Learning Structured Sparsity in Deep Neural Networks

Wei Wen
University of Pittsburgh
wew57@pitt.edu

Chunpeng Wu
University of Pittsburgh
chw127@pitt.edu

Yandan Wang
University of Pittsburgh
yaw46@pitt.edu

Yiran Chen
University of Pittsburgh
yic52@pitt.edu

Hai Li
University of Pittsburgh
hal66@pitt.edu

Abstract

High demand for computation resources severely hinders deployment of large-scale *Deep Neural Networks* (DNN) in resource constrained devices. In this work, we propose a *Structured Sparsity Learning* (SSL) method to regularize the structures (*i.e.*, filters, channels, filter shapes, and layer depth) of DNNs. SSL can: (1) learn a compact structure from a bigger DNN to reduce computation cost; (2) obtain a hardware-friendly structured sparsity of DNN to efficiently accelerate the DNN's evaluation. Experimental results show that SSL achieves on average 5.1$\times$ and 3.1$\times$ speedups of convolutional layer computation of *AlexNet* against CPU and GPU, respectively, with off-the-shelf libraries. These speedups are about twice speedups of non-structured sparsity; (3) regularize the DNN structure to improve classification accuracy. The results show that for CIFAR-10, regularization on layer depth reduces a 20-layer Deep Residual Network (*ResNet*) to 18 layers while improves the accuracy from 91.25% to 92.60%, which is still higher than that of original *ResNet* with 32 layers. For *AlexNet*, SSL reduces the error by $\sim 1\%$.

1 Introduction

Deep neural networks (DNN), especially deep *Convolutional Neural Networks* (CNN), made remarkable success in visual tasks [1][2][3][4][5] by leveraging large-scale networks learning from a huge volume of data. Deployment of such big models, however, is computation-intensive. To reduce computation, many studies are performed to compress the scale of DNN, including sparsity regularization [6], connection pruning [7][8] and low rank approximation [9][10][11][12][13]. Sparsity regularization and connection pruning, however, often produce non-structured random connectivity and thus, irregular memory access that adversely impacts *practical* acceleration in hardware platforms. Figure 1 depicts practical layer-wise speedup of *AlexNet*, which is non-structurally sparsified by ℓ_1-norm. Compared to original model, the accuracy loss of the sparsified model is controlled within 2%. Because of the poor data locality associated with the scattered weight distribution, the achieved speedups are either very limited or negative even the actual sparsity is high, say, >95%. We define sparsity as the ratio of zeros in this paper. In recently proposed low rank approximation approaches, the DNN is trained first and then each trained weight tensor is decomposed and approximated by a product of smaller factors. Finally, fine-tuning is performed to restore the model accuracy. Low rank approximation is able to achieve practical speedups because it coordinates model parameters in dense matrixes and avoids the locality problem of non-structured sparsity regularization. However, low rank approximation can only obtain the compact structure within each layer, and the structures of the layers are fixed during fine-tuning such that costly reiterations of decomposing and fine-tuning are required to find an optimal weight approximation for performance speedup and accuracy retaining.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

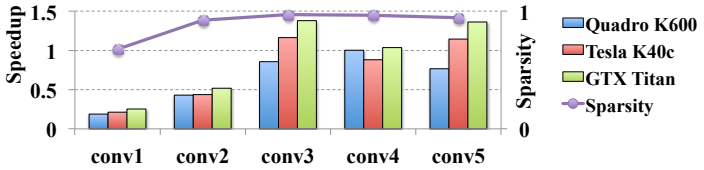


Figure 1: Evaluation speedups of AlexNet on GPU platforms and the sparsity. conv1 refers to convolutional layer 1, and so forth. Baseline is profiled by GEMM of cuBLAS. The sparse matrixes are stored in the format of Compressed Sparse Row (CSR) and accelerated by cuSPARSE.

Inspired by the facts that (1) there is redundancy across filters and channels [11]; (2) shapes of filters are usually fixed as cuboid but enabling arbitrary shapes can potentially eliminate unnecessary computation imposed by this fixation; and (3) depth of the network is critical for classification but deeper layers cannot always guarantee a lower error because of the exploding gradients and degradation problem [5], we propose *Structured Sparsity Learning* (SSL) method to *directly* learn a compressed structure of deep CNNs by group Lasso regularization during the training. SSL is a generic regularization to adaptively adjust multiple structures in DNN, including structures of filters, channels, filter shapes within each layer, and structure of depth beyond the layers. SSL combines structure regularization (on DNN for classification accuracy) with locality optimization (on memory access for computation efficiency), offering not only well-regularized big models with improved accuracy but greatly accelerated computation (*e.g.*, 5.1× on CPU and 3.1× on GPU for *AlexNet*). Our source code can be found at `https://github.com/wenwei202/caffe/tree/scnn`.

2 Related works

Connection pruning and weight sparsifying. Han *et al.* [7][8] reduced parameters of *AlexNet* and *VGG-16* using connection pruning. Since most reduction is achieved on fully-connected layers, no practical speedups of convolutional layers are observed for the similar issue shown in Figure 1. However, convolution is more costly and many new DNNs use fewer fully-connected layers, *e.g.*, only 3.99% parameters of *ResNet-152* [5] are from fully-connected layers, compression and acceleration on convolutional layers become essential. Liu *et al.* [6] achieved >90% sparsity of convolutional layers in *AlexNet* with 2% accuracy loss, and bypassed the issue of Figure 1 by hardcoding the sparse weights into program. In this work, we also focus on convolutional layers. Compared to the previous techniques, our method coordinates sparse weights in adjacent memory space and achieve higher speedups. Note that hardware and program optimizations based on our method can further boost the system performance which is not covered in this paper due to space limit.

Low rank approximation. Denil *et al.* [9] predicted 95% parameters in a DNN by exploiting the redundancy across filters and channels. Inspired by it, Jaderberg *et al.* [11] achieved 4.5× speedup on CPUs for scene text character recognition and Denton *et al.* [10] achieved 2× speedups for the first two layers in a larger DNN. Both of the works used *Low Rank Approximation* (LRA) with ∼1% accuracy drop. [13][12] improved and extended LRA to larger DNNs. However, the network structure compressed by LRA is fixed; reiterations of decomposing, training/fine-tuning, and cross-validating are still needed to find an optimal structure for accuracy and speed trade-off. As the number of hyper-parameters in LRA method increases linearly with the layer depth [10][13], the search space increases linearly or even exponentially. Comparing to LRA, our contributions are: (1) SSL can dynamically optimize the compactness of DNNs with only one hyper-parameter and no reiterations; (2) besides the redundancy within the layers, SSL also exploits the necessity of deep layers and reduce them; (3) DNN filters regularized by SSL have *lower* rank approximation, so it can work together with LRA for more efficient model compression.

Model structure learning. Group Lasso [14] is an efficient regularization to learn sparse structures. Liu *et al.* [6] utilized group Lasso to constrain the structure scale of LRA. To adapt DNN structure to different databases, Feng *et al.* [16] learned the appropriate number of filters in DNN. Different from prior arts, we apply group Lasso to regularize multiple DNN structures (filters, channels, filter shapes, and layer depth). A most related parallel work is Group-wise Brain Damage [17], which is a subset (*i.e.*, learning filter shapes) of our work and further justifies the effectiveness of our techniques.

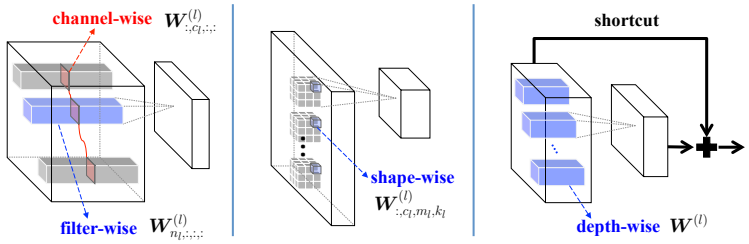


Figure 2: The proposed *Structured Sparsity Learning* (SSL) for DNNs. The weights in filters are split into multiple groups. Through group Lasso regularization, a more compact DNN is obtained by removing some groups. The figure illustrates the filter-wise, channel-wise, shape-wise, and depth-wise structured sparsity that are explored in the work.

3 Structured Sparsity Learning Method for DNNs

We focus mainly on the *Structured Sparsity Learning* (SSL) on convolutional layers to regularize the structure of DNNs. We first propose a generic method to regularize structures of DNN in Section 3.1, and then specify the method to structures of filters, channels, filter shapes and depth in Section 3.2. Variants of formulations are also discussed from computational efficiency viewpoint in Section 3.3.

3.1 Proposed structured sparsity learning for generic structures

Suppose the weights of convolutional layers in a DNN form a sequence of 4-D tensors $\boldsymbol{W}^{(l)} \in \mathbb{R}^{N_l \times C_l \times M_l \times K_l}$, where N_l, C_l, M_l and K_l are the dimensions of the l-*th* ($1 \leq l \leq L$) weight tensor along the axes of filter, channel, spatial height and spatial width, respectively. L denotes the number of convolutional layers. Then the proposed generic optimization target of a DNN with structured sparsity regularization can be formulated as:

$$E(\boldsymbol{W}) = E_D(\boldsymbol{W}) + \lambda \cdot R(\boldsymbol{W}) + \lambda_g \cdot \sum_{l=1}^{L} R_g\left(\boldsymbol{W}^{(l)}\right). \tag{1}$$

Here $\boldsymbol{W}$ represents the collection of all weights in the DNN; $E_D(\boldsymbol{W})$ is the loss on data; $R(\cdot)$ is non-structured regularization applying on every weight, *e.g.*, ℓ_2-norm; and $R_g(\cdot)$ is the structured sparsity regularization on each layer. Because *group Lasso* can effectively zero out all weights in some groups [14][15], we adopt it in our SSL. The regularization of group Lasso on a set of weights $\boldsymbol{w}$ can be represented as $R_g(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g$, where $\boldsymbol{w}^{(g)}$ is a group of partial weights in $\boldsymbol{w}$ and G is the total number of groups. Different groups may overlap. Here $||\cdot||_g$ is the group Lasso, or $||\boldsymbol{w}^{(g)}||_g = \sqrt{\sum_{i=1}^{|\boldsymbol{w}^{(g)}|} \left(w_i^{(g)}\right)^2}$, where $|\boldsymbol{w}^{(g)}|$ is the number of weights in $\boldsymbol{w}^{(g)}$.

3.2 Structured sparsity learning for structures of filters, channels, filter shapes and depth

In SSL, the learned "structure" is decided by the way of splitting groups of $\boldsymbol{w}^{(g)}$. We investigate and formulate the *filer-wise*, *channel-wise*, *shape-wise*, and *depth-wise* structured sparsity in Figure 2. For simplicity, the $R(\cdot)$ term of Eq. (1) is omitted in the following formulation expressions.

Penalizing unimportant filers and channels. Suppose $\boldsymbol{W}^{(l)}_{n_l,:,:,:}$ is the n_l-*th* filter and $\boldsymbol{W}^{(l)}_{:,c_l,:,:}$ is the c_l-*th* channel of all filters in the l-*th* layer. The optimization target of learning the filter-wise and channel-wise structured sparsity can be defined as

$$E(\boldsymbol{W}) = E_D(\boldsymbol{W}) + \lambda_n \cdot \sum_{l=1}^{L}\left(\sum_{n_l=1}^{N_l} ||\boldsymbol{W}^{(l)}_{n_l,:,:,:}||_g\right) + \lambda_c \cdot \sum_{l=1}^{L}\left(\sum_{c_l=1}^{C_l} ||\boldsymbol{W}^{(l)}_{:,c_l,:,:}||_g\right). \tag{2}$$

As indicated in Eq. (2), our approach tends to remove less important filters and channels. Note that zeroing out a filter in the l-*th* layer results in a dummy zero output feature map, which in turn makes a corresponding channel in the $(l+1)$-*th* layer useless. Hence, we combine the filter-wise and channel-wise structured sparsity in the learning simultaneously.

Learning arbitrary shapes of filers. As illustrated in Figure 2, $\boldsymbol{W}^{(l)}_{:,c_l,m_l,k_l}$ denotes the vector of all corresponding weights located at spatial position of (m_l, k_l) in the 2D filters across the c_l*-th* channel. Thus, we define $\boldsymbol{W}^{(l)}_{:,c_l,m_l,k_l}$ as the *shape fiber* related to learning arbitrary filter shape because a homogeneous non-cubic filter shape can be learned by zeroing out some shape fibers. The optimization target of learning shapes of filers becomes:

$$E(\boldsymbol{W}) = E_D(\boldsymbol{W}) + \lambda_s \cdot \sum_{l=1}^{L} \left(\sum_{c_l=1}^{C_l} \sum_{m_l=1}^{M_l} \sum_{k_l=1}^{K_l} ||\boldsymbol{W}^{(l)}_{:,c_l,m_l,k_l}||_g \right). \tag{3}$$

Regularizing layer depth. We also explore the depth-wise sparsity to regularize the depth of DNNs in order to improve accuracy and reduce computation cost. The corresponding optimization target is $E(\boldsymbol{W}) = E_D(\boldsymbol{W}) + \lambda_d \cdot \sum_{l=1}^{L} ||\boldsymbol{W}^{(l)}||_g$. Different from other discussed sparsification techniques, zeroing out all the filters in a layer will cut off the message propagation in the DNN so that the output neurons cannot perform any classification. Inspired by the structure of highway networks [18] and deep residual networks [5], we propose to leverage the shortcuts across layers to solve this issue. As illustrated in Figure 2, even when SSL removes an entire unimportant layers, feature maps will still be forwarded through the shortcut.

3.3 Structured sparsity learning for computationally efficient structures

All proposed schemes in section 3.2 can learn a compact DNN for computation cost reduction. Moreover, some variants of the formulations of these schemes can directly learn structures that can be efficiently computed.

2D-filter-wise sparsity for convolution. 3D convolution in DNNs essentially is a composition of 2D convolutions. To perform efficient convolution, we explored a fine-grain variant of filter-wise sparsity, namely, *2D-filter-wise* sparsity, to spatially enforce group Lasso on each 2D filter of $\boldsymbol{W}^{(l)}_{n_l,c_l,:,:}$. The saved convolution is proportional to the percentage of the removed 2D filters. The fine-grain version of filter-wise sparsity can more efficiently reduce the computation associated with convolution: Because the distance of weights (in a smaller group) from the origin is shorter, which makes group Lasso more easily to obtain a higher ratio of zero groups.

Combination of filter-wise and shape-wise sparsity for GEMM. Convolutional computation in DNNs is commonly converted to modality of *GEneral Matrix Multiplication* (GEMM) by lowering weight tensors and feature tensors to matrices [19]. For example, in Caffe [20], a 3D filter $\boldsymbol{W}^{(l)}_{n_l,:,:,:}$ is reshaped to a row in the weight matrix where each column is the collection of weights $\boldsymbol{W}^{(l)}_{:,c_l,m_l,k_l}$ related to shape-wise sparsity. Combining filter-wise and shape-wise sparsity can directly reduce the dimension of weight matrix in GEMM by removing zero rows and columns. In this context, we use *row-wise* and *column-wise* sparsity as the interchangeable terminology of *filter-wise* and *shape-wise sparsity*, respectively.

4 Experiments

We evaluate the effectiveness of our SSL using published models on three databases – MNIST, CIFAR-10, and ImageNet. Without explicit explanation, SSL starts with the network whose weights are initialized by the baseline, and speedups are measured in matrix-matrix multiplication by Caffe in a single-thread Intel Xeon E5-2630 CPU. Hyper-parameters are selected by cross-validation.

4.1 *LeNet* and multilayer perceptron on MNIST

In the experiment of MNIST, we examine the effectiveness of SSL in two types of networks: *LeNet* [21] implemented by Caffe and a *multilayer perceptron* (*MLP*) network. Both networks were trained without data augmentation.

***LeNet*:** When applying SSL to *LeNet*, we constrain the network with filter-wise and channel-wise sparsity in convolutional layers to penalize unimportant filters and channels. Table 1 summarizes the remained filters and channels, *floating-point operations* (FLOP), and practical speedups. In the table, *LeNet 1* is the baseline and the others are the results after applying SSL in different strengths

Table 1: Results after penalizing unimportant filters and channels in *LeNet*

LeNet #	Error	Filter # [§]	Channel # [§]	FLOP [§]	Speedup [§]
1 (*baseline*)	0.9%	20—50	1—20	100%—100%	1.00×—1.00×
2	0.8%	5—19	1—4	25%—7.6%	1.64×—5.23×
3	1.0%	3—12	1—3	15%—3.6%	1.99×—7.44×

[§]In the order of *conv1*—*conv2*

Table 2: Results after learning filter shapes in *LeNet*

LeNet #	Error	Filter size [§]	Channel #	FLOP	Speedup
1 (*baseline*)	0.9%	25—500	1—20	100%—100%	1.00×—1.00×
4	0.8%	21—41	1—2	8.4%—8.2%	2.33×—6.93×
5	1.0%	7—14	1—1	1.4%—2.8%	5.19×—10.82×

[§] The sizes of filters after removing zero shape fibers, in the order of *conv1*—*conv2*

of structured sparsity regularization. The results show that our method achieves the similar error ($\pm 0.1\%$) with much fewer filters and channels, and saves significant FLOP and computation time.

To demonstrate the impact of SSL on the structures of filters, we present all learned *conv1* filters in Figure 3. It can be seen that most filters in *LeNet 2* are entirely zeroed out except for five most important detectors of stroke patterns that are sufficient for feature extraction. The accuracy of *LeNet 3* (that further removes the weakest and redundant stroke detector) drops only 0.2% from that of *LeNet 2*. Compared to the random and blurry filter patterns in *LeNet 1* which are resulted from the high freedom of parameter space, the filters in *LeNet 2 & 3* are regularized and converge to smoother and more natural patterns. This explains why our proposed SSL obtains the same-level accuracy but has much less filters. The smoothness of the filters are also observed in the deeper layers.

The effectiveness of the shape-wise sparsity on *LeNet* is summarized in Table 2. The baseline *LeNet 1* has *conv1* filters with a regular 5×5 square (size = 25) while *LeNet 5* reduces the dimension that can be constrained by a 2×4 rectangle (size = 7). The 3D shape of *conv2* filters in the baseline is also regularized to the 2D shape in *LeNet 5* within only one channel, indicating that only one filter in *conv1* is needed. This fact significantly saves FLOP and computation time.

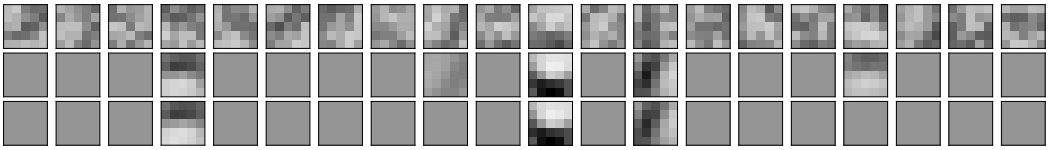


Figure 3: Learned *conv1* filters in *LeNet 1* (top), *LeNet 2* (middle) and *LeNet 3* (bottom)

***MLP*:** Besides convolutional layers, our proposed SSL can be extended to learn the structure (*i.e.*, the number of neurons) of fully-connected layers. We enforce the group Lasso regularization on all the input (or output) connections of each neuron. A neuron whose input connections are all zeroed out can degenerate to a bias neuron in the next layer; similarly, a neuron can degenerate to a removable dummy neuron if all of its output connections are zeroed out. Figure 4(a) summarizes the learned structure and FLOP of different *MLP* networks. The results show that SSL can not only remove hidden neurons but also discover the sparsity of images. For example, Figure 4(b) depicts the number of connections of each input neuron in *MLP 2*, where 40.18% of input neurons have zero connections and they concentrate at the boundary of the image. Such a distribution is consistent with our intuition: handwriting digits are usually written in the center and pixels close to the boundary contain little discriminative classification information.

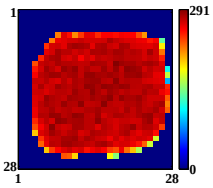
4.2 *ConvNet* and *ResNet* on CIFAR-10

We implemented the *ConvNet* of [1] and *deep residual networks* (*ResNet*) [5] on CIFAR-10. When regularizing filters, channels, and filter shapes, the results and observations of both networks are similar to that of the MNIST experiment. Moreover, we simultaneously learn the filter-wise and shape-wise sparsity to reduce the dimension of weight matrix in GEMM by *ConvNet*. We also learn the depth-wise sparsity of *ResNet* to regularize the depth of the DNNs.

MLP #	Error	Neuron # per layer [§]	FLOP per layer [§]
1 (*baseline*)	1.43%	784–500–300–10	100%–100%–100%
2	1.34%	469–294–166–10	35.18%–32.54%–55.33%
3	1.53%	434–174–78–10	19.26%–9.05%–26.00%

[§] In the order of *input layer–hidden layer 1–hidden layer 2–output layer*

(a)



(b)

Figure 4: (a) Results of learning the number of neurons in *MLP*. (b) the connection numbers of input neurons (*i.e.*, pixels) in *MLP 2* after SSL.

Table 3: Learning row-wise and column-wise sparsity of *ConvNet* on CIFAR-10

ConvNet #	Error	Row sparsity [§]	Column sparsity [§]	Speedup [§]
1 (*baseline*)	17.9%	12.5%–0%–0%	0%–0%–0%	1.00×–1.00×–1.00×
2	17.9%	50.0%–28.1%–1.6%	0%–59.3%–35.1%	1.43×–3.05×–1.57×
3	16.9%	31.3%–0%–1.6%	0%–42.8%–9.8%	1.25×–2.01×–1.18×

[§] in the order of *conv1–conv2–conv3*

***ConvNet*:** We use the network from Alex Krizhevsky *et al.* [1] as the baseline and implement it using Caffe. All the configurations remain the same as the original implementation except that we added a dropout layer with a ratio of 0.5 in the fully-connected layer to avoid over-fitting. *ConvNet* is trained without data augmentation. Table 3 summarizes the results of three *ConvNet* networks. Here, the row/column sparsity of a weight matrix is defined as the percentage of all-zero rows/columns. Figure 5 shows their learned *conv1* filters. In Table 3, SSL can reduce the size of weight matrix in *ConvNet 2* by 50%, 70.7% and 36.1% for each convolutional layer and achieve good speedups without accuracy drop. Surprisingly, without SSL, four *conv1* filters of the baseline are actually all-zeros as shown in Figure 5, demonstrating the great potential of filter sparsity. When SSL is applied, half of *conv1* filters in *ConvNet 2* can be zeroed out without accuracy drop.

On the other hand, in *ConvNet 3*, SSL lowers 1.0% ($\pm$0.16%) error with a model even smaller than the baseline. In this scenario, SSL performs as a structure regularization to dynamically learn a better network structure (including the number of filters and filer shapes) to reduce the error.

***ResNet*:** To investigate the necessary depth of DNNs by SSL, we use a 20-layer deep residual networks (*ResNet-20*) [5] as the baseline. The network has 19 convolutional layers and 1 fully-connected layer. *Identity shortcuts* are utilized to connect the feature maps with the same dimension while 1×1 convolutional layers are chosen as shortcuts between the feature maps with different dimensions. Batch normalization [22] is adopted after convolution and before activation. We use the same data augmentation and training hyper-parameters as that in [5]. The final error of baseline is 8.82%. In SSL, the depth of *ResNet-20* is regularized by depth-wise sparsity. Group Lasso regularization is only enforced on the convolutional layers between each pair of shortcut endpoints, excluding the first convolutional layer and all convolutional shortcuts. After SSL converges, layers with all zero weights are removed and the net is finally fine-tuned with a base learning rate of 0.01, which is lower than that (*i.e.*, 0.1) in the baseline.

Figure 6 plots the trend of the error vs. the number of layers under different strengths of depth regularizations. Compared with original *ResNet* in [5], SSL learns a *ResNet* with 14 layers (*SSL-ResNet-14*) reaching a lower error than that of the baseline with 20 layers (*ResNet-20*); *SSL-ResNet-18* and *ResNet-32* achieve an error of 7.40% and 7.51%, respectively. This result implies that SSL can work as a depth regularization to improve classification accuracy. Note that SSL can efficiently learn shallower DNNs without accuracy loss to reduce computation cost; however, it does not mean the depth of the network is not important. The trend in Figure 6 shows that the test error generally declines as more layers are preserved. A slight error rise of *SSL-ResNet-20* from *SSL-ResNet-18* shows the suboptimal selection of the depth in the group of "32×32".

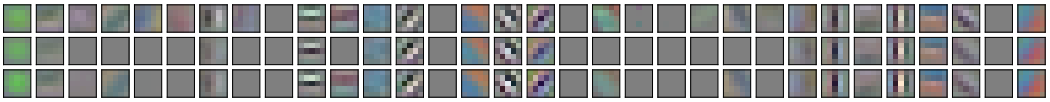


Figure 5: Learned *conv1* filters in *ConvNet 1* (top), *ConvNet 2* (middle) and *ConvNet 3* (bottom)

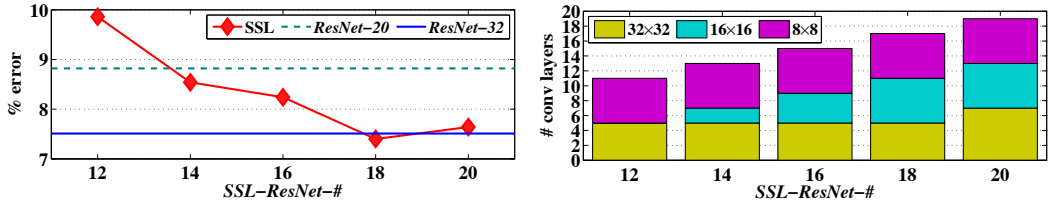


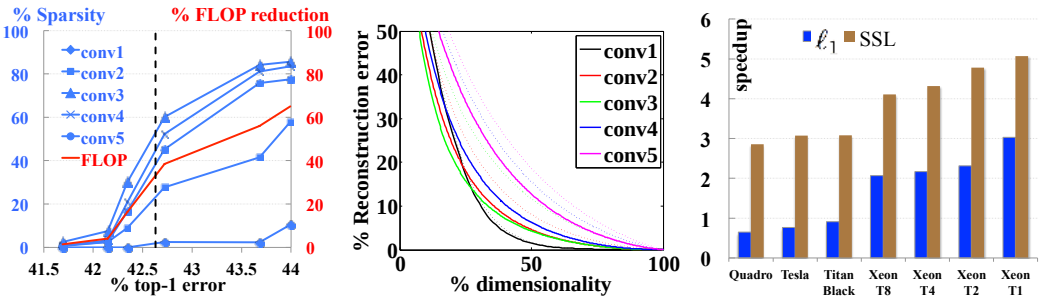
Figure 6: Error vs. layer number after depth regularization. # is the number of layers including the last fully-connected layer. *ResNet-#* is the *ResNet* in [5]. *SSL-ResNet-#* is the depth-regularized *ResNet* by SSL. 32×32 indicates the convolutional layers with an output map size of 32×32, *etc.*

4.3 *AlexNet* on ImageNet

To show the generalization of our method to large scale DNNs, we evaluate SSL using *AlexNet* with ILSVRC 2012. *CaffeNet* [20], the replication of *AlexNet* [1] with mirror changes, is used in our experiment. All training images are rescaled to the size of 256×256. A 227×227 image is randomly cropped from each scaled image and mirrored for data augmentation and only the center crop is used for validation. The final top-1 validation error is 42.63%. In SSL, *AlexNet* is first trained with structure regularization; when it converges, zero groups are removed to obtain a DNN with the new structure; finally, the network is fine-tuned without SSL to regain the accuracy.

We first study 2D-filter-wise and shape-wise sparsity by exploring the trade-offs between computation complexity and classification accuracy. Figure 7(a) shows the 2D-filter sparsity (the ratio between the removed 2D filters and total 2D filters) and the saved FLOP of 2D convolutions vs. the validation error. In Figure 7(a), deeper layers generally have higher sparsity as the group size shrinks and the number of 2D filters grows. 2D-filter sparsity regularization can reduce the total FLOP by 30%–40% without accuracy loss or reduce the error of *AlexNet* by ∼1% down to 41.69% by retaining the original number of parameters. Shape-wise sparsity also obtains similar results. In Table 4, for example, *AlexNet 5* achieves on average 1.4× layer-wise speedup on both CPU and GPU without accuracy loss after shape regularization; The top-1 error can also be reduced down to 41.83% if the parameters are retained. In Figure 7(a), the obtained DNN with the lowest error has a very low sparsity, indicating that the number of parameters in a DNN is still important to maintain learning capacity. In this case, SSL works as a regularization to add restriction of smoothness to the model in order to avoid over-fitting. Figure 7(b) compares the results of dimensionality reduction of weight tensors in the baseline and our SSL-regularized *AlexNet*. The results show that the smoothness restriction enforces parameter searching in lower-dimensional space and enables *lower* rank approximation of the DNNs. Therefore, SSL can work together with low rank approximation to achieve even higher model compression.

Besides the above analyses, the computation efficiencies of structured sparsity and non-structured sparsity are compared in Caffe using standard off-the-shelf libraries, *i.e.*, Intel Math Kernel Library



(a) (b) (c)

Figure 7: (a) 2D-filter-wise sparsity and FLOP reduction vs. top-1 error. Vertical dash line shows the error of original *AlexNet*; (b) The reconstruction error of weight tensor vs. dimensionality. *Principal Component Analysis* (PCA) is utilized to perform dimensionality reduction. The eigenvectors corresponding to the largest eigenvalues are selected as basis of lower-dimensional space. Dash lines denote the results of the baselines and solid lines indicate the ones of the *AlexNet 5* in Table 4; (c) Speedups of ℓ_1-norm and SSL on various CPUs and GPUs (In labels of x-axis, T# is the number of maximum physical threads in CPUs). *AlexNet 1* and *AlexNet 2* in Table 4 are used as testbenches.

on CPU and CUDA cuBLAS and cuSPARSE on GPU. We use SSL to learn a *AlexNet* with high column-wise and row-wise sparsity as the representative of structured sparsity method. ℓ_1-norm is selected as the representative of non-structured sparsity method instead of connection pruning [7] because ℓ_1-norm get a higher sparsity on convolutional layers as the results of *AlexNet 3* and *AlexNet 4* depicted in Table 4. Speedups achieved by SSL are measured by GEMM, where all-zero rows (and columns) in each weight matrix are removed and the remaining ones are concatenated in consecutive memory space. Note that compared to GEMM, the overhead of concatenation can be ignored. To measure the speedups of ℓ_1-norm, sparse weight matrices are stored in the format of Compressed Sparse Row (CSR) and computed by sparse-dense matrix multiplication subroutines.

Table 4 compares the obtained sparsity and speedups of ℓ_1-norm and SSL on CPU (Intel Xeon) and GPU (GeForce GTX TITAN Black) under approximately the same errors, *e.g.*, with acceptable or no accuracy loss. To make a fair comparison, after ℓ_1-norm regularization, the DNN is also fine-tuned by disconnecting all zero-weighted connections so that, *e.g.,* 1.39% accuracy is recovered for the *AlexNet 1*. Our experiments show that the DNNs require a very high non-structured sparsity to achieve a reasonable speedup (the speedups are even negative when the sparsity is low). SSL, however, can always achieve positive speedups. With an acceptable accuracy loss, our SSL achieves on average $5.1\times$ and $3.1\times$ layer-wise acceleration on CPU and GPU, respectively. Instead, ℓ_1-norm achieves on average only $3.0\times$ and $0.9\times$ layer-wise acceleration on CPU and GPU, respectively. We note that, at the same accuracy, our average speedup is indeed higher than that of [6] which adopts heavy hardware customization to overcome the negative impact of non-structured sparsity. Figure 7(c) shows the speedups of ℓ_1-norm and SSL on various platforms, including both GPU (Quadro, Tesla and Titan) and CPU (Intel Xeon E5-2630). SSL can achieve on average $\sim 3\times$ speedup on GPU while non-structured sparsity obtain no speedup on GPU platforms. On CPU platforms, both methods can achieve good speedups and the benefit grows as the processors become weaker. Nonetheless, SSL can always achieve averagely $\sim 2\times$ speedup compared to non-structured sparsity.

5 Conclusion

In this work, we propose a *Structured Sparsity Learning* (SSL) method to regularize filter, channel, filter shape, and depth structures in *Deep Neural Networks* (DNN). Our method can enforce the DNN to dynamically learn more compact structures without accuracy loss. The structured compactness of the DNN achieves significant speedups for the DNN evaluation both on CPU and GPU with off-the-shelf libraries. Moreover, a variant of SSL can be performed as structure regularization to improve classification accuracy of state-of-the-art DNNs.

Acknowledgments

This work was supported in part by NSF XPS-1337198 and NSF CCF-1615475. The authors thank Drs. Sheng Li and Jongsoo Park for valuable feedback on this work.

Table 4: Sparsity and speedup of *AlexNet* on ILSVRC 2012

#	Method	Top1 err.	Statistics	conv1	conv2	conv3	conv4	conv5
1	ℓ_1	44.67%	sparsity	67.6%	92.4%	97.2%	96.6%	94.3%
			CPU ×	0.80	2.91	4.84	3.83	2.76
			GPU ×	0.25	0.52	1.38	1.04	1.36
2	SSL	44.66%	column sparsity	0.0%	63.2%	76.9%	84.7%	80.7%
			row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%
			CPU ×	1.05	3.37	6.27	9.73	4.93
			GPU ×	1.00	2.37	4.94	4.03	3.05
3	pruning [7]	42.80%	sparsity	16.0%	62.0%	65.0%	63.0%	63.0%
4	ℓ_1	42.51%	sparsity	14.7%	76.2%	85.3%	81.5%	76.3%
			CPU ×	0.34	0.99	1.30	1.10	0.93
			GPU ×	0.08	0.17	0.42	0.30	0.32
5	SSL	42.53%	column sparsity	0.00%	20.9%	39.7%	39.7%	24.6%
			CPU ×	1.00	1.27	1.64	1.68	1.32
			GPU ×	1.00	1.25	1.63	1.72	1.36

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105. 2012.

[2] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2015.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[6] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[7] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143. 2015.

[8] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[9] Misha Denil, Babak Shakibi, Laurent Dinh, Marc' Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156. 2013.

[10] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277. 2014.

[11] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[12] Yani Ioannou, Duncan P. Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744*, 2015.

[13] Cheng Tai, Tong Xiao, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[14] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[15] Seyoung Kim and Eric P Xing. Tree-guided group lasso for multi-task regression with structured sparsity. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[16] Jiashi Feng and Trevor Darrell. Learning the structure of deep convolutional networks. In *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[17] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[18] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[19] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[20] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.