>162K</td>
<td>45K</td>
<td>7.3</td>
<td>3.62 x 0.3%</td>
<td>20%</td>
</tr>
<tr>
<td>TE [42]</td>
<td>2-12-127</td>
<td>91K</td>
<td>28K</td>
<td>6.0</td>
<td>4.40 x 20%</td>
<td>20%</td>
</tr>
<tr>
<td>CGES [49]</td>
<td>-</td>
<td>332K</td>
<td>156K</td>
<td>5.7</td>
<td>4.64 x 0.05%</td>
<td>20%</td>
</tr>
<tr>
<td>GSS [43]</td>
<td>4-11-109</td>
<td>119K</td>
<td>21K</td>
<td>5.5</td>
<td>4.80 x 0.20%</td>
<td>20%</td>
</tr>
<tr>
<td>SSR-L1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SSR-L2,1</td>
<td>3-11-108</td>
<td>118K</td>
<td>21K</td>
<td>6.6</td>
<td>4.00 x 0.05%</td>
<td>20%</td>
</tr>
<tr>
<td>SSR-L2,0</td>
<td>2-11-106</td>
<td>93K</td>
<td>28K</td>
<td>5.6</td>
<td>4.71 x 0.20%</td>
<td>20%</td>
</tr>
</tbody>
</table>

---

**Fig. 2. Illustration of pruning ResNet. The red value is the number of remaining filters/channels.**

---

6Our source code is available at https://github.com/ShaoHuLu/SSR.
7https://github.com/BVLC/caffe/wiki/Model-Zoo
8https://github.com/ethereon/caffe-tensorflow
9The accuracies of models may be slight different from that reported by other works due to a different learning framework.
10For AlexNet, VGG-16, and ResNet-50, we also keep the number nodes of the final layer unchanged and prune other remaining layers.
on six groups, such as (0.1, 0.1, and 0.3), (0.3, 0.3, and 0.4), (0.3, 0.5, and 0.5), (0.4, 0.5, and 0.5), (0.5, 0.4, and 0.5), and
(0.5, 0.5, and 0.5). We compare our method to different criteria of filter selection [20], [45] and also to other state-of-the-art
methods in filter pruning [23], [42], [43], [49].

Fig. 3 shows the pruning results of LeNet based on different
filter selection criteria. Compared with these criteria and
baselines, both FLOPs and parameter sizes are significantly
reduced with a lower error by using the proposed SSR,
as expected. In contrast, the \( \ell^1 \)-norm-based filter selection per-
forms poorly. To explain, due to the nonlinear transformation
in the network, filters with small \( \ell^1 \)-norm are still likely to be
important, which have a large impact on the final loss function.
When a large proportion of filters with small \( \ell^1 \)-norm are
pruned, the classification error can be significantly increased.
Note that the simplest scheme of random filter selection works
reasonably well, which is due to the self-recovery ability
of the distributed representations [70]. However, the random
criterion is not robust in practice and may lead to large
accuracy loss when being applied to compress large network
(e.g., VGG-16). For an average percentage of zeros (APoZ),
the sparsity of feature maps is quite reasonable to prune
the redundant filters, which is due to the self-sparsity of the
pretrained model with ReLU activation. In contrast, compared
with these filter pruning methods, our SSR-L2,1 achieves the
best performance with an increase of 0.29% classification
error, \( 40.37 \times \) FLOPs reduction, and \( 51.89 \times \) parameter reduc-
tion. SSR-L2,0 achieves relatively consistent results with SSR-
L2,1. In particular, with a significant high compression ratio
(i.e., \( 52.03 \times \) parameter reduction ratio), SSR-L2,0 achieves
much better performance than SSR-L2,1, as the \( \ell^2,0 \)-norm is
used to directly measure the cardinality of the filter structure.

The quantitative performance for compressing LeNet using
the proposed scheme is further shown in Table I. First,
we found that the FLOPs do not directly reflect the actual
speedup ratio in online inference. For instance, compared
with LeNet, the proposed SSR-L2,1 can reach \( 19.5 \times \) FLOPs

<table>
<thead>
<tr>
<th>Model</th>
<th>#Param</th>
<th>PLOPs</th>
<th>CPU (ms)</th>
<th>GPU (ms)</th>
<th>Top-1 Error</th>
<th>Top-5 Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>62.4M</td>
<td>572.7M</td>
<td>2.194</td>
<td>30</td>
<td>0.43</td>
<td>0.88</td>
</tr>
<tr>
<td>VGG-16</td>
<td>138.4M</td>
<td>153.5M</td>
<td>3.287</td>
<td>266</td>
<td>31.66%</td>
<td>11.55%</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>25.5M</td>
<td>3.9B</td>
<td>7.790</td>
<td>355</td>
<td>24.84%</td>
<td>7.70%</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>7.0M</td>
<td>1.6B</td>
<td>3480</td>
<td>136</td>
<td>31.90%</td>
<td>11.45%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>#Param</th>
<th>Speedup</th>
<th>CPU (ms)</th>
<th>GPU (ms)</th>
<th>Top-1 Error</th>
<th>Top-5 Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>TE [42]</td>
<td>60.2M</td>
<td>1.8×</td>
<td>1.4×</td>
<td>1.87%</td>
<td>1.59%</td>
<td></td>
</tr>
<tr>
<td>SSL [23]</td>
<td>60.9M</td>
<td>1.5×</td>
<td>1.2×</td>
<td>1.32%</td>
<td>1.24%</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>58.2M</td>
<td>1.5×</td>
<td>1.3×</td>
<td>3.26%</td>
<td>2.48%</td>
<td></td>
</tr>
<tr>
<td>L1 [20]</td>
<td>48.0M</td>
<td>2.0×</td>
<td>1.5×</td>
<td>6.39%</td>
<td>4.66%</td>
<td></td>
</tr>
<tr>
<td>APoZ [45]</td>
<td>48.0M</td>
<td>2.0×</td>
<td>1.5×</td>
<td>4.31%</td>
<td>2.75%</td>
<td></td>
</tr>
<tr>
<td>SSR-L1</td>
<td>60.7M</td>
<td>1.0×</td>
<td>1.0×</td>
<td>0.79%</td>
<td>0.68%</td>
<td></td>
</tr>
<tr>
<td>SSR-L2,1</td>
<td>22.0M</td>
<td>1.2×</td>
<td>1.1×</td>
<td>1.58%</td>
<td>1.38%</td>
<td></td>
</tr>
<tr>
<td>SSR-L2,0</td>
<td>60.8M</td>
<td>1.6×</td>
<td>1.3×</td>
<td>0.19%</td>
<td>0.20%</td>
<td></td>
</tr>
<tr>
<td>SSR-L2,0-GAP</td>
<td>45.9M</td>
<td>2.1×</td>
<td>1.7×</td>
<td>1.57%</td>
<td>1.28%</td>
<td></td>
</tr>
<tr>
<td>SSR-L2,1-GAP</td>
<td>2.5M</td>
<td>1.8×</td>
<td>1.4×</td>
<td>0.05%</td>
<td>0.14%</td>
<td></td>
</tr>
<tr>
<td>SSR-L2,0-GAP</td>
<td>2.5M</td>
<td>1.8×</td>
<td>1.5×</td>
<td>5.25%</td>
<td>4.63%</td>
<td></td>
</tr>
</tbody>
</table>
reduction, with a $4 \times$ actual speedup ratio. To explain, memory accesses for both interlayer and intralayer can significantly increase computation consumption. Second, TE [42] achieves the best tradeoff between the speedup ratio and the classification error among all the baselines (e.g., SSL [23], CGES [49], and GSS [43]), as it inherits the effectiveness of filter selection by estimating the loss increase of pruning each filter with TE. Note that CGES+ [49] is the combination of iterative pruning [15] and CGES for further compressing LeNet, which achieves 0.04% increase of classification error using 10% parameters of the full network. Compared with CGES+, the proposed SSR-L1 with $\ell_1$-norm achieves a significantly higher compression ratio by 47.78× (i.e., 9K number of parameters), only with an increase of 0.12% Top-1 error. However, there are no structural constraints on filters/weights, which leads to very low speedup under the same hardware/software evaluation environment.11 Third, by using the proposed AULM with low speed up under the same hardware/software evaluation environment.

We take AlexNet and VGG-16, for instance, most layers are robust to prune, as shown in Fig. 4. But still, there exist a small number of sensitive layers, which locate at the top convolutional layers. For example, it is sensitive to prune the fourth and the fifth convolutional layers for AlexNet and the last three convolutional layers (i.e., Conv5_1, Conv5_2, and Conv5_3) for VGG-16. To explain, these top layers often have high-level semantic information that is necessary for maintaining the classification accuracy. In addition, pruning some filters in the specific layers (e.g., Conv1_1, Conv3_1, and Conv4_1 in VGG-16 when $\lambda$ is set to be 0.2) gets a slightly better accuracy compared with the original network, which reveals that the redundant filters can reduce the discriminability of the original network. Therefore, we use a different $\lambda$ value for each layer to reduce the impact of these sensitive layers (i.e., we set a large $\lambda$ value in insensitive layers and set a small one in sensitive layers).

2) Quantitative Results: Since the fully connected layers occupy over 90% storage in AlexNet and VGG-16, we replace the original fully connected layers with GAP [50] to further compress the whole network. “X-GAP” refers to the model using the GAP after all convolutional layers are pruned via “X” methods (e.g., ThiNet and SSR). The “X-GAP” is fine-tuned with the same fine-tuning parameters as described in Section IV-A.

As shown in Table III, we prune AlexNet with three groups of $\lambda$, such as (0.2, 0.4, 0.5, 0.6, 0.1, 0.1, and 0.3), (0.4, 0.5, 0.7, 0.6, 0.1, 0.3, and 0.3), and (0.2, 0.4, 0.5, 0.6, 0.1, and GAP). Compared with other filter pruning methods [20], [23], [42], [45], our SSR scheme achieves the best tradeoff between the speedup/compression rate and the Top-1/5 classification error. First, we compare SSR-L2,1 to three alternative selection criteria (i.e., random, L1-norm [20], and APoZ [45]) with the same pruning number in each layer.12 SSR-L2,1 achieves the lowest Top-1/5 classification error. To explain, all selection criteria are naive methods to prune the filters based on the statistical property, thus resulting in a large approximation error of each layer that is propagated

Fig. 4. Sensitivity of pruning filter in each layer. Left: sensitivity of AlexNet. Right: sensitivity of VGG-16.
throughout the network. Second, by directly employing SGD with filterwise sparsity to solve the SSL problem, the redundant filters cannot be pruned efficiently, which only achieves 1.5× CPU speedup with an increase of 1.32% Top-1 error.\textsuperscript{13}

Third, the work in [42] uses TE to approximate the loss increase, which is similar to ours but is with a totally different selection criterion. Quantitatively, it is time-consuming for pruning one filter and then fine-tuning the network iteratively. In contrast, SSR-L2,1 achieves the lowest Top-1 error increase of 1.72% and Top-5 error increase of 1.38% while reducing a much larger amount of parameters. Finally, we also compare two different kinds of SSRs (i.e., \(\ell_2,1\)-regularization and \(\ell_{2,0}\)-regularization) with elementwise sparsity regularization (i.e., \(\ell_1\)-regularization) and observe that SSR-L1 significantly reduces the memory storage with only 22.6M parameters, which is twice less than SSR-L2,1 and SSR-L2,0 with a comparable error increase. However, SSR-L1 does not boost the inference efficiency, as elementwise sparsity cannot significantly reduce the number of filters that the computation is on par with the full network (see later in Section IV-D for more detailed discussions). As for SSR, compared with SSR-L2,1, SSR-L2,0 achieves a lower error increase (i.e., 1.57% versus 1.72% in Top-1 error increase) with a much smaller amount of parameters and higher speedup, i.e., 45.9M parameters and 2.1× CPU speedup versus 48M parameters and 2.0× CPU speedup. Moreover, to further compress AlexNet using SSR, GAP enables the pruned network to be more compact, leading to a 24.96× compression rate (i.e., 2.5M parameters).

For VGG-16, we summarize the performance in comparison to [19], [20], [42], and [45] in Table IV. In the experiments, \(\lambda\) in the first ten convolutional layers is set to be (0.5, 0.4, 0.5, 0.3, 0.5, 0.3, 0.4, 0.5, and 0.3) with large values, while the last three convolutional layers are all set to be 0.1. \(\lambda\) is set to be (0.1 and 0.6) in the fully connected layers. First, instead of directly pruning filters, ThiNet [19] conducts a greedy local channel selection, while TE [42] uses a greedy feature map selection to prune the feature maps. Compared with ThiNet, TE achieves a higher GPU speedup (i.e., 2.7× versus 2.2× in ThiNet), but has a significant increase in Top-5 error, which affects the discriminative ability of the compressed model. For three alternative criteria of filter selection, APoZ [45] achieves the lowest increase in both Top-1 and Top-5 classification errors by the same factor of GPU and CPU speedup, e.g., 2.0× CPU speedup and 2.4× GPU speedup with a decrease of 0.64% Top-1 classification error and 0.43% Top-5 classification error, respectively. Compared with all the baselines with fully connected layers, SSR-L2,1 achieves the best tradeoff between the classification error and speedup, e.g., a decrease of 1.46% Top-1 error by a factor of 2.4× GPU speedup. To explain, the relationship between the final output and the local filters is directly considered in SSR, which can therefore adaptively prune redundant filters that have less impact on the global output. Second, by replacing with GAP, the network is further compressed by a large rate, e.g., ThiNet-GAP achieves 14.6× parameter reduction, i.e., 9.5M parameters versus 138.4M parameters in the full VGG-16. Compared with the above-mentioned three selection criteria and ThiNet-GAP, our SSR-L2,1-GAP still achieves the best performance, i.e., 2.5× GPU speedup and 15× parameter reduction, only with an increase of 0.52% Top-1 error. In addition, compared with SSR-L2,1-GAP, SSR-L2,0-GAP achieves a comparable result by a factor of 2.6× GPU speedup and 15.4× parameter reduction, with an increase of 0.83% Top-1 error.

We also perform the proposed SSR on multibranch networks, e.g., ResNet-50 and GoogLeNet. The results of SSR on ResNet-50 are shown in Table V. We prune ResNet-50 with two groups of \(\lambda\). In the first seven residual blocks of the first group, the hyperparameter \(\lambda\) of each residual block is set to be (0.4 and 0.3), while \(\lambda\) of each residual block is set to be (0.3 and 0.4) in the remaining residual blocks (i.e., nine residual blocks). At the second group, the corresponding \(\lambda\) value of each residual block is increased by 0.1 on the basis of the first group. Note that, we skip the first convolutional layer, which is pretty sensitive for pruning. In addition, we prune the first two layers in each residual block and leave the output and projection shortcuts of residual block unchanged, as shown in Fig. 2. We found that SGD in SSL [23] is not very effective to solve (2) with \(\ell_2,1\)-regularization, which leads to a significant error increase with a limited FLOPs reduction. For three alternative criteria of filter selection, APoZ [45] still achieves the lowest error increase at the same computation complexity and memory storage. Although ThiNet [19] achieves the best performance among these state-of-the-art baselines [20], [23], [45], it requires additional samples (i.e., new input/output pairs from hidden layers) in each layer to find the optimal channels for pruning, which is not only expensive to store additional training data sets but also time-consuming to collect them in off-line training. Moreover, ThiNet only reduces the reconstruction error of each layer, which, however, ignores the correlation between the local filter pruning and the global output, leading to the accumulation of reconstruction error. Compared with ThiNet, without supervised information of hidden layers, SSR-L2,0 employs the original ImageNet data set to improve the classification accuracy at the same speedup ratio and also achieves a higher parameter reduction (2.13× versus 2.07×). For GoogLeNet, we prune all the convolutional filters with high computation complexity,

\textsuperscript{13}It is different to the result reported by Wen et al. [23] due to the different fine-tuning frameworks and deep learning libraries.
and 1

Nesterov’s optimization to overrelax variables (i.e., structured convergence for structured filter sparsity is due to that we apply but requires 11 epochs and 20% parameters. This faster con-

Conv1 layer, while ADMM achieves almost the same error 0.9% error with only 8 epochs and 7% parameters in the epochs compared with ADMM. For instance, AULM achieves

pact network with almost the same error using much fewer

we found that our AULM can help to learn a more com-

Fig. 5. Further analysis of the $\ell_{2,1}$-regularization by our AULM and ADMM on the MNIST data set. (a) Epoch/# of parameters. (b) Epoch/Top-1 error.

TABLE VI
PRUNING RESULTS OF GOOGLENET. THE TEST MINI-BATCH SIZE IS 24

<table>
<thead>
<tr>
<th>Method</th>
<th>FLOPs</th>
<th>#Param.</th>
<th>Speedup</th>
<th>Top-1 Err. ↑</th>
<th>Top-5 Err. ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>1.0B</td>
<td>4.2M</td>
<td>CPU 1.5×</td>
<td>1.4×</td>
<td>4.04%</td>
</tr>
<tr>
<td>L1 [20]</td>
<td>1.0B</td>
<td>4.2M</td>
<td>CPU 1.5×</td>
<td>1.4×</td>
<td>3.00%</td>
</tr>
<tr>
<td>AFOZ [45]</td>
<td>1.0B</td>
<td>4.2M</td>
<td>CPU 1.5×</td>
<td>1.4×</td>
<td>2.30%</td>
</tr>
<tr>
<td>SSR-L2.1</td>
<td>1.0B</td>
<td>4.2M</td>
<td>CPU 1×</td>
<td>1.4×</td>
<td>1.90%</td>
</tr>
<tr>
<td>SSR-L2.0</td>
<td>1.0B</td>
<td>4.1M</td>
<td>CPU 1×</td>
<td>1.6×</td>
<td>1.81%</td>
</tr>
</tbody>
</table>

i.e., filter size of 3 $\times$ 3 and 5 $\times$ 5. For $\lambda$, it is set to be 0.5 and 0.3 in the first three inception blocks and the remaining inception blocks, respectively. We skip the first convolutional layer and kernels with a size of 1 $\times$ 1 for effective pruning. As shown in Table VI, compared with three alternative criteria of filter selection, SSR-L2.1 achieves a lower error increase at the same pruning level. By replacing $\ell_{2,1}$-regularization with $\ell_{2,0}$-regularization, SSR-L2.0 achieves the best performance with an increase of 1.05% Top-5 error by 1.7× compression and 1.6× CPU speedup.

D. Analysis

1) Efficiency Analysis: We first analyze the empirical efficiency of AULM and ADMM. As shown in Fig. 5, we found that our AULM can help to learn a more compact network with almost the same error using much fewer epochs compared with ADMM. For instance, AULM achieves 0.9% error with only 8 epochs and 7% parameters in the Conv1 layer, while ADMM achieves almost the same error but requires 11 epochs and 20% parameters. This faster convergence for structured filter sparsity is due to that we apply Nesterov’s optimization to overrelax variables (i.e., structured sparse filters and dual variables) for accelerating the alternative optimization. Then, we further study the influence of different optimization strategies (i.e., SSL versus AULM) on filter pruning. In SSL [23], the structured sparsity of filters with $\ell_{2,1}$-norm is learned by directly solving (2) with SGD, which is different from the proposed AULM-L2.1. Taking the first three layers of AlexNet, for instance, they occupy a significant proportion of the computational overhead.

Fig. 6 presents the convergence process of different solvers over different layers in Conv1, Conv2, and Conv3, respectively. Compared with SSL, as shown in Fig. 6(a), AULM-L2.1 is faster to reduce the training loss and also achieves a lower training loss, which leads to more effective training for structured pruning. Moreover, by using AULM-L2.1 instead of SSL, the number of pruned filters is always larger, the corresponding classification error is lower, and the convergence is faster, especially in the first convolutional layer (the convergence after 8 epochs in AULM-L2.1 versus 20 epochs in SSL). Therefore, alternative optimization in AULM is more effective to prune the network than SGD in SSL. Furthermore, we also make a comparison between two different structured regularizers (i.e., $\ell_{2,1}$-norm and $\ell_{2,0}$-norm) to explicitly analyze their convergence by AULM.14 As shown in Fig. 6(b) and (c), compared with AULM-L2.1, we observe that AULM-L2.0 is not only significantly faster to generate more structured filters but also achieves a lower Top-5 error. Interestingly, the number of structured filters is almost constant during training, which is due to the closed-form solution by (15) that leads to almost the same structured sparsity of intermediate filters after the first updating.

2) Visualization: To verify the effectiveness of structured filter sparsity, we visualize the filters in the first convolutional layer of AlexNet using SSR with three different regularizers (i.e., $\ell_1$-norm, $\ell_{2,1}$-norm, and $\ell_{2,0}$-norm), as shown in Fig. 7. Although SSR with $\ell_1$-regularization obtains a large number of sparse filter elements, it is unable to remove the whole filter, which leads to a very limited speedup without the specialized software. In contrast, SSR with $\ell_{2,1}$-regularization or $\ell_{2,0}$-regularization results in complete filter removal, which directly accelerates the network inference. Compared with $\ell_{2,1}$-regularization, SSR with $\ell_{2,0}$-regularization achieves a lower Top-5 error with more structured sparse filters. This is due to that the $\ell_{2,0}$-regularization explicitly provides the natural constraint for filter selection.

E. Generalization Ability for Transfer Learning

SSR has demonstrated its effectiveness to simultaneously accelerate and compress CNNs on MNIST and ImageNet 2012 classification tasks. We further investigate the generalization ability of the compressed model in transfer learning.

14We have not compared the SGD with our AULM on $\ell_{2,0}$-norm regularization since SGD cannot solve the NP-hard problem.
including domain adaptation and object detection. For a better discussion, we take VGG-16 as our baseline model.

1) Domain Adaptation: Since SSR does not change the network structure, a model pruned on ImageNet can be easily transferred into other domains. To evaluate the domain adaptation ability of the compressed model, we consider a practical application in which we transfer the pruned model trained on ImageNet into a smaller one by using a domain-specific data set. To this end, we select the public domain-specific data set CUB-200 [71] for fine-grained classification to evaluate the ability of domain adaptation. CUB-200 contains 11,788 images of 200 different bird species, which contains 5994 training images and 5794 testing images. To make a fair comparison, we fine-tune the compressed models on ImageNet by Random, L1, APoZ, and SSR with the same hyperparameters and epochs.15 The results of the fine-grained classification are shown in Table VII.

The pretrained VGG-16 is first fine-tuned on the CUB-200 data set, which is an effective method to directly transfer the model from ImageNet domain to CUB-200 domain. As shown in Table VII, the pretrained VGG-16 achieves the lowest error (27.60% Top-1 error) but has a huge memory cost and slow inference speed (i.e., 124.6M parameters and 15.5B FLOPs). We then fine-tune the compressed networks, which are previously compressed in the ImageNet domain by Random, L1, APoZ, and SSR, respectively. Compared with Random, L1, and APoZ, the model compressed by SSR-L2,1 achieves the best performance by an increase of only 1.1% Top-1 error, with 124.6M parameters and 4.5B FLOPs. Furthermore, we also fine-tune the compressed models, in which the traditional fully connected layers are replaced with GAP. We obtain a more compact model with 8.8M parameters and 4.4B FLOPs, i.e., 15.4 × lower memory cost and theoretical 3.5 × inference speedup than VGG-16. Compared with the three alternative selection criteria, SSR-L2,1-GAP achieves the lowest Top-1 error, i.e., 29.55% Top-1 error, which is an increase of only 1.95% Top-1 error.

2) Object Detection: We also evaluate the ability of transfer learning for the compressed VGG-16 by Random, L1 [20], APoZ [45], and SSR with $\ell_{2,1}$-regularization, which are deployed over faster R-CNN [11] for object detections. The PASCAL VOC 2007 object detection benchmark is selected

---

15 The related implementation details are described in the following: https://github.com/Roll920/fine-tune-avg-vgg16.
to evaluate the performance of our models by mAP, which contains about 5K training/validation images and 5K testing images. In our experiments, we first compress VGG-16 by Random, L1, APOZ, and SSR-L2, 1 on ImageNet and then use the compressed models as the pretrained models for faster R-CNN with the default training settings.

The actual running time of faster R-CNN is 189 ms/image on Titan X GPU. Compared with VGG-16, we get an actual detection time of 77 ms with 2.45× acceleration on Titan X. As shown in Table VIII, interestingly, filter pruning by random criterion works interestingly well for object detection, which is due to the self-recovery ability in training the specific PASCAL VOC 2007 data set. On the contrary, APOZ may be unsuitable for object detection, which achieves the lowest mAP, i.e., 1.7% mAP drops. Compared with three alternative pruning criteria, SSR-L2, 1 achieves the best performance by a factor of 2.45× speedup on Titan X with only 0.3% mAP drops, which is still very practical for real-world application.

V. CONCLUSION

In this paper, we propose a unified filter pruning scheme, termed SSR, for CNN acceleration and compression. SSR captures the relationship between the global output and local filter pruning, forming a novel optimization problem based on the two structured sparsities with $\ell_{2,1}$-norm and $\ell_{2,0}$-norm, both of which can be efficiently solved by a novel AULM. The proposed AULM is fast to generate structured filters and is adaptive to prune redundant filters. We have demonstrated that the proposed SSR scheme achieves superior performance over the state-of-the-art filter pruning methods \cite{19, 20, 23, 42, 45}. We further evaluate the effectiveness of the compressed model by SSR when being applied to domain adaptation and object detection.

In the future, we would like to investigate the specific design of filter pruning for ResNet and DenseNet, including: 1) how to effectively prune the shortcut connection in the residual block; 2) how to design a more effective strategy for accelerating batch normalization and pooling layers, which are left unexploited in the existing works; and 3) how to design a novel filter selection layer to prevent the dimension mismatch of different dense blocks, which is due to the dense connectivity in the natural DenseNet.

REFERENCES


Yuchao Li received the B.E. degree from Xiamen University, Xiamen, China, in 2017, where he is currently pursuing the degree in information and computing science with the Group of Media Analytics and Computing. His current research interests include deep learning and computer vision.

Cheng Deng (S’09–M’11) received the B.E., M.S., and Ph.D. degrees in signal and information processing from Xidian University, Xi’an, China. He is currently a Full Professor with the School of Electronic Engineering, Xidian University. He has authored or co-authored more than 50 scientific articles at top venues, including the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, the IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE TRANSACTIONS ON CYBERNETICS, the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, the IEEE TRANSACTIONS ON IMAGE PROCESSING, the IEEE International Conference on Computer Vision, the IEEE Conference on Computer Vision and Pattern Recognition, the International Joint Conferences on Artificial Intelligence, and the Association for the Advancement of Artificial Intelligence. His current research interests include computer vision, multimedia processing and analysis, and information hiding.

Xuelong Li (M’02–SM’07–F’12) is currently a Full Professor with the School of Computer Science and the Center for OPTical IMagery Analysis and Learning (OPTIMAL), Northwestern Polytechnical University, Xi’an, China. Dr. Li is a fellow of the SPIE, the Institution of Engineering and Technology, and the International Association for Pattern Recognition.