Non-Structured DNN Weight Pruning – Is It Beneficial in Any Platform?

Xiaolong Ma[†], Sheng Lin[†], Shaokai Ye, Zhezhi He, Linfeng Zhang, Geng Yuan, Sia Huat Tan, Zhengang Li, Deliang Fan, Xuehai Qian, Xue Lin, Kaisheng Ma, and Yanzhi Wang

Abstract—Large deep neural network (DNN) models pose the key challenge to energy efficiency due to the significantly higher energy consumption of off-chip DRAM accesses than arithmetic or SRAM operations. It motivates the intensive research on model compression with two main approaches. Weight pruning leverages the redundancy in the number of weights and can be performed in a non-structured, which has higher flexibility and pruning rate but incurs index accesses due to irregular weights, or structured manner, which preserves the full matrix structure with lower pruning rate. Weight quantization leverages the redundancy in the number of bits in weights. Compared to pruning, quantization is much more hardware-friendly, and has become a "must-do" step for FPGA and ASIC implementations. Thus, any evaluation of the effectiveness of pruning should be on top of quantization. The *key open question* is, with quantization, what kind of pruning (non-structured vs. structured) is most beneficial? This question is fundamental because the answer will determine the design aspects that we should really focus on to avoid diminishing return of certain optimizations.

This paper provides a definitive answer to the question for the first time. First, we build ADMM-NN-S by extending and enhancing ADMM-NN, a recently proposed joint weight pruning and quantization framework, with the algorithmic supports for structured pruning, dynamic ADMM regulation, and masked mapping and retraining. Second, we develop a methodology for fair and fundamental comparison of non-structured and structured pruning in terms of both storage and computation efficiency. Our results show that ADMM-NN-S consistently outperforms the prior art: (i) it achieves $348\times$, $36\times$, and $8\times$ overall weight pruning on LeNet-5, AlexNet, and ResNet-50, respectively, with (almost) zero accuracy loss; (ii) we demonstrate the first fully binarized (for all layers) DNNs can be lossless in accuracy in many cases. These results provide a strong baseline and credibility of our study. Based on the proposed comparison framework, with the same accuracy and quantization, the results show that non-structured pruning is not competitive in terms of both storage and computation efficiency. Thus, we conclude that structured pruning has a greater potential compared to non-structured pruning. We encourage the community to focus on studying the DNN inference acceleration with structured sparsity.

Index Terms—Deep neural network, Weight pruning, Quantization, Hardware acceleration.

I. Introduction

DEEP neural networks (DNNs) with very large model sizes are the key enabler for the recent success of deep learning. However, large models incur excessive DRAM accesses which consume significant more energy than arithmetic or SRAM operations. Thus, *model compression* of DNNs became an active and intensively studied research topic. These techniques, which are applied during the training phase of

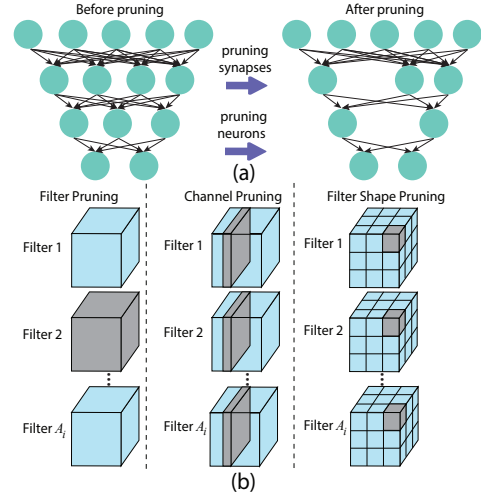[†]These authors contributed equally.

Fig. 1. (a) Non-structured weight pruning (arbitrary weight can be pruned) and (b) three types of structured pruning.

the DNNs, exploit the redundancy in weights. The aim is to simultaneously reduce the model size (thus, the storage requirement) and accelerate the computation for inference, — all to be achieved with minor classification accuracy loss. These techniques are of particular interests to the hardware acceleration of DNN inference engine [1]–[17]. Two important model compression techniques are weight pruning and weight quantization.

Weight pruning leverages the redundancy in the number of weights. One early work [18] used heuristic and iterative weight pruning to achieve weight parameter reduction with negligible accuracy loss. It has been extended in [19]–[23] with more sophisticated heuristics. On the downside, such *non-structured* methods lead to *irregular, sparse weight matrices* (as shown in Figure 1 (a), arbitrary weight can be pruned), which rely on indices to be stored in a compressed format. As a result, they are less compatible with the data parallel execution model in GPUs and multicore CPUs. This drawback is confirmed by the throughput degradation reported in recent works [24], [25]. To overcome the limitation of non-structured pruning, recent works [24], [26] proposed the idea of incorporating *regularity or "structures" in weight pruning*, such as filter pruning, channel pruning, and filter shape pruning, shown in Figure 1 (b). The structured approaches maintain a full matrix with reduced dimensions, and *indices are no longer needed*. As a result, it leads to much higher speedups in GPUs.

Weight quantization is an orthogonal compression technique

	GPU	Multi-core CPU	FPGA	ASIC
Non-Structured Pruning	Not Suitable	Not Suitable	Storage/Computation Dependent ?	
Structured Pruning	Suitable	Suitable		

Fig. 2. Is non-structured pruning beneficial at all?

that leverages the redundancy in the number of bits of weight representation [27]–[34]. Compared to weight pruning, weight quantization is inherently more hardware-friendly, since both storage and computation of DNNs will be reduced proportionally to the weight precision without additional overhead due to indices. Moreover, multiplication operations may be eliminated with binary, ternary, or power-of-2 weight quantizations [32]–[34]. Thanks to these advantages, weight quantization has been a "must-do" step for DNN inference engines. Besides FPGA and ASIC, it is also well supported in GPU, CPU, and mobile devices, e.g., [35], [36].

Given the pros and cons of non-structured/structured weight pruning and weight quantization, they need to be investigated jointly to fully understand the interactions between them. In particular, since weight quantization is a must-do step, especially for FPGA and ASIC, i.e., weight pruning will not be performed alone. The **key open question** is, *with quantization, what kind of pruning (non-structured vs. structured) is most beneficial*? The answer to the question is far from obvious. Using LeNet-5 (for MNIST data set) as an example, we achieve an unprecedented $348\times$ (non-structured) weight reduction with 3-bit quantization, maintaining 99%+ accuracy. However, each index needs to be at least 9-bit on account of $348\times$ weight pruning. This makes index storage larger than that of weights (in addition, indices cannot be further quantized). In this example, non-structured weight pruning results in larger actual storage than structured pruning. Thus, we can see the importance of answering such question: it will determine the design aspects that we should really focus on to avoid diminishing return of certain optimizations. As shown in Figure 2, we need answers for all platforms.

Two recent works ADMM-NN [37] and [27], that perform systematic joint weight pruning and quantization, are in the best position to perform this study. Using advanced variable-splitting optimization method ADMM (Alternating Direction Methods of Multipliers) [38]–[40], state-of-the-art results are achieved (e.g., $21\times$ weight reduction [41] in AlexNet), — outperforming heuristic counterparts. Unfortunately, the current framework is *insufficient* to perform such study. First, ADMM-NN lacks the algorithmic mechanisms to enforce structured weight pruning, and guarantee the solution feasibility. Second, we lack the methodology to fairly and fundamentally compare non-structured and structured pruning in an "apple-to-apple" manner. This paper is the *first* study to provide the answer to the open question with two key contributions.

The **first contribution** of the paper is the development of ADMM-NN-S by extending and enhancing of ADMM-NN [37]. It is extended with the algorithmic supports for structured pruning. We achieve this by adjusting the constraints in each layer to express the structured requirements. For example, for filter pruning, the constraint for a layer can be specified as *number of non-zero filters* is less than or equal to a threshold. Moreover, we develop a systematic framework of dynamic ADMM regulation, masked mapping and retraining to guarantee solution feasibility (satisfying all constraints) and provide high solution quality (ensuring pruning and quantization rate under the same accuracy).

The **second contribution** is the methodology for the fair and fundamental comparison of non-structured and structured weight pruning with quantization in place. We focus on two metrics with the *same accuracy*: 1) total storage (weight+indices), which is computed based on both absolute and relative indices; 2) computation efficiency, which is captured by a metrics called pruning-to-performance ratio (PPR). After pruning, suppose $\alpha\times$ weight reduction results in $\beta\times$ speedup, the PPR value is defined as α/β. Intuitively, the less the value of PPR, the higher the computation efficiency, — same speedup can be achieved by smaller pruning rate. For structured pruning, PPR value is approximately 1 due to the absence of indices. For non-structured pruning, recent accelerators based on non-structured sparsity [37], [42]–[44] show that PPR values are larger than 2.7. We can fairly compare non-structured and structured pruning by conservatively comparing PPR: non-structured pruning is more beneficial if it can achieve $2.7\times$ or higher pruning rate than structured pruning. No prior work has conducted such study and the answer to the above comparison is *unknown*.

The fairness of the proposed methodology is ensured due to three reasons: 1) it is performed by our new ADMM-NN-S framework that significantly *outperforms* prior arts (in both non-structured and structured pruning); 2) the comparison of storage and computation is *hardware implementation-agnostic*; 3) the comparison is performed at *the same rate of accuracy*. We also strengthen weight quantization after non-structured pruning by selectively leveraging state-of-art ternary quantization solution [45].

Based on the proposed ideas, we perform extensive and representative testing of our comparison framework with AlexNet, VGGNet, ResNet-18/50, MobileNet, and LeNet-5 models based on ImageNet, CIFAR-10, and MNIST data sets. Due to space limitation, we focus on convolutional (CONV) layers, which are the most computationally intensive layers in DNNs and are becoming the major storage as well as in state-of-art ResNet and MobileNet models. We do observe similar (and more significant) effect on fully-connected (FC) layers and on RNNs. We highlight our results and findings.

First, ADMM-NN-S framework guarantees solution feasibility while providing high solution quality. Our results consistently and significantly outperform prior art. This is the key to ensure the credibility of our conclusion. Specifically, we 1) achieve *unprecedented* $348\times$, $36\times$, and $8\times$ overall weight pruning on LeNet-5, AlexNet, and ResNet-50 models, respectively, with (almost) zero accuracy loss; 2) derive the first lossless, fully binarized (for all layers) LeNet-5 for MNIST and VGG-16 for CIFAR-10; and 3) derive the first fully binarized (for all layers) ResNet for ImageNet with reasonable accuracy loss.

Second, comparing non-structured and structured pruning, we find that the storage overhead of indices for non-structured pruning is always more than its additional weight storage reduction, thus the amount of total storage for non-structured pruning is actually larger.

In term of computation efficiency, we find that the PPR for structured pruning in all models are less than $2.7\times$. For the first time, our results show that, despite more flexibility and weight pruning rate, *non-structured pruning is not competitive in terms of both storage and computation efficiency* with quantization and the same accuracy with common sparse representation format. As a result, we reach the conclusion that our structured weight pruning framework with weight quantization is extremely suitable for current DNN inference engines as compared to non-structured weight pruning. And we recommend the community to focus on investigating to support this structured sparsity. We release codes and all the models of this work at anonymous link: http://bit.ly/2WMQSRi.

II. MODEL COMPRESSION BACKGROUND

A. Weight Pruning

Non-structured weight pruning. The early work by Han *et al.* [18] achieved $9\times$ reduction in the number of parameters in AlexNet and $13\times$ in VGG-16. However, most reduction is achieved in FC layers, and the $2.7\times$ reduction achieved in CONV layers will not lead to an overall acceleration in GPUs [24]. Extensions of iterative weight pruning, such as [21] (dynamic network surgery), [19] (NeST) and [46], use more delicate algorithms such as selective weight growing and pruning. But the weight pruning rates on CONV layers are still limited, e.g., $3.1\times$ in [21], $3.23\times$ in [19], and $4.16\times$ in [46] for AlexNet with no accuracy degradation. This level of non-structured weight pruning cannot guarantee sufficient speedups in GPUs. In fact, based on the enhanced ADMM-NN framework, we can achieve $11.2\times$ non-structured weight pruning in CONV layers with almost no accuracy degradation. Ironically, it even results in *20% speed degradation* on an NVIDIA 1080Ti GPU.

Structured weight pruning. To overcome the limitation in non-structured, irregular weight pruning, SSL [24] proposes to learn structured sparsity at the levels of filters, channels, filter shapes, layer depth, etc. This work is among the firsts that reported the actually measured GPU accelerations. This is because CONV layers after structured pruning will transform to a full matrix multiplication with reduced matrix size. However, the weight pruning rate is limited in the prior work on structured pruning. The average weight pruning rate on CONV layers of AlexNet is only $1.4\times$ without accuracy loss. More recently, [26] achieved $2\times$ channel pruning with 1% accuracy degradation on VGGNet. More importantly, the structured pruning has never been evaluated with weight quantization.

Special fine-grained structured weight pruning. Recent studies [47]–[49] investigate a new type of fine-grained structured weight pruning which combines non-structured sparsity and structured sparsity into one model. [47] proposed an SIMD-aware pruning method which divides weight matrix into blocks with carefully tailored size for SIMD support, and prunes weight inside each block independently. This hybrid sparsity can fully leverage the multi-level parallel hierarchy in modern CPU [50]. However, it only applicable to devices that support SIMD operation. And another kernel level pruning in [48], [49] is studied, which incorporates non-structured pattern pruning inside convolution kernel and structured kernel pruning. But it needs supports from compiler-assisted inference framework to achieve platform acceleration and is only targeted at edge computing devices such as mobile platforms.

B. Weight Quantization

Weight quantization. This method takes advantages of the inherent redundancy in the number of bits for weight representation. Many of the prior works [27]–[34] focused on quantization of weights to binary values, ternary values, or powers of 2 to facilitate hardware implementation, with acceptable accuracy loss. The state-of-the-art techniques [27], [34] adopt an iterative quantization and retraining framework, with some degree of randomness incorporated into the quantization step. This method results in less than 3% accuracy loss on AlexNet for binary weight quantization [27].

Compared to weight pruning, weight quantization is the major DNN model compression technique utilized in industry, due to its "hardware-friendliness" and the proportional reduction of computation and storage. Thus, weight quantization has been a must-do step in FPGA and ASIC designs of DNN inference engines. Also, it is well supported in GPUs and mobile devices, e.g., PyTorch [36] in NVIDIA GPU and TensorFlow Lite [35] for mobile devices.

C. ADMM for Weight Pruning/Quantization

Recent work [27], [37] have incorporated ADMM for DNN weight pruning and weight quantization, respectively. ADMM is a powerful tool for optimization, by decomposing an original problem into two subproblems that can be solved separately and efficiently. For example, considering optimization problem $\min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{x})$. In ADMM, this problem is decomposed into two subproblems on $\mathbf{x}$ and $\mathbf{z}$ (auxiliary variable), which will be solved iteratively until convergence. The first subproblem derives $\mathbf{x}$ given $\mathbf{z}$: $\min_{\mathbf{x}} f(\mathbf{x}) + q_1(\mathbf{x}|\mathbf{z})$. The second subproblem derives $\mathbf{z}$ given $\mathbf{x}$: $\min_{\mathbf{z}} g(\mathbf{z}) + q_2(\mathbf{z}|\mathbf{x})$. Both q_1 and q_2 are quadratic functions.

ADMM is conventionally utilized to accelerate the convergence of convex optimization problems and enable distributed optimization, in which the optimality and fast convergence rate has been proven [38], [40]. As a special property, ADMM can effectively deal with a subset of combinatorial constraints and yields optimal (or at least high quality) solutions [51], [52]. Luckily, the associated constraints in the DNN weight pruning and quantization belong to this subset of combinatorial constraints, making ADMM applicable to DNN mode compression. However, due to the non-convex nature of the objective function for DNN training, there is still a lack of guarantee in the prior work [27], [37] on *solution feasibility* and *solution quality*. Moreover, [37] only supports non-structured pruning.
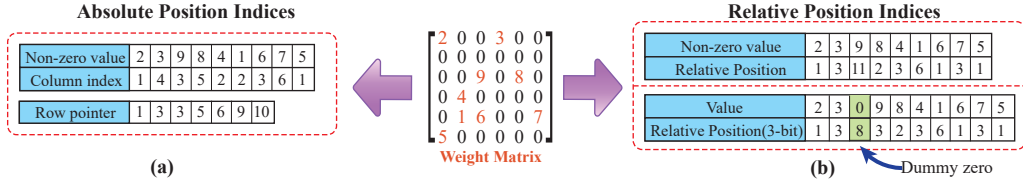
Fig. 3. Compressed sparse row (CSR) format with (a) absolute indices and (b) relative indices.

III. NON-STRUCTURED VS. STRUCTURED WEIGHT PRUNING

A. Non-Structured Pruning: Indexing Overhead

Indices are used to represent weight matrices in the sparse format, thereby achieving storage reduction in non-structured weight pruning. A representative sparse representation format is the *compressed sparse row* (CSR) format, which was also utilized in prior work [2], [18]. As shown in Figure 3 (a), it represents a matrix by three arrays, which respectively contains nonzero (weight) values, column indices and the extents of rows. This representation requires $2n + r + 1$ numbers, where n is the number of nonzero values and r is the number of rows.

We call the above representation as *CSR with absolute indices*. Instead of storing the absolute position, we can compute the index difference and store the indices with relative position. This representation requires $2n$ numbers, where n is the number of nonzero (weight) values. For further compression, one can restrict the number of bits (3 bits in this example) to represent the relative position and add a dummy zero weight when the relative position exceeds the largest value (8 for this example) that can be represented, both shown in Figure 3 (b). These cases are called *CSR with relative indices*.

Comparing the two options, CSR with relative indices is good for compression [18], while CSR with absolute indices leads to better hardware acceleration [42]–[44]. In this work, we aim to allow the highest freedom for non-structured pruning in storage and computation evaluations, — we allow CSR with relative indices in storage calculation and CSR with absolute indices for computation estimation for non-structured pruning. There are also other sparse representation formats for sparse weight storage and corresponding hardware accelerator design [53]–[55], which provide even higher compression gains. However, we focus on the efficient execution of DNNs for general hardware platform. To conduct fair comparisons with previous works, we choose CSR because of its better support in common deep learning packages.

B. Structured Pruning: Three Types

Wen *et al.* [24] introduced three types of structured pruning: *filter pruning*, *channel pruning*, and *filter shape pruning*, as shown in Figure 1 (b). Filter pruning removes whole filter(s); channel pruning removes whole channels; and filter shape pruning removes the weights in the same locations of all filters in one specific layer. Moreover, as shown in Figure 4, filter pruning and channel pruning are correlated. Pruning a filter in layer i is equivalent to pruning the corresponding channel in layer $i + 1$, which is generated by this specific filter. As
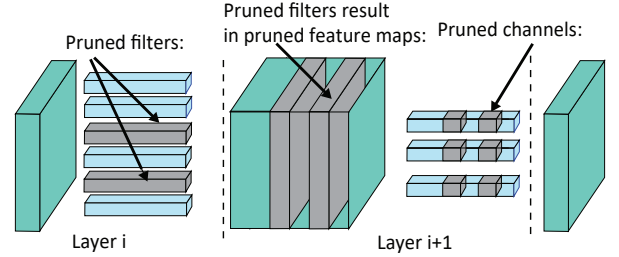

Fig. 4. Relation between filter pruning and channel pruning. Pruned filters in layer i results in pruned feature maps and therefore pruned (inactivated) channels in layer $i + 1$.

a result, filter pruning (and channel pruning) has a roughly quadratic effect on the weight parameter reduction (and the amount of computations) of the DNNs.

The CONV operations in (one layer of) DNNs are commonly transformed to matrix multiplications by converting weight tensors and feature map tensors to matrices [13], named *general matrix multiplication* or GEMM, as shown in Figure 5. From Figure 5 (b), filter pruning corresponds to reducing one row, and thus is also termed *row pruning*. Filter shape pruning corresponds to reducing one column, and thus is also termed *column pruning*. Channel pruning corresponds to reducing multiple consecutive columns. The three structured pruning techniques, along with their combinations, will reduce the dimensions in GEMM while maintaining a full matrix format. Thus, indices are not needed. It is why structured pruning is in general more suitable for hardware accelerations.

On one hand, the major advantage of filter/channel pruning has the superlinear effect on storage/computation reduction, i.e., $\alpha\times$ filter pruning on all layers results in over $\alpha\times$ reduction in number of weight parameters. On the other hand, column pruning has a higher degree of flexibility. These techniques can be largely combined in order to achieve the highest rates in reductions of computation and storage, and effective heuristic for the desirable combination is needed.

C. Debate on Non-Structured VS. Structured: Our Stand

There are discussions on the value between structured vs. non-structured pruning. A representing work [56] stated that the large-sparse models outperform small-dense models, and demonstrates a significant accuracy gap between non-structured sparsified model and structurally size-reduced model. We found the reason for this phenomenon is the pruning method used in [56] is heuristic, which can not achieve good non-structured pruning results, not to mention performing a more challenging structured pruning. In this paper, the

proposed ADMM-NN-S pruning framework is systematic that solves pruning problems with an optimization technique.When the pruning rate is low, ADMM-NN-S can even improve accuracy consistently. In Section VII, our structured pruning results achieve comparable high pruning rates and high accuracy to non-structured pruning results. More importantly, structured pruning is more suitable for hardware acceleration, yet [56] did not discuss its results in this aspect. In this paper, we state that structured pruning is more favorable than non-structured pruning due to its high accuracy and hardware friendliness.
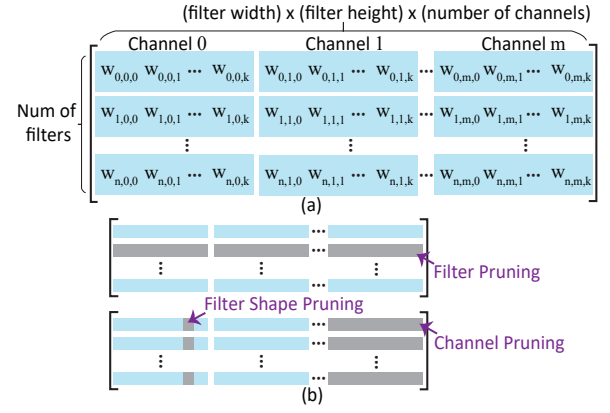

Fig. 5. (a) To support GEMM, weight tensor representation of a CONV layer is transformed into weight matrix representation. (b) How different structured weight pruning schemes are implemented on weight matrix representation.

IV. ADMM-NN-S FRAMEWORK

In this section, we build ADMM-NN-S, a unified solution framework of both non-structured and structured weight pruning, as well as weight quantization problems by extending ADMM-NN, the state-of-the-art ADMM-based framework [37]. The differences between ADMM-NN-S and ADMM-NN are: 1) it supports structured pruning; 2) it can guarantee solution feasibility and provide high solution quality; and 3) we propose effective techniques for enhancing convergence.

A. Enforcing Structured Pruning

This section discusses the extension of ADMM-NN with structured pruning constraints. Consider an N-layer DNN with both CONV and FC layers. The weights and biases of the i-th layer are respectively denoted by $\mathbf{W}_i$ and $\mathbf{b}_i$, and the loss function associated with the DNN is denoted by $f\big(\{\mathbf{W}_i\}_{i=1}^{N}, \{\mathbf{b}_i\}_{i=1}^{N}\big)$; see [41]. In our discussion, $\{\mathbf{W}_i\}_{i=1}^{N}$ and $\{\mathbf{b}_i\}_{i=1}^{N}$ respectively characterize the collection of weights and biases from layer 1 to layer N. Then DNN weight pruning or weight quantization is formulated as optimization problem:

$$\begin{aligned}
\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad & f\big(\{\mathbf{W}_i\}_{i=1}^{N}, \{\mathbf{b}_i\}_{i=1}^{N}\big), \\
\text{subject to} \quad & \mathbf{W}_i \in \mathcal{S}_i, \ i = 1, \ldots, N,
\end{aligned} \tag{1}$$

Next we introduce constraint sets $\mathcal{S}_i$'s corresponding to the non-structured weight pruning, different types of structured pruning, as well as weight quantization. We use CONV layers as illustrative example since CONV layers are the most computationally intensive. The problem formulation can be well applied to FC layers [41].

The collection of weights in the i-th CONV layer is a four-dimensional tensor, i.e., $\mathbf{W}_i \in R^{A_i \times B_i \times C_i \times D_i}$, where $A_i, B_i, C_i,$ and D_i are respectively the number of filters, the number of channels in a filter, the height of the filter, and the width of the filter, in layer i. In the following, if $\mathbf{X}$ denotes the weight tensor in a specific layer, let $(\mathbf{X})_{a,:,:,:}$ denote the a-th filter in $\mathbf{X}$, $(\mathbf{X})_{:,b,:,:}$ denote the b-th channel, and $(\mathbf{X})_{:,b,c,d}$ denote the collection of weights located at position $(:,b,c,d)$ in every filter of $\mathbf{X}$, as illustrated in Figure 1 (b).

Weight pruning: For *non-structured weight pruning*, the constraint on the weights in i-th layer is $\mathbf{W}_i \in \mathcal{S}_i := \{\mathbf{X} \mid$ number of nonzero elements in $\mathbf{X}$ is less than or equal to $\alpha_i\}$. For *filter pruning* (row pruning), the constraint in the i-th CONV layer becomes $\mathbf{W}_i \in \mathcal{S}_i := \{\mathbf{X} \mid$ the number of nonzero filters in $\mathbf{X}$ is less than or equal to $\beta_i\}$. For *channel*

pruning, the constraint becomes $\mathbf{W}_i \in \mathcal{S}_i := \{\mathbf{X} \mid$ the number of nonzero channels in $\mathbf{X}$ is less than or equal to $\gamma_i\}$. Finally, for *filter-shape pruning* (column pruning), the constraint in the i-th CONV layer is $\mathbf{W}_i \in \mathcal{S}_i := \{\mathbf{X} \mid$ the number of nonzero vectors in $\{\mathbf{X}_{:,b,c,d}\}_{b,c,d=1}^{B_i,C_i,D_i}$ is less than or equal to $\theta_i\}$. These α_i, β_i, γ_i, and θ_i values are hyperparameters determined in prior, and the determination procedure will be discussed in Section IV-D.

Weight quantization: For weight quantization, elements in $\mathbf{W}_i$ assume one of $q_{i,1}, q_{i,2}, ..., q_{i,M_i}$ values, where M_i denotes the number of these fixed values. The $q_{i,j}$ values are *quantization levels* of weights of layer i in increasing order, and we focus on *equal-distance quantization* (the same distance between adjacent quantization levels) to facilitate hardware implementation.

B. Enhancing Solution Feasibility and High Solution Quality

In problem (1), the constraint is combinatorial. As a result, this problem cannot be solved directly by stochastic gradient descent methods like original DNN training. However, the form of the combinatorial constraints on $\mathbf{W}_i$ is compatible with ADMM which is recently shown to be an effective method to deal with such clustering-like constraints [51], [52].

Despite such compatibility, it is still challenging to directly apply ADMM due to the non-convexity in objective function. To overcome this challenge, we propose dynamic ADMM regularization, masked mapping and retraining steps for both non-structured and structured pruning. By integrating these techniques, ADMM-NN-S can guarantee solution feasibility (satisfying all constraints) and provide high solution quality (pruning/quantization rate under the same accuracy). The procedure of ADMM-NN-S is shown in Figure 6.

ADMM Regularization Step: The ADMM regularization decomposes the original problem (1) into two subproblems through[1] (i) defining indicator function

$$g_i(\mathbf{W}_i) = \begin{cases} 0 & \text{if } \mathbf{W}_i \in \mathcal{S}_i, \\ +\infty & \text{otherwise} \end{cases}$$

---

[1] The details of ADMM are presented in [40], [41]. We omit the details due to space limitation.
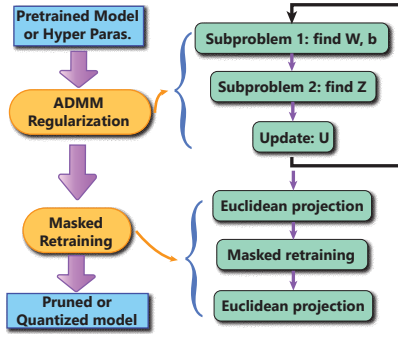
Fig. 6. Procedure of ADMM-NN-S.

corresponding to every set $\mathcal{S}_i$; (ii) incorporating auxiliary variables $\mathbf{Z}_i$, $i = 1,\ldots,N$; and (iii) adopting augmented Lagrangian [40]. These decomposed subproblems will be iteratively solved until convergence. The first subproblem is

$$\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad f\left(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\right) + \sum_{i=1}^N \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i^k + \mathbf{U}_i^k\|_F^2, \quad (2)$$

where $\mathbf{U}_i^k := \mathbf{U}_i^{k-1} + \mathbf{W}_i^k - \mathbf{Z}_i^k$. The first term in the objective function of (2) is the differentiable loss function of the DNN, and the second term is a quadratic regularization term of the $\mathbf{W}_i$'s, which is differentiable and convex. As a result (2) can be solved by stochastic gradient descent as original DNN training. Please note that this first subproblem maintains the same form and solution for (non-structured and structured) weight pruning and quantization problems.

The second subproblem is given by

$$\underset{\{\mathbf{Z}_i\}}{\text{minimize}} \quad \sum_{i=1}^N g_i(\mathbf{Z}_i) + \sum_{i=1}^N \frac{\rho_i}{2} \|\mathbf{W}_i^{k+1} - \mathbf{Z}_i + \mathbf{U}_i^k\|_F^2. \quad (3)$$

Note that $g_i(\cdot)$ is the indicator function of $\mathcal{S}_i$, thus this subproblem can be solved analytically and optimally [40]. For $i = 1,\ldots,N$, the optimal solution is the Euclidean projection of $\mathbf{W}_i^{k+1} + \mathbf{U}_i^k$ onto $\mathcal{S}_i$. For *non-structured weight pruning*, we can prove that the Euclidean projection results in keeping α_i elements in $\mathbf{W}_i^{k+1} + \mathbf{U}_i^k$ with the largest magnitudes and setting the remaining weights to zeros. For *filter pruning*, we first calculate $O_a = \|(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{a,:,:,:}\|_F^2$ for $a = 1,\ldots,A_i$, where $\|\cdot\|_F$ denotes the Frobenius norm. We then keep β_i elements in $(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{a,:,:,:}$ corresponding to the β_i largest values in $\{O_a\}_{a=1}^{A_i}$ and set the rest to zero. For *channel pruning*, we first calculate $O_b = \|(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{:,b,:,:}\|_F^2$ for $b = 1,\ldots,B_i$. We then keep γ_i elements in $(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{:,b,:,:}$ corresponding to the γ_i largest values in $\{O_b\}_{b=1}^{B_i}$ and set the rest to zero. The optimal solution of the second subproblem for *filter shape pruning* is similar, and is omitted due to space limitation. For *weight quantization*, we can prove that the Euclidean projection results in mapping every element of $\mathbf{W}_i^{k+1} + \mathbf{U}_i^k$ to the quantization level closest to that element.

After both subproblems solved, we update the dual variables $\mathbf{U}_i$'s according to the ADMM rule [40] and thereby complete one iteration in ADMM regularization. Overall the ADMM regularization step can be understood as a smart, dynamic L_2

regularization, in which the regularization target $\mathbf{Z}_i^k - \mathbf{U}_i^k$ will change judiciously and analytically in each iteration. On the other hand, conventional regularization methods (based on L_1, L_2 norms or their combinations) use a fixed regularization target, and the penalty is applied on all the weights. This will inevitably cause accuracy degradation. Sample comparison results are in Section V.

Masked mapping and retraining: After ADMM regularization, we obtain intermediate $\mathbf{W}_i$ solutions. The subsequent step of masked mapping and retraining will guarantee the solution feasibility and improve solution quality. For non-structured and structured weight pruning, the procedure is more straightforward. We first perform the said Euclidean projection (mapping) to guarantee that pruning constraints are satisfied. Next, we mask the zero weights and retrain the DNN with non-zero weights using training sets, while keeping the masked weights 0. In this way test accuracy (solution quality) can be (partially) restored, and solution feasibility (constraints) will be maintained.

For weight quantization, the procedure is more complicated. The reason is that the retraining process will affect the quantization results, thereby solution feasibility. To deal with this issue, we first perform Euclidean projection (mapping) of weights that are close enough (defined by a threshold value ϵ) to nearby quantization levels. Then we perform retraining on the remaining, unquantized weights (with quantized weights fixed) for accuracy improvement. Finally we perform Euclidean mapping on the remaining weights as well. In this way the solution feasibility will be guaranteed.

C. Techniques for Enhancing Convergence

In this section we discuss two techniques for enhancing convergence (rate and results): multi-ρ method in ADMM regularization, and progressive weight pruning. We abandon the extragradient descent method in [27] as we did not find the advantage in convergence speed, not to mention the additional hyperparameters introduced by this method.

Increasing ρ in ADMM regularization: The ρ_i values are the most critical hyperparameter in ADMM regularization. We start from smaller ρ_i values, say $\rho_1 = \cdots = \rho_N = 1.5 \times 10^{-3}$, and gradually increase with ADMM iterations. This coincides with the theory of ADMM convergence [51], [52]. It in general takes 8 - 12 ADMM iterations for convergence, corresponding to 100 - 150 epochs in PyTorch. This convergence rate is comparable with the original DNN training.

Progressive weight pruning: The ADMM regularization is L_2 regularization. As a result, there is a large portion of very small weights values after one round of ADMM-based (non-structured or structured) weight pruning. This gives rise to the opportunity to perform a second round of weight pruning. In practice, we perform *two rounds* of ADMM-based weight pruning consecutively, where the weight pruning results in the first round will be the starting point of the second round (weights that are already pruned to zero will not be recovered). This method has an additional benefit of reducing the search space in each step, thereby accelerating convergence.

D. Determining Hyperparameters

Hyperparameter determination mainly refers to the determination process of pruning rate (e.g., the α_i value) and/or the number of quantization levels per layer of DNN. This is a more challenging task for pruning than quantization in general. For quantization, it is typically preferred for the same number of quantization levels for all (or most of) layers, like binarized or ternarized weights, which is preferred by hardware. For weight pruning, on the other hand, these pruning rate values are flexible and shall be judiciously determined.

As hyperparameter determination is not our primary focus, we use a heuristic method as follows. We observe that we can achieve at least $3\times$ more weight pruning than prior, heuristic weight pruning methods without accuracy loss. Hence, we adopt the per-layer pruning rates reported in prior work, and increase proportionally. In the progressive pruning procedure, we set the target of the first round to be $1.5\times$ pruning than prior work, and the second round to be doubled based on that. We will further increase the pruning rates if there is still margin for weight pruning without accuracy loss.

V. NON-STRUCTURED DNN WEIGHT PRUNING AND QUANTIZATION RESULTS

In this section, we demonstrate the effectiveness of ADMM-NN-S for non-structure pruning and quantization, based on ImageNet ILSVRC-2012, CIFAR-10, and MNIST data sets, using AlexNet [57], VGGNet [58], ResNet-18/ResNet-50 [59], MobileNet-V2 [60], and LeNet-5 DNN models. Due to space limitation, we only show the results of the overall DNN model (which has the most prior work for comparison), and binarized quantization of DNNs. Our implementations are based on PyTorch, and the baseline accuracy results are in many cases higher than those utilized in prior work, which reflects the recent training advances. For example, in the AlexNet model we utilize a baseline with Top-1 accuracy 60.0% and Top-5 accuracy 82.2%, both higher than prior work (57.2% Top-1 and 80.2% Top-5). We conduct a fair comparison because we focus on *relative accuracy* with our baseline instead of the absolute accuracy (which has outperformed prior work).

Thanks to the compatibility of ADMM-NN-S with DNN training, directly training a DNN model using the framework achieves the same result as using a pre-trained DNN model. When a pre-trained DNN model is utilized, we limit the number of epochs in both steps in the progressive framework to be 120, similar to the original DNN training in PyTorch and is much lower than the iterative pruning heuristic [18].

A. Non-Structured Weight Pruning Results

AlexNet Results for ImageNet Dataset: Table I compares the overall pruning rates of the whole AlexNet model (CONV and FC layers) vs. accuracy, between the proposed framework and various prior methods. We can clearly observe that the proposed framework outperforms prior methods, including the prior ADMM method [41]. With almost no accuracy loss even based on the high baseline accuracy, we achieve $36\times$ overall pruning rate. We achieve a notable $63\times$ weight reduction

TABLE I
OVERALL WEIGHT PRUNING RATE COMPARISONS ON ALEXNET MODEL
FOR IMAGENET DATA SET.

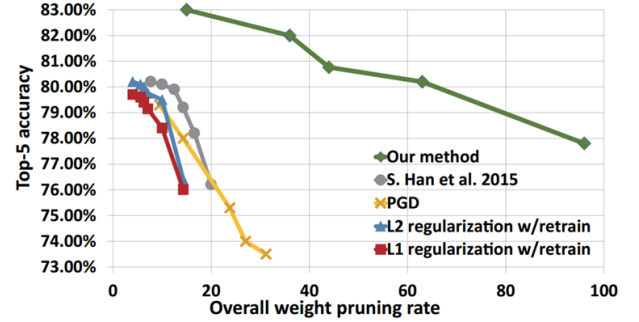| Method	Top-5 accuracy	Relative accuracy loss	Overall prun. rate
Iter. prun. [18]	80.3%	−0.1%	9.1×
NeST [19]	80.3%	−0.1%	15.7×
Dyn. surg. [21]	80.0%	+0.2%	17.7×
ADMM [41]	80.2%	−0.0%	17.7×
Our method	82.0%	+0.2%	36×
Our method	80.8%	+1.4%	44×
Our method	80.3%	+1.9%	63×
Our method	77.8%	+4.4%	96×

Fig. 7. Top-5 accuracies for different pruning methods on AlexNet for ImageNet dataset.

with 80.3% Top-5 accuracy, just slightly below the baseline accuracy in prior work.

Figure 7 illustrates the absolute top-5 accuracy for different pruning methods, on AlexNet model for ImageNet dataset. These methods include our proposed solution, iterative pruning [18], fixed regularization techniques like L_1 and L_2 regularizations, and projected gradient descent (PGD). The results clearly show that the proposed method outperforms the others both in absolute accuracy and in relative accuracy loss.

ResNet-50 Results for ImageNet Dataset: Due to the lack of existing effective pruning results, we conduct uniform weight pruning, — use the same pruning rate for all CONV and FC layers. The results are shown in Table II. We achieve $8\times$ overall pruning rate (also $8\times$ pruning rate on CONV layers) on ResNet-50 without accuracy loss. These results clearly outperform the prior work.

TABLE II
COMPARISONS OF OVERALL WEIGHT PRUNING RESULTS ON RESNET-50
FOR IMAGENET DATA SET.

Method	Top-5 Acc. Loss	Pruning rate
Uncompressed	0.0%	1×
Fine-grained [46]	0.1%	2.6×
ADMM-NN [37]	0.0%	7×
Our method	0.0%	8×
Our method	0.7%	17.4×

MobileNet-V2 Results for ImageNet Dataset: Table III shows the non-structured pruning results on MobileNet-V2 for ImageNet dataset. MobileNet-V2 is a lightweight network structure with less than 2.2M parameters. The Top-5 accuracy of the uncompressed model is 90.4%. Due to the fact that MobileNet-V2 is one of the most difficult networks to prune, there are no existing effective irregular pruning results. Thus,

we only include our results in Table III.

TABLE III
COMPARISONS OF OVERALL WEIGHT PRUNING RESULTS ON
MOBILENET-V2 FOR IMAGENET DATA SET.

Method	Top-5 Acc. Loss	Pruning rate
Uncompressed	0.0%	1×
Our method	0.6%	2.0×

MobileNet-V2 Results for CIFAR-10 Dataset: The baseline accuracy is as high as 95.07% due to the adoption of mixup technique. We present our results in Table IV due to the lack of prior work for fair comparison. We achieve 5.7× weight pruning with almost no accuracy loss, starting from the high-accuracy baseline. We achieve 10× weight pruning (which is highly challenging for MobileNet) with only 1.3% accuracy loss.

TABLE IV
OUR WEIGHT PRUNING RESULTS ON MOBILENET-V2 FOR CIFAR-10
DATA SET.

Method	Accuracy	Pruning rate
Uncompressed	95.07%	1×
Our method	94.95%	5.7×
Our method	94.70%	6.7×
Our method	93.75%	10×

LeNet-5 Results for MNIST Dataset: Table V demonstrates the comparison results on LeNet-5 model using MNIST data set. We achieve an unprecedented 348× overall weight reduction with almost no accuracy loss. It clearly outperforms prior methods including one-shot ADMM-based method [41].

TABLE V
COMPARISONS OF OVERALL WEIGHT PRUNING RESULTS ON LENET-5 FOR
MNIST DATA SET.

Method	Accuracy	Pruning rate
Uncompressed	99.2%	1×
Network Pruning [18]	99.2%	12.5×
ADMM [41]	99.2%	71.2×
Our method	99.2%	246×
Our method	99.0%	348×

B. Binary Weight Quantization Results

Due to space limitation, we mainly show the results on fully binarized DNN models (i.e., weights in all layers, including the first and the last, are binarized), which is a highly challenging task. Please note that the amount of prior work on fully binarized weight quantization is very limited due to the highly challenging nature.

Weight Quantization Results on LeNet-5 and CIFAR-10: To the best of our knowledge, we achieve *the first lossless, fully binarized LeNet-5 model*. The accuracy is still 99.21%, lossless compared with baseline. In prior works, achieving lossless is challenging even for MNIST. For example, recent work [61] results in 2.3% accuracy degradation on MNIST for full binarization, with baseline accuracy 98.66%. We also achieve *the first lossless, fully binarized VGG-16 for CIFAR-10*. The accuracy is 93.53%. We would like to point out that fully ternarized quantization results in 93.66% accuracy. Table VI shows our results and comparisons.

TABLE VI
COMPARISONS OF FULLY BINARY (TERNARY) WEIGHT QUANTIZATION
RESULTS ON VGG-16 FOR CIFAR-10 DATA SET.

Method	Accuracy	Num. of bits
Baseline of [61]	84.80%	32
Binary [61]	81.56%	1
Our baseline	93.70%	32
Our ternary	93.66%	2 (ternary)
Our binary	93.53%	1

Binary Weight Quantization Results on ResNet for ImageNet: The binarization of ResNet models on ImageNet data set is widely acknowledged as an extremely challenging task. As a result, there are very limited prior work (e.g., the prior ADMM-based method [27]) with binarization results on ResNet models. As [27] targets ResNet-18, we make a fair comparison on the same model. Table VII demonstrates the comparison results (Top-5 accuracy loss). In prior work, by default the first and last layers are not quantized (to 8 bits) as these layers have a significant effect on overall accuracy. When leaving the first and last layers unquantized, we observe the higher accuracy compared with the prior method. The Top-1 accuracy has similar result: 3.8% degradation in our method and 4.3% in [27].

Furthermore, we can derive a *fully binarized ResNet-18*, in which weights in all layers are binarized. The accuracy degradation is 5.8%, which is noticeable and shows that the full binarization of ResNet is a challenging task even for the proposed framework. We did not find prior work to compare with this result.

TABLE VII
COMPARISONS OF WEIGHT QUANTIZATION RESULTS ON RESNET-18 FOR
IMAGENET DATA SET.

Method	Relative Top-5 acc. loss	Num. of bits
Uncompressed	0.0%	32
ADMM [27]	2.9%	1 (8 for the first and last)
Our method	2.5%	1 (8 for the first and last)
Our method	5.8%	1

Summary The results presented in this section show that ADMM-NN-S can achieve better results compared to state-of-the-art. In certain cases, ADMM-NN-S achieves unprecedented weight reduction. These results provide a strong baseline and credibility of our study.

VI. NON-STRUCTURED VS. STRUCTURED: THE COMPARISONS METHODOLOGY

A Motivation Example: The previous section has shown the superior results on joint weight pruning and quantization. Using LeNet-5 (MNIST data set) as an example, we achieve an unprecedented 348× non-structured weight reduction together with 3-bit quantization, maintaining 99%+ accuracy. When indices are not accounted for, the overall compression rate is an unprecedented 3,712× compared with the original LeNet-5 model without compression. However, each index needs to be at least 9-bit considering 348× weight pruning. This makes index storage even larger than weights, and indices cannot be further quantized. As a result, *non-structured weight pruning in fact results in larger actual storage than structured pruning*.

The fundamental phenomena shown here is that, with quantization the weight reduction by non-structured pruning is offset by the extra index storage. It motivates us to study whether it is a common trend with weight quantization in place? If the answer is yes, then the value of non-structured weight pruning will be further in doubt. This is because non-structured pruning is already less preferred for GPU and CPUs [24], [25], the only benefit is the potentially higher pruning rates due to greater pruning flexibility. If this benefit is also lost, there will be nearly no merit of non-structured sparsity for hardware acceleration of DNNs, considering the impacts on computation efficiency and degraded parallelism. Importantly, such conclusion will also be true for FPGA and ASIC and guide us to the design aspects that we should really focus on.

In this section, we conduct the *first(to the best of our knowledge) comprehensive study to understand the value of non-structured and structured pruning, with quantization in place and the same accuracy*. It is worth noting that without ADMM-NN-S framework, this study is *not possible*, — we need a framework that achieves competitive results and can jointly perform both weight pruning and quantization.

A Hardware Implementation-Agnostic Comparison Methodology: We conduct a fair comparison between non-structured and structured weight pruning with quantization in place, based on the unified solution framework. Note that the comparison framework is more FPGA and ASIC oriented as flexible weight quantization is assumed. However, we would like to point out that a moderate, fixed weight quantization, e.g., 8 bit, supported in GPU [36], TPU [62], and mobile devices [35], will result in a similar conclusion.

The key characteristic of our comparison framework is that it is *hardware implementation-agnostic*. Our intention is that the comparison results will be independent of specific hardware implementations, and as a result, the conclusion will unlikely to change for architectural advances in either type of pruning. Therefore, we directly compare the amounts of *storage* and estimated *computation efficiency* for non-structured and structured weight pruning with quantization in place, which capture the fundamental trade-offs. Intuitively, storage is measured as the total weight and index storage with quantization in place. Storage of intermediate results is not considered, and this favors non-structured pruning, — structured, filter/channel pruning will likely benefit more in intermediate results storage reduction.

On the other hand, computation efficiency is estimated using the PPR values derived from prior work on non-structured sparsity accelerators [37], [42]–[44]. For structured pruning, $\alpha\times$ weight reduction results in around $\alpha\times$ speedup (slightly higher or lower depending on platform and problem), and the PPR value is approximately 1. For non-structured pruning, $\alpha\times$ weight reduction only results in $\beta\times$ speedup with $\beta < \alpha$. In the state-of-art tapeouts [42], the PPR value $\alpha/\beta > 3$, which is close to 3 with a low pruning rate and higher than 4 for a high pruning rate. In synthesis results [37], [43], [44], this PPR value ranges from 2.7 to 3.5. We use the smallest value 2.7 that favors non-structured pruning the most. In other words, if non-structured pruning achieves more than $2.7\times$ pruning rate than structured one (or equivalently, structured pruning rate is
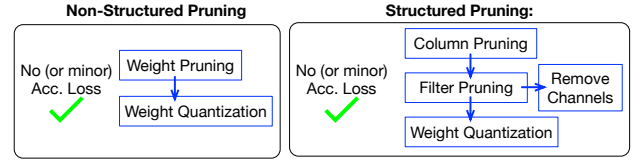

Fig. 8. Procedure for maintaining accuracy.

less than 37% of non-structured one) under the same accuracy and quantization level, the former is more preferred in terms of computation. Otherwise, the latter is more preferred.

Maintaining the Same Accuracy for Comparison: The proposed comparison is performed under the same accuracy for non-structured and structured pruning with quantization in place. The *precise accuracy control*, which is challenging for prior work, is enabled by the unified solution framework. For most cases, we would like to have (almost) no accuracy degradation compared with the baseline DNN model without pruning or quantization. For non-structured pruning, it is achieved in two steps: 1) perform weight pruning to the maximum extent such that there will be no accuracy loss; and 2) perform weight quantization (hopefully) not to cause accuracy loss. For structured pruning, we give priority to column pruning, and perform three steps: 1) perform column pruning to the maximum extent without accuracy loss; 2) perform filter pruning and reduce corresponding redundant channels; and 3) perform weight quantization (hopefully) without accuracy loss. Figure 8 illustrates the procedure for maintaining accuracy. Of course the proposed framework is also applicable if certain accuracy degradation is allowed. A larger margin of accuracy loss in general favors structured pruning, because higher pruning rates can be achieved for both pruning schemes, but non-structured pruning requires more bits for indices.

There is more subtlety in the combination of non-structured pruning and quantization. If a weight is non-zero after pruning but quantized to zero, this weight can be added to the pruned list to achieve a higher pruning rate. Please note that this phenomenon does *not* apply to structured pruning. To better exploit this phenomenon and achieve even higher storage/computation reduction for non-structured pruning (plus quantization), we leverage the state-of-art ternary quantization technique [45] with dedicated optimizations. We apply this technique for weight quantization after non-structured pruning in cases when it outperforms our proposed method, thereby providing enough opportunity to non-structured weight pruning.

VII. COMPARISON OF NON-STRUCTURED AND STRUCTURED WEIGHT PRUNING

Due to space limitation, we focus on CONV layers, which are the most computationally intensive layers in DNNs and are becoming the major storage in state-of-art ResNet and MobileNet models. We do observe similar (and more significant) effect on FC layers and on RNNs, without providing detailed results due to space.

TABLE VIII
COMPARISON ON NON-STRUCTURED VS. STRUCTURED PRUNING USING ALEXNET ON IMAGENET DATASET

	Method	Top-5 Accuracy	CONV Prune Rate	CONV No. of Weights	CONV Quant Bits	CONV Weight Store	Index Bits	Weight+Index Storage (Relative)	Weight+Index Storage (Absolute)	CONV Compress Rate
Baseline AlexNet		82.2%	1.0×	2.3M	32	9.3MB	-	9.3MB	9.3MB	1.0×
Non-structured	Han [63]	80.3%	2.7×	0.86M	8	0.86MB	4	1.3MB	N/A	7.1×
	Dyn. surg. [21]	80.0%	3.1×	0.74M	N/A	N/A	N/A	N/A	N/A	N/A
	Nest [19]	80.3%	3.23×	0.71M	N/A	N/A	N/A	N/A	N/A	N/A
	Fine-grained [46]	80.3%	4.16×	0.55M	N/A	N/A	N/A	N/A	N/A	N/A
	our's	81.9%	**11.2×**	0.3M	7	0.26MB	6	0.51MB	0.61MB	**25.5×**
Structured	SSL [24]	80.4%	1.4×	1.6M	N/A	N/A	-	N/A	N/A	N/A
	Taylor [64]	79.8%	2.5×	0.92M	N/A	N/A	-	N/A	N/A	N/A
	NISP [65]	80.2%	1.9×	1.2M	N/A	N/A	-	N/A	N/A	N/A
	our's	81.8%	**5.1×**	0.65M	7	0.56MB	-	0.56MB	0.56MB	**23.3×**

TABLE IX
COMPARISON ON NON-STRUCTURED VS. STRUCTURED PRUNING USING RESNET-18 (RESNET-50 IN PRIOR WORK NISP AND THINET, WITH STARTING TOP-5 ACCURACY 91.1%), IMAGENET DATASET

	Method	Top-5 Accuracy	CONV Prune Rate	CONV No. of Weights	CONV Quant Bits	CONV Weight Store	Index Bits	Weight+Index Storage (Relative)	Weight+Index Storage (Absolute)	CONV Compress Rate
Baseline ResNet-18		89.1%	1.0×	11.2M	32	44.7MB	-	44.7MB	44.7MB	1.0×
Non-structured	our's	89.1%	**6.4×**	1.75M	6	1.32MB	5	2.47MB	3.11MB	**18.1×**
	our's	87.9%	**8.9×**	1.26M	6	0.94MB	5	1.89MB	2.29MB	**23.6×**
Structured	DCP [66]	87.6%	2×	5.7M	N/A	N/A	-	N/A	N/A	N/A
	DCP [66]	85.7%	3.3×	3.5M	N/A	N/A	-	N/A	N/A	N/A
	ThiNet-50 [67]	90.7%	2×	12.8M	N/A	N/A	-	N/A	N/A	N/A
	ThiNet-30 [67]	88.3%	3.3×	7.7M	N/A	N/A	-	N/A	N/A	N/A
	NISP [65]	90.2%	1.8×	14.2M	N/A	N/A	-	N/A	N/A	N/A
	our's	89.1%	**2.5×**	4.46M	6	3.34MB	-	3.34MB	3.34MB	**13.4×**
	our's	87.8%	**4.3×**	2.60M	6	1.95MB	-	1.95MB	1.95MB	**22.9×**

TABLE X
COMPARISON ON NON-STRUCTURED VS. STRUCTURED PRUNING USING MOBILENET-V2 ON IMAGENET DATASET

	Method	Top-1 Accuracy	CONV Prune Rate	CONV No. of Weights	CONV Quant Bits	CONV Weight Store	Index Bits	Weight+Index Storage (Relative)	Weight+Index Storage (Absolute)	CONV Compress Rate
Baseline MobileNet-V2		71.6%	1.0×	2.19M	32	8.77 MB	-	8.77 MB	8.77 MB	1.0×
Non-Structured	our's	70.7%	2×	1.09M	6	0.82 MB	5	1.65 MB	2.01 MB	**5.3×**
Structured	AMC [68]	70.8%	1.4×	1.55M	N/A	N/A	-	N/A	N/A	N/A
	CA MBv2 [69]	69.7%	1.27×	1.73M	N/A	N/A	-	N/A	N/A	N/A
	our's	70.0%	**1.7×**	1.29M	6	0.97 MB	-	0.97 MB	0.97 MB	**9.0×**

As discussed in Section V, our implementations are based on PyTorch with high baseline accuracies. We limit the number of epochs in both structured pruning and non-structured pruning to be 240 (much lower than the iterative pruning heuristic [18]), and the number of epochs in weight quantization to be 120. We adopt hyperparameter determination heuristic discussed in Section IV-D for both structured and non-structured pruning.

For non-structured weight pruning, we show results on both CSR with relative indices and with absolute indices. The former is more appropriate for storage reduction, but the latter achieves higher computation efficiency. For absolute indices we assume $4K = 64 \times 64$ blocks that are reasonable for hardware [42]. Besides the comparison between two pruning schemes, our results also consistently outperform prior work, in terms of both non-structured and structured pruning, as well as combination with weight quantization.

A. Comparison Results on ImageNet Dataset

Table VIII, Table IX and Table X demonstrate the comparison using AlexNet, ResNet-18 and MobileNet-V2 models on ImageNet dataset. In these tables, "CONV Prune Rate" refers to the reduction ratio in the number of weights in overall CONV layers, and the number of remaining weights is "CONV No. of Weights". "CONV Quant Bits" refers to the

number of bits used for equal-distance weight quantization, while "CONV Weight Store" is the storage required only for weights (not account for indices). "Index Bits" refers to the number of bits in CSR with relative indices. In our results, we already optimized this index bit value to minimize the overall storage (accounting for the additional dummy zeros as well). The next two columns refer to the total storage size accounting for relative indices and absolute indices, respectively. For structured pruning, they are the same as weight storage. The final column "CONV Compress Rate" refers to the storage compression rate compared with the original baseline DNN model without compression, assuming relative indices that are more favorable to non-structured pruning. We use "N/A" if the specific prior work only focuses on weight pruning without performing quantization.

It can be observed that we achieve significant pruning rate gains for both non-structured and structured pruning. Especially for structured pruning, we achieve 5.1× and 2.5× structured weight pruning in CONV layers of AlexNet and ResNet-18 models, respectively, without accuracy loss. We further achieve 4.3× structured pruning with minor accuracy loss around 1%. For ResNet on ImageNet dataset, it is difficult for prior work to achieve lossless structured pruning. For example, [26] results in 1% accuracy loss with 2× structured pruning, on ResNet-50 model with more redundancy. For MobileNet-

TABLE XI
COMPARISON ON NON-STRUCTURED VS. STRUCTURED PRUNING USING VGG-16 ON CIFAR-10 DATASET

	Method	Accuracy	CONV Prune Rate	CONV No. of Weights	CONV Quant Bits	CONV Weight Store	Index Bits	Weight+Index Storage (Relative)	Weight+Index Storage (Absolute)	CONV Compress Rate
Baseline VGG-16		93.7%	1.0×	14.7M	32	58.8MB	-	58.8MB	58.8MB	1.0×
Non-Structured	Iter. prun. [18], [70]	92.2%	2×	≈7.4M	N/A	N/A	-	N/A	N/A	N/A
	One-shot prun. [70]	92.4%	2.5×	≈5.9M	N/A	N/A	-	N/A	N/A	N/A
	our's	93.1%	**57.4×**	0.26M	5	0.16MB	7	0.54MB	0.72MB	**109×**
Structured	2PFPCE [71]	92.8%	4×	3.7M	N/A	N/A	-	N/A	N/A	N/A
	2PFPCE [71]	91.0%	8.3×	1.8M	N/A	N/A	-	N/A	N/A	N/A
	ConvNet [72]	93.4%	2.7×	5.3M	N/A	N/A	-	N/A	N/A	N/A
	our's	93.1%	**50.0×**	0.29M	5	0.18MB	-	0.18MB	0.18MB	**327×**

TABLE XII
COMPARISON RESULTS ON NON-STRUCTURED VS. STRUCTURED PRUNING USING RESNET-18 (RESNET-50 IN PRIOR WORK AMC AND RESNET-56 IN PRIOR WORK NISP) ON CIFAR-10 DATASET

	Method	Accuracy	CONV Prune Rate	CONV No. of Weights	CONV Quant Bits	CONV Weight Store	Index Bits	Weight+Index Storage (Relative)	Weight+Index Storage (Absolute)	CONV Compress Rate
Baseline ResNet-18		93.9%	1.0×	11.2M	32	44.6MB	-	44.6MB	44.6MB	1.0×
Non-Structured	our's	93.3%	**69.0×**	0.16M	5	0.10MB	8	0.33MB	0.53MB	**135×**
Structured	AMC [68]	93.5%	1.7×	N/A	N/A	N/A	-	N/A	N/A	N/A
	NISP [65]	93.2%	1.7×	N/A	N/A	N/A	-	N/A	N/A	N/A
	our's	93.3%	**59.8×**	0.19M	5	0.12MB	-	0.12MB	0.12MB	**372×**

TABLE XIII
COMPARISON RESULTS ON NON-STRUCTURED VS. STRUCTURED PRUNING USING MOBILENET-V2 ON CIFAR-10 DATASET

	Method	Accuracy	CONV Prune Rate	CONV No. of Weights	CONV Quant Bits	CONV Weight Store	Index Bits	Weight+Index Storage (Relative)	Weight+Index Storage (Absolute)	CONV Compress Rate
Baseline MobileNet-V2		95.1%	1.0×	2.2M	32	9.0MB	-	9.0MB	9.0MB	1.0×
Non-Structured	our's	94.9%	**6.1×**	0.37M	4	0.19MB	4	0.48MB	0.55MB	**18.8×**
Structured	DCP [66]	94.7%	1.3×	1.68M	N/A	N/A	-	N/A	N/A	N/A
	our's	95.1%	**4.9×**	0.45M	4	0.23MB	-	0.23MB	0.23MB	**39.2×**

TABLE XIV
COMPARISON ON NON-STRUCTURED VS. STRUCTURED PRUNING USING LENET-5 ON MNIST DATASET

	Method	Accuracy	CONV Prune Rate	CONV No. of Weights	CONV Quant Bits	CONV Weight Store	Index Bits	Weight+Index Storage (Relative)	Weight+Index Storage (Absolute)	CONV Compress Rate
Baseline LeNet-5		99.2%	1.0×	25.5K	32	102KB	-	102KB	102KB	1.0×
Non-structured	Han [63]	99.2%	7.7×	3.33K	8	3.33KB	5	7.0KB	N/A	14.5×
	our's	99.0%	**114.3×**	223	3	0.08KB	8	0.39KB	0.93KB	**262×**
Structured	SSL [24]	99.0%	26.1×	975	N/A	N/A	-	N/A	N/A	N/A
	our's	99.0%	**87.9×**	290	3	0.11KB	-	0.11KB	0.11KB	**944×**

V2 on ImageNet dataset, the uncompressed model training is already very challenging, which implies pruning such network is even harder. But we still achieve 1.7× structured pruning rate with only 0.7% accuracy loss.

When comparing non-structured vs. structured pruning, the overall CONV compression rate is comparable for the AlexNet case and the 1% accuracy loss case for ResNet-18. For the lossless case in ResNet-18, non-structured pruning is slightly better in storage, especially when relative indices are utilized. This is because the number of bits for indexing is relatively small in this case, and the slight benefit will diminish if certain accuracy loss is tolerable. The occasional gain cannot outweigh the difficulty in hardware support of non-structured sparsity. It would be difficult to choose non-structured pruning over the other one even if the storage results are comparable.

B. Comparison Results on CIFAR-10 Dataset

Table XI and Table XII demonstrate the comparison results using VGG-16 and ResNet-18 models on CIFAR-10 dataset. We observe that very significant pruning rates can be achieved compared with prior work (over 35× improvement in certain case). We investigated deeper and found that the underlying reason is the CIFAR-10 dataset itself, in that it is both "simple" and "difficult". "Simple" means that the input image scale is small and the number of classes is only 10; while "difficult" means that input images are blurred and feature extraction is not straightforward. As a result, researchers tend to migrate large-scale DNN models originally designed for ImageNet, such as VGG-16 and ResNet-18 (prior work even used ResNet-50). Consequently, there is significant margin of model compression, which can be exploited in the proposed systematic framework but difficult for heuristic methods.

Another observation is that non-structured pruning has only marginal gain in pruning rates (reduction in the number of weights) compared with structured one. Our hypothesis is that it is due to the high search space in non-structured pruning. Together with the large number of index bits due to high pruning rates, non-structured pruning is not preferable compared with structured one considering total storage size. The storage size gap is becoming surprisingly large when absolute indices are utilized.

Table XIII demonstrates the comparison results using MobileNet-V2 model on CIFAR-10 dataset. MobileNet is already compact and relatively difficult for further weight

pruning, but we still achieve 5× structured pruning along with 4-bit quantization. Again non-structured pruning only shows minor gain in weight reduction, and it is not preferable considering indexing overheads.

C. Comparison Results on MNIST Dataset

Table XIV demonstrates the comparison results using LeNet-5 model on MNIST data set. It is a simple dataset, and we achieve 87.9× structured pruning on CONV layers, together with 3-bit quantization. Non-structured pruning is again not preferred due to the high index bit and marginal increase in weight reduction rate. Ironically, it results in multiple times the amount of storage compared with structured pruning, when weight quantization is in place.

D. Comparison on Computation Efficiency

We have shown that non-structured pruning is not preferable in terms of storage even assuming the storage-friendly CSR format with relative indices, not to mention absolute indices. Based on our methodology, we find that computation efficiency shows the similar trend.

As discussed before, structured pruning will have higher computation efficiency if it achieves more than 37% in the pruning rate as non-structured pruning. In all our testing, the ratio between weight pruning rates of structured vs. non-structured pruning ranges from 40% to 87%, with a large variation but consistently higher than 37%. Even for the 40% case, the choice is clear considering the difficulty in hardware design for non-structured sparsity. As a result, we draw the final conclusion that non-structured weight pruning is in general not preferred compared with structured pruning across different platforms, application scenarios, DNN types, etc.

VIII. CONCLUSION

Non-structured and structured weight pruning and weight quantization are major methods for model compression, but the interaction among different techniques are never clearly understood. This paper is the first to investigate the value of non-structured and structured DNN weight pruning, when the weight quantization is in place. We build ADMM-NN-S, a joint weight pruning and quantization framework with algorithmic supports for structured pruning, dynamic ADMM regulation, and masked mappling and retraining. To perform fair and fundamental comparison between non-structured and structured pruning in an implementation-agnostic manner, we propose a methodology that captures storage overhead and computation efficiency. We perform extensive and representative testing of ADMM-NN-S with AlexNet, VGGNet, ResNet-18/50, MobileNet, and LeNet-5 models based on ImageNet, CIFAR-10, and MNIST data sets. We show that ADMM-NN-S can significant outperform the state-of-the-art results for non-structured pruning with quantization. More importantly, for the first time we show that with quantization in place and the same accuracy, structured pruning is more preferable than non-structured pruning in terms of computation efficiency.

REFERENCES

[1] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, and N. S. Kim, "A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 175–188, IEEE, 2018.

[2] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 243–254, IEEE, 2016.

[3] M. Buckler, P. Bedoukian, S. Jayasuriya, and A. Sampson, "Eva^2: Exploiting temporal redundancy in live computer vision," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 533–546, IEEE, 2018.

[4] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. W. Fletcher, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pp. 674–687, IEEE Press, 2018.

[5] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, "Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 405–418, 2017.

[6] R. Cai, A. Ren, N. Liu, C. Ding, L. Wang, X. Qian, M. Pedram, and Y. Wang, "Vibnn: Hardware acceleration of bayesian neural networks," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 476–488, ACM, 2018.

[7] X. Wei, Y. Liang, P. Zhang, C. H. Yu, and J. Cong, "Overcoming data transfer bottlenecks in dnn accelerators via layer-conscious memory managment," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 120–120, ACM, 2019.

[8] J. Zhang and J. Li, "Unleashing the power of soft logic for convolutional neural network acceleration via product quantization," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 120–120, ACM, 2019.

[9] S. Zeng, Y. Lin, S. Liang, J. Kang, D. Xie, Y. Shan, S. Han, Y. Wang, and H. Yang, "A fine-grained sparse accelerator for multi-precision dnn," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 185–185, ACM, 2019.

[10] H. Nakahara, A. Jinguji, M. Shimoda, and S. Sato, "An fpga-based fine tuning accelerator for a sparse cnn," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 186–186, ACM, 2019.

[11] L. Lu, Y. Liang, R. Huang, W. Lin, X. Cui, and J. Zhang, "Speedy: An accelerator for sparse convolutional neural networks on fpgas," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 187–187, ACM, 2019.

[12] Z. Tang, G. Luo, and M. Jiang, "Ftconv: Fpga acceleration for transposed convolution layers in deep neural networks," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 189–189, ACM, 2019.

[13] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[14] M. A. Rumi, X. Ma, Y. Wang, and P. Jiang, "Accelerating sparse cnn inference on gpus with performance-aware weight pruning," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, pp. 267–278, 2020.

[15] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks," in *Proceedings of the 35th International Conference on Computer-Aided Design*, p. 12, ACM, 2016.

[16] http://www.techradar.com/news/computing-components/processors/google-s-tensor-processing-unit-explained-\this-is-what-the-future-of-computing-looks-\like-1326915.

[17] https://www.sdxcentral.com/articles/news/intels-deep-learning-chips-will-arrive-2017/2016/11/.

[18] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, pp. 1135–1143, 2015.

[19] X. Dai, H. Yin, and N. K. Jha, "Nest: A neural network synthesis tool based on a grow-and-prune paradigm," *IEEE Transactions on Computers*, 2019.

[20] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *arXiv preprint arXiv:1611.05128*, 2016.

[21] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.

[22] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Advances in Neural Information Processing Systems*, pp. 4857–4867, 2017.

[23] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates," in *AAAI*, pp. 4876–4883, 2020.

[24] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

[25] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pp. 548–560, IEEE, 2017.

[26] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 1398–1406, IEEE, 2017.

[27] C. Leng, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," *arXiv preprint arXiv:1707.09870*, 2017.

[28] E. Park, J. Ahn, and S. Yoo, "Weighted-entropy-based quantization for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7197–7205, 2017.

[29] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," in *International Conference on Learning Representations (ICLR)*, 2017.

[30] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, pp. 2849–2858, 2016.

[31] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4820–4828, 2016.

[32] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, pp. 525–542, Springer, 2016.

[33] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, pp. 4107–4115, 2016.

[34] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, pp. 3123–3131, 2015.

[35] https://www.tensorflow.org/mobile/tflite/.

[36] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch," 2017.

[37] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "ADMM-NN: An Algorithm-Hardware Co-Design Framework of DNNs Using Alternating Direction Methods of Multipliers," in *ASPLOS*, pp. 925–938, ACM, 2019.

[38] H. Ouyang, N. He, L. Tran, and A. Gray, "Stochastic alternating direction method of multipliers," in *International Conference on Machine Learning*, pp. 80–88, 2013.

[39] T. Suzuki, "Dual averaging and proximal gradient descent for online alternating direction multiplier method," in *International Conference on Machine Learning*, pp. 392–400, 2013.

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

[41] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[42] Z. Yuan, J. Yue, H. Yang, Z. Wang, J. Li, Y. Yang, Q. Guo, X. Li, M.-F. Chang, H. Yang, *et al.*, "Sticker: A 0.41-62.1 tops/w 8bit neural network processor with multi-sparsity compatible convolution arrays and online

tuning acceleration for fully connected layers," in *2018 IEEE Symposium on VLSI Circuits*, pp. 33–34, IEEE, 2018.

[43] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, p. 20, IEEE Press, 2016.

[44] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 27–40, ACM, 2017.

[45] Z. He and D. Fan, "Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation," *arXiv preprint arXiv:1810.01018*, 2018.

[46] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the regularity of sparse structure in convolutional neural networks," *arXiv preprint arXiv:1705.08922*, 2017.

[47] J. Jang, K. Choi, and H. Yang, "A simd-aware pruning technique for convolutional neural networks with multi-sparsity levels: work-in-progress," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis Companion*, pp. 1–2, 2019.

[48] X. Ma, F.-M. Guo, W. Niu, X. Lin, J. Tang, K. Ma, B. Ren, and Y. Wang, "Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices," *AAAI*, 2020.

[49] X. Ma, W. Niu, T. Zhang, S. Liu, S. Lin, H. Li, W. Wen, X. Chen, J. Tang, K. Ma, *et al.*, "An image enhancing pattern-based sparsity for real-time inference on mobile devices," in *European Conference on Computer Vision (ECCV)*, pp. 629–645, 2020.

[50] https://engineering.fb.com/ml-applications/fbgemm/.

[51] M. Hong, Z.-Q. Luo, and M. Razaviyayn, "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 337–364, 2016.

[52] S. Liu, J. Chen, P.-Y. Chen, and A. Hero, "Zeroth-order online alternating direction method of multipliers: Convergence analysis and applications," in *International Conference on Artificial Intelligence and Statistics*, pp. 288–297, 2018.

[53] B. Moons, D. Bankman, and M. Verhelst, *Embedded Deep Learning*. Springer, 2019.

[54] S. Wiedemann, K.-R. Müller, and W. Samek, "Compact and computationally efficient representation of deep neural networks," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 3, pp. 772–785, 2019.

[55] S. Wiedemann, H. Kirchhoffer, S. Matlage, P. Haase, A. Marban, T. Marinč, D. Neumann, T. Nguyen, H. Schwarz, T. Wiegand, *et al.*, "Deepcabac: A universal compression algorithm for deep neural networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 700–714, 2020.

[56] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[57] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[58] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[60] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[61] H.-P. Cheng, Y. Huang, X. Guo, Y. Yan, Y. Huang, W. Wen, H. Li, and Y. Chen, "Differentiable fine-grained quantization for deep neural network compression," in *NIPS 2018 CDNNRIA Workshop*, 2018.

[62] Google supercharges machine learning tasks with TPU custom chip, https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html.

[63] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *International Conference on Learning Representations (ICLR)*, 2016.

[64] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[65] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.

[66] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 875–886, 2018.

[67] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017.

[68] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[69] E. Elsen, M. Dukhan, T. Gale, and K. Simonyan, "Fast sparse convnets," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14629–14638, 2020.

[70] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *ICLR*, 2019.

[71] C. Min, A. Wang, Y. Chen, W. Xu, and X. Chen, "2pfpce: Two-phase filter pruning based on conditional entropy," *arXiv preprint arXiv:1809.02220*, 2018.

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

Zhezhi He Zhezhi He (IEEE Member) received the Ph.D. degree in electrical engineering from Arizona State University, Tempe, AZ, USA, in 2020. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. His primary research interests include neuromorphic computing, efficient and secure deep learning, in-memory computing, and electronic design automation.


Linfeng Zhang is an Ph.D. student in in Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University supervised by Kaisheng Ma. He publishes papers on conferences including NeurIPS, ICLR, CVPR, ICCV etc. His research interests includes computer vision, deep neural network compression and acceleration.

Xiaolong Ma received his M.S. degree in Electrical Engineering from Syracuse University in 2016. He is currently a Ph.D. candidate in Computer Engineering in the Department of Electrical and Computer Engineering at Northeastern University, supervised by Dr. Yanzhi Wang. His research interests include machine learning algorithm, deep learning system design, high performance computing and computer vision.

Geng Yuan received the M.S. of Electrical Engineering from Syracuse University in 2016. He is currently pursuing a Ph.D. degree in the Department of Electrical & Computer Engineering at Northeastern University supervised by Dr. Yanzhi Wang. His research interests include high-performance and energy-efficient deep learning system design, computer vision, deep learning architecture design with emerging technologies.
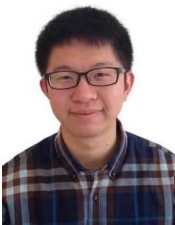
Sheng Lin received the B.S. degree from Zhejiang University, Hangzhou, China, in 2013, the M.S. degree from Syracuse University, Syracuse, USA, in 2015, and the Ph.D. degree in Computer Engineering from Northeastern University, Boston, USA, in 2020, under the supervision of Prof. Yanzhi Wang. His current research interests include privacy preserving machine learning, energy-efficient artificial intelligence systems, computer visioin, model compression and mobile acceleration of deep learning applications.

Sia Huat Tan is currently pursuing an M.S. degree at IIIS, Tsinghua University. He received his B.S. degree in Computer Science and Information Engineering from National Cheng Kung University in 2019 with the top rank in the grade level. He is interested in computer vision and currently working on the design of neural network models to improve the efficiency and robustness against common corruptions and adversarial examples. He was born in Sarawak, Malaysia.

Shaokai Ye received the B.S degree in computer engineering from Saint Louis University in 2015 and the M.S degree in computer engineering under the supervision of Dr. Yanzhi Wang in 2018 from Syracuse University. He is currently pursuing the Ph.D degree in EPFL under the supervision of Dr. Mackenzie Mathis. His research interest is making AI more efficient and robust.
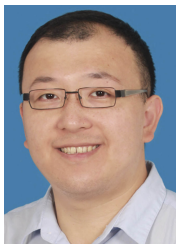

Zhengang Li received the B.E. degree in electronic information engineering from Zhejiang University, Zhejiang, China, in 2017. He is currently pursuing the Ph.D. degree under Professor Yanzhi Wang in computer engineering, Northeastern University, Boston, MA, U.S. His current research interests include model compression of deep neural networks, machine learning algorithm and computer vision.

Deliang Fan is currently an Assistant Professor in the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA. Before joining ASU in 2019, he was an assistant professor in Department of Electrical and Computer Engineering at University of Central Florida, Orlando, FL, USA. He received his M.S. and Ph.D. degrees, in Electrical and Computer Engineering from Purdue University, West Lafayette, IN, USA, in 2012 and 2015, respectively. Dr. Fan's primary research interests include Energy Efficient and High Performance Big Data Processing-In-Memory Circuit, Architecture and Algorithm, with applications in Deep Neural Network, Data Encryption, Graph Processing and Bioinformatics Acceleration-in-Memory system; Hardware-aware deep learning optimization; Brain-inspired (Neuromorphic) Computing; AI security. He has authored and co-authored 110+ peer-reviewed international journal/conference papers in above area. He is the receipt of best paper award of 2019 ACM Great Lakes Symposium on VLSI, 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), and 2017 IEEE ISVLSI. He is also the technical area chair of DAC 2021, GLSVLSI 2019/2020/2021, ISQED 2019/2020/2021, and the financial chair of ISVLSI 2019. Please refer to https://dfan.engineering.asu.edu/ for more details.

Kaisheng Ma is a tenure-track assistant professor of computer science in Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University. He leads the Algorithms, Architecture and Chipset Lab (ArChip). His research interests lie in the interdisciplinary fields of human-machine interface (HMI), brain-spired AI algorithm design, computer vision for self-driving, compact model design, computer architecture and microelectronics. He believes in vertical integration for systems, with a special emphasis on implanted medical and neural devices, and chip level system solutions for self-driving systems. He received his Ph.D. in Computer Science and Engineering at the Pennsylvania State University, following Professor Vijaykrishnan Nanrayanan @ Penn State, Yuan Xie @ UCSB, Jack Sampson @ Penn State. His previous research on Nonvolatile Processor and Energy Harvesting won 2018 EDAA Best Dissertation Award. He publishes papers on conferences including NeuralPS, ICCV, AAAI, CVPR, ISCA, ASPLOS, MICRO, HPCA, DAC etc. He has won many awards, including: 2015 HPCA Best Paper Award, 2016 IEEE MICRO Top Picks, 2017 ASP-DAC Best Paper Award. 2018 EDAA Best Dissertation Award. Dr. Ma has many honors, including 2016 Penn State CSE Department Best Graduate Research, 2016 Cover Feature of NSF ASSIST Engineering Research Center, 2011 Yang Fuqing & Wang Yangyuan Academician Scholarship.

Xuehai Qian received the BSc degree from Beihang University, China, in 2004, the MSc degree from the Institute of Computing Technology, Chinese Academy of Sciences, China, in 2007, and the PhD degree from the Computer Science Department, University of Illinois at Urbana-Champaign, in 2013. Currently, he is an assistant professor with the Ming Hsieh Department of Electrical Engineering and the Department of Computer Science, University of Southern California. His research interests include the fields of computer architecture, architectural support for programming productivity, and correctness of parallel programs.
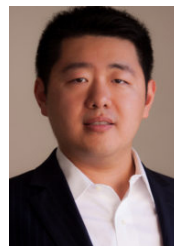
Xue Lin is an assistant professor in the Department of Electrical and Computer Engineering at Northeastern University since 2017. She received her bachelor's degree in Microelectronics from Tsinghua University, China and her PhD degree from the Department of Electrical Engineering at University of Southern California in 2016. Her research interests include deep learning security and hardware acceleration, machine learning and computing in cyber-physical systems, high-performance and mobile cloud computing systems, and VLSI. Her research work has been recognized by several NSF awards and supported by Air Force Research Lab, Office of Naval Research, Lawrence Livermore National Lab, and DARPA. She got the best paper award at ISVLSI 2014, the top paper award at CLOUD 2014, the best technical poster award at NDSS 2020, the spotlight paper at ECCV 2020, and the 1st place at ISLPED 2020 Design Contest.

Yanzhi Wang received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2009, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 2014. His research interests include energy-efficient and high-performance implementations of deep learning and artificial intelligence systems, emerging deep learning algorithms/systems, generative adversarial networks, and deep reinforcement learning.