# How to Obtain and Run Light and Efficient Deep Learning Networks

Fan Chen, Wei Wen, Linghao Song, Jingchi Zhang, Hai "Helen" Li, Yiran Chen
Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina, USA
{fan.chen, wei.wen, linghao.song, jingchi.zhang, hai.li, yiran.chen}@duke.edu

***Abstract*—As the model size of deep neural networks (DNNs) grows for better performance, the increase in computational cost associated with training and testing makes it extremely difficulty to deploy DNNs on end/edge devices with limited resources while also satisfying the response time requirement. To address this challenge, model compression which compresses model size and thus reduces computation cost is widely adopted in deep learning society. However, the practical impacts of hardware design are often ignored in these algorithm-level solutions, such as the increase of the random accesses to memory hierarchy and the constraints of memory capacity. On the other side, limited understanding about the computational needs at algorithm level may lead to unrealistic assumptions during the hardware designs. In this work, we will discuss this mismatch and provide how our approach addresses it through an interactive design practice across both software and hardware levels.**

***Index Terms*—deep neural networks, model compression, hardware accelerator**

## I. Introduction

Deep neural networks (DNNs) have been widely used in many new application fields, such as computer vision [21], text processing [32] and natural language processing [5]. To achieve continuously accuracy improvement, complex models with ever-increasing model sizes are typically trained on a huge volume of training data. Deploying such big models in resource constrained devices, *e.g.*,, end devices with limited resources and strict response time, becomes extremely difficult due to intensive computation.

Model Compression [7], [10], [11], [14], [15], [23], [27], [29] is a class of approaches which significantly compresses the scale of DNNs and hence, reducing the computation required in reference and/or training. However, as we will elaborate in Section II, previous methods are subject to some restrictions. Among these limitations, the most severe problem is that they typically produce non-structured random connectivity and result in frequent access to off-chip DRAM.Off-chip memory access consumes two orders of magnitude more energy than a floating point operation [25] and quickly becomes the bottleneck which hinders system performance.

In this article, we review the current advance in DNN model compression and point point out their limitations in Section II. Then we introduce our solutions in Section III. Different from previous research, we propose a *Structured Sparsity Learning* (SSL) method which can directly learn a compact DNN structure, offering not only well-regularized big models with improved accuracy but also significantly optimization on memory access for computation efficiency. Especially, SSL presents 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.

From an implementation perspective, we have observed that SSL-based hardware implementation and program optimization can further improve system performance. We demonstrate this observation through two research cases in Section IV: *1)* an hardware accelerator for compressed DNNs, where DNN weights are compressed utilizing SSL. We particular proposed *Structurally-compressed Weight Oriented Fetching* and *In-layer Pipeline for Memory and Computation* to capture all the benefits of hardware-friendliness from SSL. *2)* an efficient structural sparsity learning framework for speech recognition inspired by the Intel® Gaussian Neural Accelerator (GNA) [6]. In our evaluation, both designs achieve significant performance improvement compared to prior arts.

## II. Prior Arts and Motivation

### A. Related Work on Model Compression

To reduce computation, many studies are performed to compress the scale of DNN, including *1)* Connection pruning and weight sparsifying, *2)* low rank approximation, and *3)* model structure learning. Below we introduce these three approaches and summarize their main features in Table I.

**Connection pruning and weight sparsifying.** Han *et al.* [10], [11] proposed to reduce parameters in fully-connected (FC) layers in *AlexNet* and *VGG-16* using connection pruning. However, convolutional (CONV) layers are the computational bottleneck and many new DNNs use fewer FC layers. For instance, FC layers contribute only 3.99% of parameters in *ResNet-152* [13]. CONV layers, in contrast, account >90% of the total computation. As shown in Figure 1, we illustrate the speedups in CONV layers of *AlexNet* on GPU by adopting [10], [11]. The baseline is profiled by General Matrix Multiply (GeMM) of `cuBLAS`. The sparse matrixes are stored in the format of Compressed Sparse Row (CSR) and accelerated by `cuSPARSE`. Since most reduction is achieved on FC layers, no practical speedups of CONV layers are observed. To reduce the redundancy in CONV layers, Liu *et al.* [23] have achieved >90% weight sparsity of CONV layers

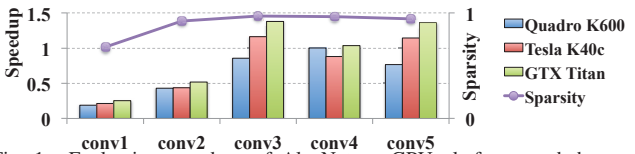



Fig. 1. Evaluation speedups of AlexNet on GPU platforms and the sparsity [28].

978-1-7281-2350-9/19/$31.00 ©2019 IEEE

TABLE I
COMPARISON WITH PREVIOUS MODEL COMPRESSION METHODS.

| | Previous Work | Our Work [28] |
|---|---|---|
| **Pruning** [10], [11] | only focus on FC layers | focus on the computational intensive CONV layers |
| | no practical speedups of CONV layers | higher speedup in CONV layers and throughout the system |
| **LRA** [7], [14], [15], [27] | network structure is fixed | reduce the redundancy within and/or cross layers |
| | reiterations, fine-tuning/cross-validating are needed | no reiterations |
| | search space increases linearly or exponentially | dynamically optimize DNNs with one hypar-parameter |
| **MSL** | use Group Lasso to learn sparse structure [29] or constrain LRA scale [23] | apply group Lasso to regularize multiple DNN structures: filters, channel, filter shapes, and layer depth |

in *AlexNet* with 2% accuracy loss, and bypassed the issue of Figure 1 by hardcoding the sparse weights into program.

**Low rank approximation (LRA).** Denil *et al.* [7] proposed to learn only a small number (*e.g.,* <5%) of weights in a DNN and predict the rest by exploiting the redundancy across filters and channels. Inspired by it, Jaderberg *et al.* [15] used *Low Rank Approximation* (LRA) to construct a low rank basis of filters which can be applied to CPU and CPU frameworks for tunerable (*e.g.,* 4.5×) speedup performance with negligible (*e.g.* ∼1%) accuracy drop. The works in [27] and [14] improved and extended LRA to larger DNNs. However, the network structure compressed by LRA is fixed and provides little flexibility. In addition, 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 [8] [27], the search space increases linearly or even exponentially.

**Model structure learning (MSL).** Group Lasso [29] is an efficient regularization to learn sparse structures. Liu *et al.* [23] utilized group Lasso to constrain the structure scale of LRA. To adapt DNN structure to different databases, Feng *et al.* [9] learned the appropriate number of filters in DNN.

### B. Motivation

Existing sparse learning or pruning methods typically result in non-structural sparsity or connectivity. On real computing platforms, such irregular data can lead to poor data locality, so it can not effectively accelerate the computational process on the hardware, and sometimes even have a negative effect. Figure 2 illustrates the locality differences in data caused by non-structured sparsity model and structured sparsity model on a real computing platform. The dark gray blocks in the figure represent non-zero data, and the dotted blocks represent data with a value of zero. Due to the limited on-chip cache area, the feature maps and/or weights required for the calculation are stored in off-chip main memory. Before each calculation starts, the cache line where the data required for the current computation is located needs to be moved from the off-chip main memory to the on-chip cache. In the non-structural sparse model, as shown in Fig. 2(a), a large amount of ineffectual data (*i.e.,* zeros) movement occurs due to the non-structured sparsity. The latency and energy consumption of off-chip data access is usually two to three orders of magnitude higher than the computation itself. It can be seen that the non-structural sparse model does not make good use of data locality, which will seriously extend the execution time.

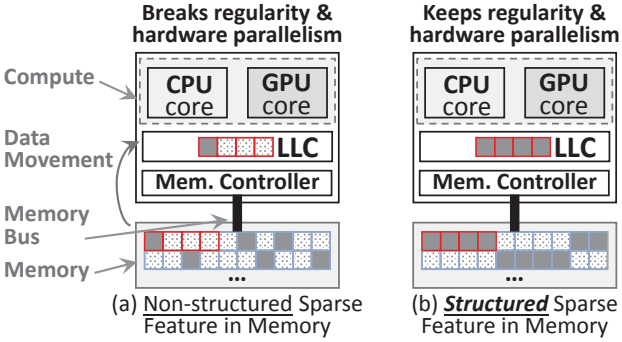



Fig. 2. An illustration of the data locality induced by non-structural and structural sparsity models and the corresponding impact on hardware execution.

We propose structured sparsity learning method which produce clustered non-zero data as shown in Figure 2(b). In this case, the effectual data can be fetched together into the cache, while ineffectual data is easily skipped. This results in less data movement between the computing unit (*i.e.,* CPU/GPU) and off-chip memory, effectively reducing computation time and power consumption. It can be seen that this method can improve the cache locality and accelerate the calculation. We also compares SSL with previous model compression schemes as shown in Table I.

## III. LEARNING STRUCTURED SPARSITY IN DNNS

In this section, we present our study of *Structured Sparsity Learning* (SSL) to regularize the structure of DNNs. We first introduce a generic method which directly learns a compact DNN model with hardware-friendly structured sparsity. Based on this approach, We further discuss methods for adaptively learning DNN structure at a finer level of granularity, such as *filter-wise, channel-wise, shape-wise* and *depth-wise*. At the end, we preset the experimental evaluation of SSL on general-purpose CPU and GPU platforms.

**Generic method.** We use $\boldsymbol{W}^{(l)} \in \mathbb{R}^{N_l \times C_l \times M_l \times K_l}$ to represent the 4-D weight tensor of the $l$-$th$ convolutional layer in a DNN, where $N_l$, $C_l$, $M_l$ and $K_l$ are the size of weight tensor along the axes of filter, channel, height and width, respectively. We denote the collection of all weights in the total $L$ convolutional layers as $\boldsymbol{W}$. Then the proposed generic optimization target of a DNN with structured sparsity regularization can be represented as:

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

The first two items on the right side of the formula, $E_D(\boldsymbol{W})$ and $R(\cdot)$, are commonly used in normal DNN training, rep-

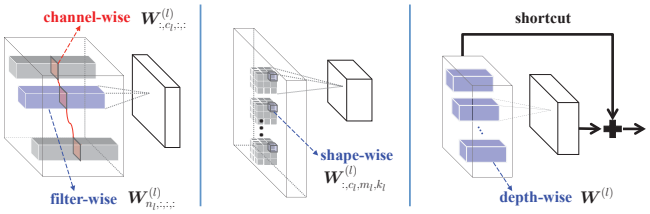



Fig. 3. The proposed *Structured Sparsity Learning* (SSL) for DNNs [28].

resenting *the loss on data* and *non-structured regularization* (*e.g.,* $\ell_2$-norm) applying on every weight respectively. The key of SSL is we adopted group Lasso [19], [30] to effectively zero out all values on a set of weights $\boldsymbol{w}$, denoted in the third item on the right side of the formula 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 $\boldsymbol{w}^{(g)}$.

**Adaptive SSL with a finer level of granularity.** By exploring the way of splitting weight groups $\boldsymbol{w}^{(g)}$, SSL is able to adaptively provide the *filer-wise*, *channel-wise*, *shape-wise*, and *depth-wise* structured sparsity as shownn in Figure 3. For simplicity, we omit the first two items in Eq. 1 and only formulate the $R_g(\boldsymbol{w})$ term in the following expressions.

*1)* 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 regularization of group Lasso to *penalize unimportant filters and channels* can be formulated as:

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

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.

*2)* Assume $\boldsymbol{W}^{(l)}_{:,c_l,m_l,k_l}$ 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. 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 regularization of group Lasso in learning shapes of filers becomes:

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

*3)* We also explore the depth-wise sparsity to regularize the depth of DNNs in order to improve accuracy and reduce computation cost. The corresponding regularization of group Lasso is $\lambda_d \cdot \sum_{l=1}^{L} ||\boldsymbol{W}^{(l)}||_g$. Note that zeroing out all the filters in a layer will cut off the message propagation in a DNN so that the output neurons cannot perform any classification, we propose to to leverage the shortcuts across layers to solve this issue which is similar as the idea proposed by the structure of highway networks [26] and deep residual networks [12]. Detailed illustration can be found in [28].

**Evaluation on CPU/GPU.** Experimental results show that SSL achieves on average 5.1× and 3.1× 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. Meanwhile, the regularized DNN structure significantly improves 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 rate by ∼1%.

## IV. SSL-based Hardware Implementation and Program Optimization

### A. CASE I: Hardware Accelerator for Compressed DNNs [16]

**Motivation.** With the rise of machine learning algorithms and their widespread use, specific hardware accelerators designed for machine learning have become a hot research topic in the computing community. The core computation in DNNs can be summarized as pairwise multiplications between the input feature map and the weight matrix, followed by additions to produce the elements in the output feature map. The key stimulus observations of the sparse DNN accelerator are that most of the computations performed by the DNN are essentially ineffectual because of the multiplication involving a large number of zero operands These calculations do not contribute to the final result and thus can be skipped. Cnvlutin [1] uses zero activation as an indicator to eliminate the relevant computation, but does not exploit zero weights to further reduce the amount of calculation. The work in [18] eliminates the computation involved in both zero weight and zero activation, however, its performance is compromised by its lower parallelism.

To overcome the shortcomings of previous work, we provide a algorithm and hardware co-designed solution to support efficient sparse DNN processing. We conclude the comparison between our work and previous works in Table II. Specifically, the DNN weights are first structurally compressed to eliminate zero parameters by leveraging SSL. Based on this hardware-friendly DNN model, we consider both zero DNN activation and zero weights at the same time. Two technologies, *Structurally-compressed Weight Oriented Fetching* (SWOF) and *In-layer Pipeline for Memory and Computation* (IPMC), are particularly proposed to further increase system performance.

**Design Methodology.** Figure 4 illustrates the overview of the proposed sparse DNN accelerator. It mainly consists of

TABLE II
Comparison with previous sparse DNN accelerators.

| | Zero Act. | Zero W. | Parallelism |
|---|---|---|---|
| Convlutin [1] | ✓ | × | medium |
| [18] | ✓ | ✓ | low |
| Our Work [16] | ✓ | ✓ | extreme high |

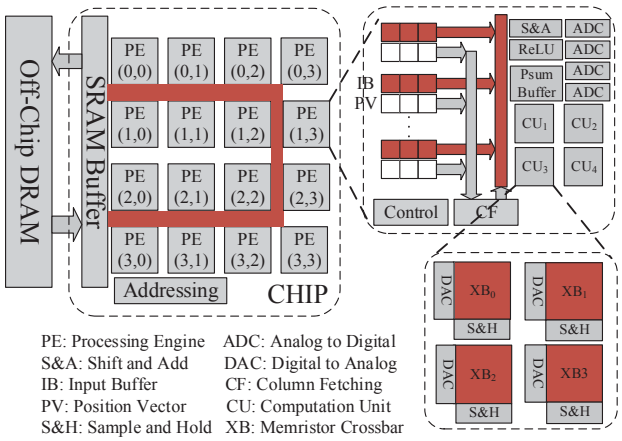



Fig. 4. The top-level of the proposed architecture [16].

a global SRAM on-chip buffer, 16 processing engines (PE) connected in a mesh and an addressing unit. Each PE consists of a number of input buffers (IB), a position vector (PV) for the index, and serveral calculation units (CUs). Note that here CUs leverage the emerging resistive random access memory (ReRAM) with computation capability to perform in-situ DNN execution. In addition, some auxiliary circuitry such as ADCs, DACs, and peripheral components (*e.g.,* buffers and rectification linear unit (ReLU) modules, *etc.*) are included.

In order to overcome the negative impact of the aforementioned non-structured sparsity, we first adopt SSL-based method to zero out weights during the training process. More specifically, we apply regularization to the filter and channel dimensions simultaneously due to the correlation that all-zero filter renders the corresponding channel useless in the next layer of the neural network. In practice, filter-wise regularization can be considered as row reduction; while channel-wise and shape-wise regularization are column reductions in the weight matrix. In this way, we obtain a hardware-friendly DNN model.

Next, we eliminate the ineffectual computation caused by zero activation by *Structurally-compressed Weight Oriented Fetching* (SWOF). Instead of naïvely fetch data from off-chip memory, SWOF locates non-zero activation associates with non-zero weights in two steps, namely, Row-SWOF and Column-SWOF. The sparsity of the rows and columns of a layer are recorded in Row sparsity vector (RSV) and column sparsity vector (CSV), respectively. Row-SWOF is based on the observation that data fetch for the activation in the $j$-th channel of the $(i+1)$-th layer can be terminated if SSL has eliminated the $j$-th filter of the $i$-th layer. The indexes of non-zero value in RSV are delivered to the addressing unit. We call the refined activation tiles as Channel Reduced Activation (CRA) in SRAM buffers. The valid activation data is first reshaped to match the original filter size before loaded into the input buffer. Then the Column-SWOF looks into each channel tiles. Only non-zero activation stay and their positions in filter are recorded in position vector (PV). Fetching unit (FU) compares the indexes of the remaining columns in CSV with those in PV. FU only delivers the activation with mutual indexes to CU.

*In-layer Pipeline for Memory and Computation (IPMC)* is further proposed to decouple the computation and memory access to speed up the DNN execution process. Unlike previous ReRAM-based DNN accelerators [2]–[4], [24], where data access and computation are performed in the same logical cycle, IPMC provides the flexibility to skip the computation stage when input activation in the buffer are all zero. In addition, IPMC also brings the benefits of throughput improvement. For instance, assuming two consecutive activation vectors ($v1$, $v2$) are non-zeros (*i.e.,* need to be computed), the buffer holds fetched vector $v1$ in one cycle, but in the next cycle, this buffer can be used to hold $v2$ while $v1$ is in computation.

TABLE III
WEIGHT MATRIX SPARSITY OF COMPRESSED MODELS [16].

| | Network | Conv1 | Conv2 | Conv3 | Conv4 | Conv5 |
|---|---|---|---|---|---|---|
| Row | LeNet | 0.00% | 0.00% | – | – | – |
| | AlexNet | 9.29% | 12.11% | 39.78% | 46.32% | 0.00% |
| | CaffeNet | 0.00% | 0.00% | 1.57% | 2.86% | 0.00% |
| Column | LeNet | 20.00% | 94.80% | – | – | – |
| | AlexNet | 0.00% | 61.21% | 77.11% | 85.03% | 81.17% |
| | CaffeNet | 0.00% | 23.50% | 39.00% | 38.50% | 24.50% |

TABLE IV
PERFORMANCE IMPROVEMENT AND ENERGY SAVING.

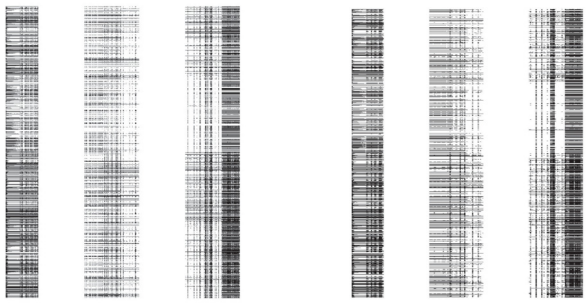
| | Scheme | LeNet | AlexNet | CaffeNet |
|---|---|---|---|---|
| Perf. Speedup | **SWOF-only** | 3.32× | 2.81× | 1.39× |
| | **SWOF+IPMC** | 4.81× | 4.40× | 2.25× |
| Eng. Saving | **SWOF-only** | 3.25× | 2.77× | 1.23× |
| | **SWOF+IPMC** | 3.70× | 3.07× | 1.59× |

**Evaluation.** We evaluate the proposed design using LeNet [22] on MNIST and AlexNet [20]/CaffeNet [17] on LSVRC 2012 (*i.e.,* a subset of ImageNet with approximately 1000 categories and 1000 images in each category). All the training data are reshaped to 256×256 pixels. We summarize the ratio of zeros in each layer in Table III.

Row sparsity indicates a reduction in the filter, and column sparsity indicates a decrease in filter shapes and channels. We can see that SSL can effectively compress the model. Table IV summarizes the normalized hardware performance and power savings achieved by using the proposed sparse DNN accelerator compared to the baseline accelerator in [24]. In general, our design achieves GeoMean speedups of 2.16× and 3.37× when applying *SWOF-only* and *SWOF+IPMC*, respectively. The increment illustrates our accelerator gains extra speedup with IPMC. In addition, our experimental results have also shown that the reduction in filters brings more acceleration on hardware platforms than the reduction in filter shapes and channels.

### *B. CASE II: An SSL Framework for Speech Recognition [31]*

**Motivation.** We further extend SSL to the more challenging *Recurrent Neural Networks* (RNNs) where the basic recurrent unit is shared across in time series. Therefore, compressing (or even removing) the unit can aggressively affect all the time steps. We proposed Efficient Sparse Structures (ESS) learning method for acoustic modeling in speech recognition. ESS simultaneously reduce the sizes of input updates, gates, hidden states, cell states and outputs within the sophisticated RNN structure.



(a) *group-8* (b) *group-16*

Fig. 5. The visualization of weight matrices learned by ESS [31].

**Design Methodology.** ESS is a three-step training pipeline. We first train RNN from scratch with structured-sparsity learning method. At the end of this step, a sparse model at the desired sparsity level will be generated. However, the accuracy of the model is relatively low and will be recovered in the following two steps. In the second step, we fix the zero parameters learned in the previous step to prevent the structural sparsity from updating in the next step. At the end, the sparse model will be used as an initial input model and be trained for additional epochs. Here, the extra group Lasso regularization term in the first step is disabled while the model is retrained to recover the accuracy. Because all the zero parameters are fixed in the second step, only nonzero elements can be updated. In this way, the learned sparsity is maintained while the accuracy gradually increases. Due to the page limit, we omit the details and refer interested readers to [31].

**Evaluation.** We use the open-source Kaldi toolkit to conduct experiments. As visualized in Figure 5, the learned structures show significant sparsity with different group sizes (*e.g.,* 8, 16). Although matrices (*i.e.,* columns in Figure 5) demonstrate difference among each other, the learning process makes it converge to an optimal configuration across all the matrices.

## V. Conclusion

In this work, we summarize our recent research on cross-layer optimization to accelerate DNNs. *Structured Sparsity Learning* (SSL) can adaptively adjusts multiple structures (*e.g.,* filter, channels, filter shapes, network depths) in a DNN to obtain a compact and light model. Through two design cases, we show that SSL-based hardware implementation and program optimization can provide further improvements. Our conclusion is that only interactive design practices that span algorithm and hardware levels can maximize system performance.

## Acknowledgment

This article was supported in part by NSF grants CCF-1910299, CNS-1717657 and CNS-1822085. The authors gratefully acknowledge the support by the memberships of NSF IUCRC for ASIC including Cadence, Unisound, Yonghui Supermarket, Ergomotion, etc. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of grant agencies or their contractors.

## References

[1] J. Albericio, *et al.*, "Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing," in *ISCA*, 2016.
[2] F. Chen and H. Li, "EMAT: An Efficient Multi-task Architecture for Transfer Learning Using ReRAM," in *ICCAD*, 2018.
[3] F. Chen, *et al.*, "ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks," in *ASP-DAC*, 2018.
[4] F. Chen, *et al.*, "ZARA: A Novel Zero-free Dataflow Accelerator for Generative Adversarial Networks in 3D ReRAM," in *DAC*, 2019.
[5] R. Collobert and J. Weston, "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning," in *ICML*, 2008.
[6] M. Deisher and A. Polonski, "Implementation of Efficient, Low Power Deep Neural Networks on Next-Generation Intel Client Platforms," *IEEE SigPort*, 2017.
[7] M. Denil, *et al.*, "Predicting Parameters in Deep Learning," in *NIPS*, 2013.
[8] E. L. Denton, *et al.*, "Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation," in *NIPS*, 2014.
[9] J. Feng and T. Darrell, "Learning The Structure of Deep Convolutional Networks," in *ICCV*, 2015.
[10] S. Han, *et al.*, "Learning both Weights and Connections for Efficient Neural Network," in *NIPS*, 2015.
[11] S. Han, *et al.*, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding," *ICLR*, 2016.
[12] K. He, *et al.*, "Deep Residual Learning for Image Recognition," *arXiv preprint arXiv:1512.03385*, 2015.
[13] K. He, *et al.*, "Deep Residual Learning for Image Recognition," *CVPR*, 2016.
[14] Y. Ioannou, *et al.*, "Training CNNs with Low-Rank Filters for Efficient Image Classification," *arXiv preprint arXiv:1511.06744*, 2015.
[15] M. Jaderberg, *et al.*, "Speeding up Convolutional Neural Networks with Low Rank Expansions," *arXiv preprint arXiv:1405.3866*, 2014.
[16] H. Ji, *et al.*, "ReCom: An efficient resistive accelerator for compressed deep neural networks," in *DATE*, 2018.
[17] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *MM*, 2014.
[18] D. Kim, *et al.*, "A novel zero weight/activation-aware hardware architecture of convolutional neural network," in *DATE*, 2017.
[19] S. Kim and E. P. Xing, "Tree-guided Group Lasso for Multi-task Regression with Structured Sparsity," in *ICML*, 2010.
[20] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.
[21] A. Krizhevsky, *et al.*, "ImageNet Classification with Deep Convolutional Neural Networks," *Commun. ACM*, 2017.
[22] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998.
[23] B. Liu, *et al.*, "Sparse Convolutional Neural Networks," in *CVPR*, 2015.
[24] L. Song, *et al.*, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *HPCA*, 2017.
[25] L. Song, *et al.*, "HyPar: Towards Hybrid Parallelism for Deep Learning Accelerator Array," in *HPCA*, 2019.
[26] R. K. Srivastava, *et al.*, "Highway Networks," *arXiv preprint arXiv:1505.00387*, 2015.
[27] C. Tai, *et al.*, "Convolutional neural networks with low-rank regularization," *arXiv preprint arXiv:1511.06067*, 2015.
[28] W. Wen, *et al.*, "Learning Structured Sparsity in Deep Neural Networks," in *NIPS*, 2016.
[29] M. Yuan and Y. Lin, "Model Selection and Estimation in Regression with Grouped Variables," *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 2006.
[30] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2006.
[31] J. Zhang, *et al.*, "Learning Efficient Sparse Structures in Speech Recognition," in *ICASSP*, 2019.
[32] X. Zhang and Y. LeCun, "Text Understanding from Scratch," *arXiv e-prints*, 2015.