A Privacy-Preserving-Oriented DNN Pruning and Mobile Acceleration Framework

Yifan Gong[1], Zheng Zhan[1], Zhengang Li[1], Wei Niu[2], Xiaolong Ma[1], Wenhao Wang[4], Bin Ren[2], Caiwen Ding[3], Xue Lin[1], Xiaolin Xu[4], and Yanzhi Wang[1]
[1]Northeastern University [2]The College of William and Mary [3]University of Connecticut [4]University of Illinois at Chicago
{gong.yifa,zhan.zhe,li.zhen,ma.xiaol,xue.lin,yanz.wang}@northeastern.edu
wniu@email.wm.edu,bren@cs.wm.edu,caiwen.ding@uconn.edu,{wwang208,xiaolin8}@uic.edu

ABSTRACT

Weight pruning of deep neural networks (DNNs) has been proposed to satisfy the limited storage and computing capability of mobile edge devices. However, previous pruning methods mainly focus on reducing the model size and/or improving performance without considering the privacy of user data. To mitigate this concern, we propose a privacy-preserving-oriented pruning and mobile acceleration framework that does not require the private training dataset. At the algorithm level of the proposed framework, a systematic weight pruning technique based on the alternating direction method of multipliers (ADMM) is designed to iteratively solve the pattern-based pruning problem for each layer with randomly generated synthetic data. In addition, corresponding optimizations at the compiler level are leveraged for inference accelerations on devices. With the proposed framework, users could avoid the time-consuming pruning process for non-experts and directly benefit from compressed models. Experimental results show that the proposed framework outperforms three state-of-art end-to-end DNN frameworks, i.e., TensorFlow-Lite, TVM, and MNN, with speedup up to 4.2×, 2.5×, and 2.0×, respectively, with almost no accuracy loss, while preserving data privacy.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; • **Security and privacy → Privacy protections**; • **Software and its engineering → Source code generation**; • **Human-centered computing → Mobile computing**.

KEYWORDS

Model compression; Pattern-based pruning; Privacy-preserving; Real-time mobile acceleration

ACM Reference Format:
Yifan Gong[1], Zheng Zhan[1], Zhengang Li[1], Wei Niu[2], Xiaolong Ma[1], Wenhao Wang[4], Bin Ren[2], Caiwen Ding[3], Xue Lin[1], Xiaolin Xu[4], and Yanzhi Wang[1]. 2020. A Privacy-Preserving-Oriented DNN Pruning and Mobile Acceleration Framework. In *Proceedings of the Great Lakes Symposium on VLSI 2020 (GLSVLSI '20), September 7–9, 2020, Virtual Event, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3386263.3407650

1 INTRODUCTION

Recent years have witnessed substantial progress and remarkable breakthroughs of deep neural networks (DNNs), especially deep convolutional neural networks (CNNs) in solving complicated visual tasks [1–3]. Along with the great success are the ever-increasing model size and the computing demand, which highly restrict the deployments of DNNs on mobile and edge devices with limited capacities. To mitigate the challenges brought by the large amount of computations and achieve the goal of real-time inference for modern DNN models, weight pruning [4–17] is proposed to reduce the inherent redundancy in model parameters. Early works on non-structured pruning [4–6] prune weights at arbitrary locations using heuristic methods. Via the successful applications of the powerful Alternating Direction Methods of Multipliers (ADMM) optimization framework, later research works [7, 8] achieved substantial weight reduction while maintaining promising accuracy. However, non-structured pruning leads to sparse and irregular weight matrices, which require additional indices for the storage in a compact format. Consequently, these methods are not compatible with parallel hardware accelerations for the inference. By incorporating regularity into weight pruning, structured pruning [9–15] eliminates the requirements for weight indices, thus is more hardware friendly. On the downside, the coarse-grained nature of structured pruning degrades the accuracy more significantly. Recently, pattern-based pruning [16, 17] is proposed to inherit the benefits from fine-grained pruning while maintaining structures that can be exploited for hardware accelerations.

Although the above-mentioned weight pruning techniques differ in sparsity schemes and pruning algorithms, most of them [5–17] are based on the assumption that the training dataset is available. However, this is not always the use case for real-world implementations. For example, in many areas, most notably those related to medicine, sharing data about individuals is not even permitted by law or regulations [18, 19]. Furthermore, in commercial applications, the training data should be kept as business confidentiality.

To deal with this problem, we propose a privacy-preserving-oriented DNN pruning and mobile acceleration framework. At the algorithm level, a DNN model compression entity prunes the pre-trained models provided by users with pattern-based sparsity without the usage of any information about the private training dataset. Specifically, the pruning of the DNN model is achieved by pruning layers sequentially with randomly generated syntheic data. Instead of using the loss value, we measure the difference of the

Frobenius norm between the original output of user's pre-trained model and the output of the compressed model given the same input for each layer to evaluate whether enough information is maintained after pruning. By forming the pruning problem into an optimization problem, the proposed framework solves the pattern-based pruning problems iteratively and analytically by extending the potent ADMM algorithm [20]. At the compiler level, corresponding pattern-enabled compiler optimizations are leveraged. After retraining the compressed model, users can achieve real-time inference without accuracy loss. The highlights of our contributions in this paper are summarized as follows: 1) We formulate the privacy-preserving-oriented pattern-based pruning problem as an optimization problem with combinatorial constraints. 2) We solve the optimization problem with an extension of the ADMM framework. 3) We accelerate the DNN execution on mobile devices with a compiler-based framework consisting of several optimizations enabled by our pattern-based design. 4) We conduct extensive experiments to compare the proposed framework with the state-of-the-art pruning methods on representative CNNs.

2 RELATED WORK

In practice, the data used for DNN training is often massively distributed among different users, or is owned by a single party but is inconvenient or forbidden to share with others. On the one hand, users tend to store their confidential data locally for privacy concerns. On the other hand, many data owners, e.g., medical institutions, are prevented by regulations from sharing their data with others [18, 19]. Meanwhile, the demand for model compression on mobile devices is imperative because of the limited capacity nature.

Only few works achieve weight pruning without the original training dataset. The early weight pruning work [4] proposed a magnitude-based heuristic method. Only 10∼20% of weights can be pruned without hurting accuracy when no retraining is adopted. If a higher compression rate is desired, several rounds of pruning and retraining are needed. Work [21] only prunes fully-connected layers while neglecting computation intensive convolutional (CONV) layers. Recently, there are tools for data-free pruning that prunes small value weights directly, such as the one used in Cambricon AI chips. However, this approach suffers from notable accuracy loss even when removing only 20% weights.

To overcome the limitations of prior works and achieve real-time DNN inference, we consider to design a privacy-preserving-oriented DNN pruning and mobile acceleration framework that provides high-performance compressed model for users without the usage of the training dataset. With the proposed framework, users could directly benefit from the compressed model without privacy concerns and have no need to handle the time-consuming weight pruning process.

3 FRAMEWORK OVERVIEW

The overview of the proposed framework is presented in this section. We begin by introducing the threat model, then the adopted pattern-based sparsity. Next, the algorithm-level and compiler-level frameworks are demonstrated.

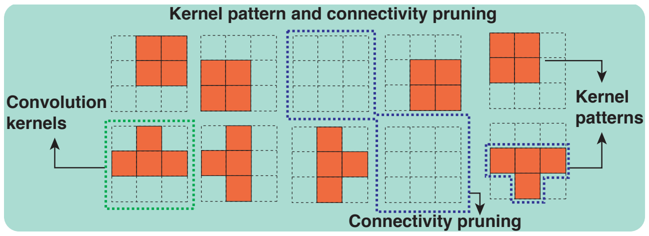

Figure 1: Pattern-based pruning.

3.1 Threat Model

We consider the following threat model in this work. Two participants, DNN compression entity and data owner, also known as the user, work together to formulate a DNN model with high performance on the private dataset of the user, as shown in Fig. 2. The DNN compression entity has no access to the original training data but just a pre-trained DNN model from the user. This work specifically focuses on protecting the privacy of user data. We assume that the pre-trained model is trained in a trusted way and its security is out of the scope of this paper.

3.2 Sparse Convolution Patterns

To incorporate the advantages of both structured pruning and non-structured pruning while getting rid of the respective shortcomings, our pruning framework adopts two pattern-based pruning dimensions, i.e., kernel pattern pruning and connectivity pruning. The objective is to achieve both high inference accuracy and satisfying execution efficiency. An illustration of pattern-based pruning is given in Fig. 1, with orange blocks representing remaining weights while green blocks are pruned weights.

Kernel pattern pruning removes weights at an intra-kernel level. The locations of the remaining weights in each kernel form a specific pattern. In this work, we focus on the kernel patterns for 3×3 kernels because they are widely adopted in various DNN architectures [2, 3]. Different kernels can have different patterns, but the total types of patterns are restricted to a pattern library with a fixed size. We represent the finite pattern library as $\mathcal{P} = \{M_1, \cdots, M_m\}$, with m representing the size of the pattern library. As Fig. 1 shows, we reserve four non-zero weights in a kernel to match the single-instruction multiple-data (SIMD) architecture of embedded CPU/GPU processors, thereby maximizing hardware throughput.

Connectivity Pruning achieves inter-kernel level pruning by removing whole kernels, as illustrated in Fig. 1. Connectivity pruning is a good supplement to kernel pattern pruning for a higher compression and acceleration rate. Both pruning schemes can be integrated into the same algorithm-level solution and compiler assisted acceleration framework.

3.3 Algorithm-Level Framework

Fig. 2 illustrates the algorithm-level framework. The DNN compression entity is responsible for the pruning of user's pre-trained model while the user only needs to retrain the compressed model with the help of a retraining function. In the DNN compression entity side, ADMM algorithm is leveraged to achieve pattern-based

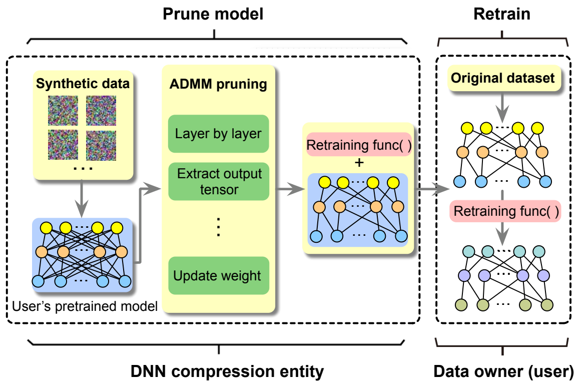

Figure 2: Algorithm-level framework overview.

pruning layer-by-layer systematically. Note that the DNN compression entity has no access to the user's training dataset but only the pre-trained model to preserve data privacy. Randomly generated synthetic data is used as the input for the pruning process. In the user side, retraining is similar to the traditional training of a DNN model, except that the retraining function sets corresponding gradients as zero for pruned weights. Only a few epochs are required before the model can make acceptable predictions. Therefore, users do not have to grasp details about pruning and can easily obtain compressed models with the proposed framework.

3.4 Compiler-Level Framework

After pattern-based pruning, we rely on a compiler-based acceleration framework to achieve real-time DNN executions on resource-restricted mobile devices, as shown in Fig. 3. This framework aims to address three key challenges in the pruned DNN execution: 1) heavy control-flow dependency existing within each thread; 2) computation divergence and load imbalance among different threads; 3) poor memory performance caused by irregular memory accesses. Correspondingly, we design three pattern-enabled compiler optimizations that work on each DNN layer: filter kernel reorder, compressed weight storage, and load redundancy elimination. These optimizations are conducted on a layer-wised representation that consists of multiple parts like layer shape, pattern style, connectivity information, etc. These optimizations are general, working for both CPU and GPU code generations.

Filter kernel reorder addresses two challenges i) increased control-flow instructions, and ii) thread divergence and load imbalance, by grouping the filters and kernels with similar lengths/patterns together. This optimization is specifically enabled by our pattern-based pruning design.

Compressed weight storage is specifically designed for our kernel pattern and connectivity pruning by leveraging the fact that the preserved weights follow our designed patterns. Together with filter kernel reordering, this compact data structure yields much better compression rates than the conventional CSR (compressed sparse row) format [22].

Load redundancy elimination [23] addresses the poor memory performance of non-structured pruning by exploring register-level load redundancy opportunities during executable kernel code generation. It is crucial when data accesses between memory and cache have already been optimized using tiling [24].

4 PRIVACY-PRESERVING-ORIENTED WEIGHT PRUNING

Our algorithm-level design is presented in this section. We first provide the formulation of the privacy-preserving-oriented pruning problem. Then the ADMM-based solution is demonstrated. Finally, the overall pruning algorithm is outlined.

4.1 Formulation of the Pruning Problem

We consider the pruning of an N-layer DNN with a major focus on the computation-intensive CONV layers. Our objective is to find specific sparse patterns for the kernels without the usage of the original dataset. The compressed model is then sent back to the user for retraining. For each layer n, the weights and biases are denoted as $\boldsymbol{W}_n$ and $\boldsymbol{b}_n$, respectively. During the pruning process, the input to the DNN is $\boldsymbol{X}_0$. Note that $\boldsymbol{X}_0$ is not derived from the original training dataset, but only randomly generated synthetic data, to preserve privacy for the original training dataset. In our experiment, we set the value of each pixel within the synthetic data with a discrete uniform distribution in the range of 0 to 255. Layer n takes the output $\boldsymbol{X}_{n-1}$ from the previous layer as the input volume, and produces an output volume $\boldsymbol{X}_n = \sigma(\boldsymbol{W}_n \boldsymbol{X}_{n-1} + \boldsymbol{b}_n)$, where σ is the activation function. In order to evaluate whether representative weights are kept to maintain enough information after pruning, we measure the Frobenius norm between the original output volume X'_n of user's pre-trained model and the output $\boldsymbol{X}_n$ of the compressed model given the same input volume $\boldsymbol{X}_{n-1}$. A smaller value means that the layer can provide similar results after pruning, indicating that more information is kept. Therefore, we could formulate the problem of the pruning for the n-th layer as

$$\begin{aligned} &\min_{\boldsymbol{W}_n, \boldsymbol{b}_n} \quad \left\| \sigma(\boldsymbol{W}_n \boldsymbol{X}_{n-1} + \boldsymbol{b}_n) - \boldsymbol{X}'_n \right\|_F^2, \\ &\text{subject to} \quad \boldsymbol{W}_n \in C_n, \end{aligned} \tag{1}$$

where C_n denotes the constraint set for the n-th layer. C_n restricts the pattern shapes and is defined as $C_n := \{\boldsymbol{W}_n |$ each kernel in $\boldsymbol{W}_n$ needs to satisfy one specific pattern shape in the pattern set $\mathcal{P}\}$.

4.2 Pattern Library (Set) Design

An appropriate design of the pattern library $\mathcal{P}$ is the prerequisite to attain both good pruning results and efficient hardware implementations. The design includes the size of the pattern library and the shape of each specific candidate pattern in the library. If the size of the pattern library is too small, the pruning might not be flexible enough, thereby leading to accuracy degradation. On the contrary, it is more challenging to generate efficient codes by the compiler for hardware accelerations and finding the sparsity solution with a large pattern library size. Through empirical study, we found out that a library with $m = 6 - 8$ patterns achieves a desirable balance between compiler overhead and accuracy for the commonly used 3×3 kernels.

After the size of the pattern library is settled and 4-entry patterns are utilized, the influence of patterns on the compiler and

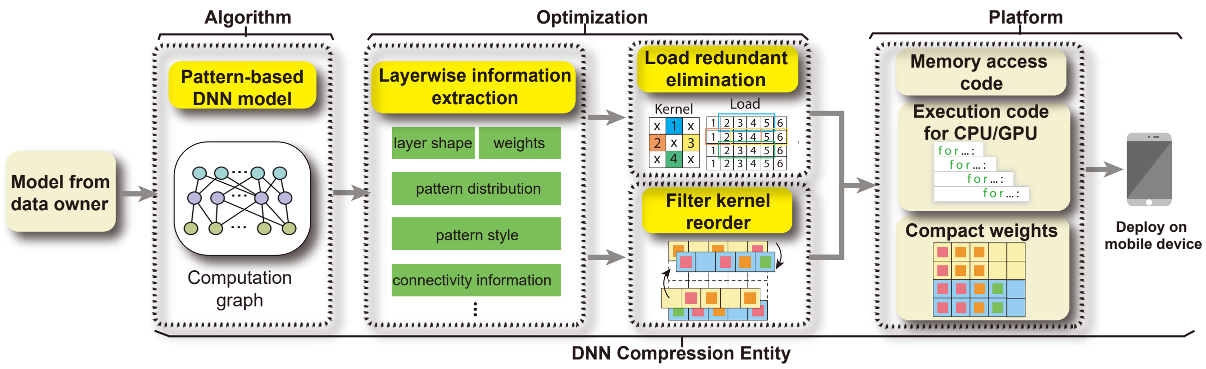

Figure 3: Compiler-level inference acceleration framework overview.

hardware is determined, regardless of specific pattern shapes. However, the pattern shapes will affect the accuracy of the compressed model and thus should be designed carefully. We select candidate pattern shapes using a simple but effective heuristic based on the following insights: 1) the central weight in a 3×3 kernel is critical and shall not be pruned; and 2) a smaller distortion of the kernel after pruning is preferred. Therefore, we find 3 largest weights for each kernel within the pre-trained model. The locations of the 3 largest weights together with the central weight form a 4-entry pattern. Then, top-m most commonly appeared patterns in the whole DNN are selected as the candidates, forming the pattern library $\mathcal{P} = \{\boldsymbol{M}_1, \cdots, \boldsymbol{M}_m\}$, where each $\boldsymbol{M}$ only contains binary-valued elements and has the same size as the kernels. Pattern pruning is achieved by applying element-wise multiplication of $\boldsymbol{M} \in \mathcal{P}$ with kernels. A found pattern library with 8 candidate patterns is illustrated in Fig. 1, with orange blocks representing the locations to maintain weights.

4.3 ADMM-based Kernel Pattern Pruning

Directly solving the optimization problem (1) is difficult due to the non-convex constraint. Hence, we resort to the ADMM framework by decomposing the original problem into two subproblems to be solved separately. To leverage the ADMM optimization framework, we define an indicator function $I_n(\boldsymbol{W}_n)$, that is zero when the constraint $\boldsymbol{W}_n \in C_n$ is satisfied, but $+\infty$ otherwise. After incorporating auxiliary variable $\boldsymbol{A}_n$, problem (1) can be rewritten as

$$\begin{aligned} &\min_{\boldsymbol{W}_n, \boldsymbol{b}_n} \ \left\|\sigma(\boldsymbol{W}_n \boldsymbol{X}_{n-1} + \boldsymbol{b}_n) - \boldsymbol{X}'_n\right\|_F^2 + \mathcal{I}_n(\boldsymbol{A}_n), \\ &\text{subject to} \quad \boldsymbol{W}_n = \boldsymbol{A}_n. \end{aligned} \tag{2}$$

Note that our ADMM-based method is different from previous work [7, 8] as we not only remove redundant weights, but also enforce certain regularity of the remaining weights in a kernel with the leverage of kernel patterns. The augmented Lagrangian [20] of the optimization problem (2) is given as

$$\begin{aligned} \mathcal{L}(\boldsymbol{W}_n, \boldsymbol{b}_n, \boldsymbol{A}_n, \boldsymbol{D}_n) &= \left\|\sigma(\boldsymbol{W}_n \boldsymbol{X}_{n-1} + \boldsymbol{b}_n) - \boldsymbol{X}'_n\right\|_F^2 \\ +\mathcal{I}_n(\boldsymbol{A}_n) &+ \frac{\rho}{2} \left\|\boldsymbol{W}_n - \boldsymbol{A}_n + \boldsymbol{D}_n\right\|_F^2 + \frac{\rho}{2} \left\|\boldsymbol{D}_n\right\|_2^F, \end{aligned} \tag{3}$$

where $\boldsymbol{D}_n$ is the dual variable. To solve the problem above, we decompose it into two subproblems. At iteration k, the first subproblem (primal problem) is

$$\min_{\boldsymbol{W}_n, \boldsymbol{b}_n} \ \left\|\sigma(\boldsymbol{W}_n \boldsymbol{X}_{n-1} + \boldsymbol{b}_n) - \boldsymbol{X}'_n\right\|_F^2 + \frac{\rho}{2} \left\|\boldsymbol{W}_n - \boldsymbol{A}_n^{k-1} + \boldsymbol{D}_n^{k-1}\right\|_F^2. \tag{4}$$

Both of these two terms are differentiable and this subproblem could be solved by standard solvers such as stochastic gradient descent (SGD) effectively.

The second subproblem (proximal problem) is given by

$$\min_{\boldsymbol{A}_n} \ \mathcal{I}_n(\boldsymbol{A}_n) + \frac{\rho}{2} \left\|\boldsymbol{W}_n^k - \boldsymbol{A}_n + \boldsymbol{D}_n\right\|_F^2. \tag{5}$$

As $\mathcal{I}_n(\cdot)$ is the indicator function of the constraint set C_n, the globally optimal solution of the second subproblem can be derived as [20]

$$\boldsymbol{A}_n^k = \prod_{C_n} (\boldsymbol{W}_n^k + \boldsymbol{D}_n^{k-1}), \tag{6}$$

where $\prod_{C_n}$ is the Euclidean projection of $\boldsymbol{W}_n^k + \boldsymbol{D}_n^{k-1}$ onto the constraint set C_n. The special structure of C_n allows us to find the optimal analytical solutions, which is to select the pattern resulting the pruned kernel with the largest Frobenius norm, for each kernel in the layer. Then the derived $\boldsymbol{A}_n^k$ is fed into the primal problem in the next iteration $k+1$.

Next, we update the dual variable $\boldsymbol{D}_n^k$ according to

$$\boldsymbol{D}_n^k := \boldsymbol{D}_n^{k-1} + \boldsymbol{W}_n^k - \boldsymbol{A}_n^k. \tag{7}$$

The above alternating optimization process then proceeds to the next iteration until convergence.

4.4 Connectivity Pruning

Besides the above-mentioned kernel pattern pruning, we could further adopt connectivity pruning into the proposed framework to achieve a higher compression rate for users demanding a faster inference speed. Connectivity pruning can be integrated into the same algorithm-level solution in Section 4.3. We further define a constraint set $C'_n := \{\boldsymbol{W}_n |$ the number of nonzero kernels is no more than $\beta_n\}$ for connectivity pruning, where β_n is a predetermined hyperparameter. By replacing C_n with C'_n in the problem formulation and ADMM-based solution framework , we could obtain the results for connectivity pruning.

```
Algorithm 1: Overall Pruning Algorithm
Input  : User's pre-trained DNN model $\{W_n^0\}_{n=1}^N$, total
         iteration $K$, augmented penalty $\rho$, batch size $T$,
         constraint set $C_n$ for $n = 1$ to $N$
Output : Compressed model $\{W_n^K\}_{n=1}^N$, the retraining
         function
1  Initialize $\{A_n^0\}_{n=1}^N \leftarrow \{W_n^0\}_{n=1}^N$, $\{D_n^0\}_{n=1}^N \leftarrow \mathbf{0}$ ;
2  for iteration $k \leftarrow 1$ to $K$ do
3      Randomly generate a batch of $T$ synthetic images;
4      for layer $n \leftarrow 1$ to $N$ do
5          Get the output of the $n$-th layer from the current
             model and the pre-trained model;
6          Update $W_n^k$ by solving problem (4) with standard
             solvers;
7          Update $A_n^k$ by solving problem (5) using Eqn. (6);
8          $D_n^k := D_n^{k-1} + W_n^k - A_n^k$;
9      end
10 end
11 Send the pruned model $\{W_n^K\}_{n=1}^N$ and the retraining function
     back to the client for retraining process;
```

4.5 Overall Algorithm

We formally present the overall pruning algorithm as Algorithm 1. The DNN compression entity starts pattern-based pruning upon it receives the pre-trained model from a user. At the beginning of each iteration k, a batch of T synthetic images are generated and preprocessed as the input for the pruning process. The pruning is going through layer-by-layer for the whole model. The generation of the synthetic images does not depend on any information about the user's private dataset. The pruning process iteratively solves the two subproblems (4) and (5) until convergence. At last, the compressed model and retraining function are released to the user for retraining.

5 EXPERIMENTAL RESULTS

In this section, we present the evaluations of our privacy-preserving-oriented DNN pruning and mobile acceleration framework. We begin by measuring the performance of the algorithm-level method. Then, we demonstrate the accelerations achieved by the overall framework on mobile platforms.

5.1 Experiment Settings

In order to evaluate whether the proposed algorithm-level method can consistently attain efficient compressed models for tasks with different complexities, we test on three representative network structures, i.e., VGG-16, ResNet-18, and ResNet-50, with two major image classification datasets, i.e., CIFAR-10 and ImageNet. Here, CIFAR-10 and ImageNet are viewed as users' private training datasets and are not revealed to the DNN model compression entity. All these evaluations are carried out on one NVIDIA GTX 1080Ti GPU and three NVIDIA RTX 6000 GPUs. Then we experimentally analyse the execution performance of our compiler-assisted framework.

During pruning, we use the following parameter settings for the DNN model compression entity. We initialize the penalty value $\rho = 1 \times 10^{-4}$, and increase ρ by 10 times for every 11 epochs, until ρ reaches 1×10^{-1}. SGD optimizer is utilized for the optimization steps with a learning rate of 1×10^{-3}. An epoch corresponds to 10 iterations, and each iteration process a batch of data. The batch size T is set to 32. Each input image is generated by setting the value of each pixel with a discrete uniform distribution in the range of 0 to 255.

Table 1: Comparison results on CIFAR-10 dataset

	Methods	Base Accuracy	Prune Accuracy	Conv Comp.Rate	Sparsity (Pattern) Type
ResNet-18	DCP[12]	88.9%	87.6%	2.0×	Structured
	AMC[11]	90.5%	90.2%	2.0×	Structured
	Variational Pruning[13]	92.0%	91.7%	1.6×	Structured
	Privacy-Preserving	94.1%	94.9%	8×	Pattern
	Privacy-Preserving	94.1%	94.5%	12×	Pattern
	Privacy-Preserving	94.1%	94.2%	16×	Pattern
ResNet-50	One Shot Pruning[6]	93.8%	93.6%	2.5×	Irregular
	AMC[11]	93.5%	93.5%	1.7×	Structured
	Privacy-Preserving	94.2%	95.0%	8×	Pattern
	Privacy-Preserving	94.2%	94.7%	12×	Pattern
	Privacy-Preserving	94.2%	94.4%	16×	Pattern
VGG-16	Iterative Pruning[4][6]	92.5%	92.2%	2.0×	Irregular
	One Shot Pruning[6]	92.5%	92.4%	2.5×	Irregular
	2PFPCE[28]	92.9%	92.8%	4.0×	Structured
	Efficient ConvNet [10]	93.2%	93.4%	2.7×	Structured
	Privacy-Preserving	93.5%	93.1%	8×	Pattern
	Privacy-Preserving	93.5%	92.4%	12×	Pattern
	Privacy-Preserving	93.5%	91.6%	16×	Pattern

To demonstrate the acceleration of pattern-based sparsity provided by our framework on mobile devices, we compare the proposed framework with three state-of-the-art DNN inference acceleration frameworks, i.e., TFLite [25], TVM [26], and MNN [27]. Our experiments are conducted on a Samsung Galaxy S10 cell phone with the latest Qualcomm Snapdragon 855 mobile platform that consists of a Qualcomm Kryo 485 Octacore CPU and a Qualcomm Adreno 640 GPU.

5.2 Accuracy and Compression Rate Evaluations

We first experiment on CIFAR-10 dataset with the VGG-16, ResNet-18, and ResNet-50 networks. As shown in Table 1, our method not only preserves data privacy, but also reaches a very high compression rate, which is comparable or even better than other pruning algorithms that use the original training dataset during pruning. For ResNet-18, we achieve a 16× compression rate and 94.2% accuracy after pruning. And for ResNet-50, we achieve a 16× compression rate and 94.4% accuracy after pruning. Our method also works well on VGG-16, with a 16× compression rate and 91.6% accuracy.

With promising results on CIFAR-10, we also investigate the performance of our method on ImageNet using ResNet-18. We achieve a 4× compression rate with almost no top-5 accuracy degradation on ResNet-18, which is much better than Network Slimming and DCP. We could further reach a 6× compression rate with 88.0% top-5 accuracy. Moreover, it only takes 14 hours for the proposed method to finish the pruning process using one RTX 6000 GPU.

Furthermore, we notice that our proposed method can even achieve accuracy improvement compared to the pre-trained model

Table 2: Comparison results on ImageNet dataset

	Methods	Base Top-1/5 Accuracy	Prune Top-1/5 Accuracy	Conv Comp.Rate	Sparsity (Pattern) Type
ResNet-18	Network Slimming[15]	68.9/88.7%	67.2/87.4%	1.4×	Structured
	DCP[12]	69.6/88.9%	69.2/88.8%	3.3×	Structured
	Privacy-Preserving	69.9/89.1%	69.3/89.0%	4×	Pattern
	Privacy-Preserving	69.9/89.1%	68.0/88.0%	6×	Pattern

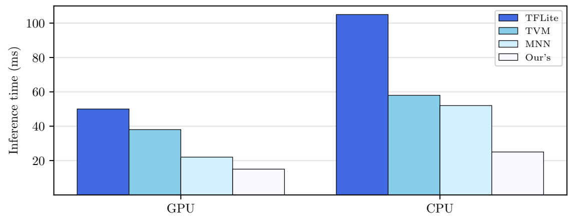

Figure 4: Compiler-based acceleration on 6× compressed ResNet-18 with ImageNet.

on various network structures. The accuracy improvement is attributed to the enhanced image processing ability with the leverage of patterns. Note that performing pattern pruning solely can reserve a 2.25× compression rate due to the intra-kernel sparsity.

5.3 Performance Evaluation on Mobile Platform

In this part, we demonstrate our evaluation results on a mobile device to show the real-time inference of our proposed pattern-based sparse model with the help of the compiler-based acceleration framework. To guarantee fairness, the same pattern-based sparse model are used for all frameworks, and the fully optimized configurations of TFLite, TVM and MNN are enabled.

Fig. 4 shows the mobile CPU/GPU execution time of the pattern-based model on different platforms. The testing model is ResNet-18 with a 6× compression rate on ImageNet dataset. We can observe that our approach achieves significant acceleration on mobile devices compared with other frameworks. On CPU, the proposed framework achieves 4.2× speedup over TFLite, 2.3× speedup over TVM, and 2.0× speedup over MNN. On GPU, our framework achieves 3.3× speedup over TFLite, 2.5× speedup over TVM and 1.4× speedup over MNN. This is because previous frameworks such as TFLite, TVM, and MNN do not have specific optimizations for compressed models as leveraged in our framework. Therefore, there is no obvious acceleration in real implementations on mobiles with such frameworks even though the models have been highly compressed. Real-time execution typically requires 30 frames/sec, i.e., 33ms/frame. From our results, all of our DNN models meet or far exceed this requirement. Furthermore, some of them can even accomplish real-time inference on a mobile CPU.

6 CONCLUSION

In this paper, we propose a privacy-preserving-oriented DNN pruning and mobile acceleration framework. At the algorithm level, we formulate the problem of pattern-based pruning without the usage of the original training dataset as an optimization problem and solve it successfully with an extension of the powerful ADMM. At the compiler level, we adopt corresponding pattern-enabled optimizations. Extensive experiments demonstrate that the proposed framework can achieve real-time inference and maintain accuracy on representative large-scale CNNs while preserving data privacy.

7 ACKNOWLEDGEMENT

This work is partly supported by the National Science Foundation (CCF-1901378, CCF-1919117, CCF-1937500, and CNS-1909172).

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.
[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.
[3] K. He, X. Zhang, S. Ren *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016.
[4] S. Han, J. Pool, J. Tran *et al.*, "Learning both weights and connections for efficient neural network," in *NeurIPS*, 2015.
[5] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *NeurIPS*, 2017.
[6] Z. Liu, M. Sun, T. Zhou *et al.*, "Rethinking the value of network pruning," in *ICLR*, 2018.
[7] T. Zhang, S. Ye, Y. Zhang *et al.*, "A systematic weight pruning of dnns using alternating direction method of multipliers," in *ECCV*, 2018.
[8] A. Ren, T. Zhang, S. Ye *et al.*, "Admm-nn: an algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *ASPLOS*, 2019.
[9] W. Wen, C. Wu, Y. Wang *et al.*, "Learning structured sparsity in deep neural networks," in *NeurIPS*, 2016.
[10] H. Li, A. Kadav, I. Durdanovic *et al.*, "Pruning filters for efficient convnets," *arXiv:1608.08710*, 2016.
[11] Y. He, J. Lin, Z. Liu *et al.*, "Amc: Automl for model compression and acceleration on mobile devices," in *ECCV*, 2018.
[12] Z. Zhuang, M. Tan, B. Zhuang *et al.*, "Discrimination-aware channel pruning for deep neural networks," in *NeurIPS*, 2018.
[13] C. Zhao, B. Ni, J. Zhang *et al.*, "Variational convolutional neural network pruning," in *CVPR*, 2019.
[14] X. Zhu, W. Zhou, and H. Li, "Improving deep neural network sparsity through decorrelation regularization," in *IJCAI*, 2018.
[15] Z. Liu, J. Li, Z. Shen *et al.*, "Learning efficient convolutional networks through network slimming," in *ICCV*, 2017.
[16] M. Yang, M. Faraj, A. Hussein *et al.*, "Efficient hardware realization of convolutional neural networks using intra-kernel regular pruning," in *ISMVL*, 2018.
[17] X. Ma, F.-M. Guo, W. Niu *et al.*, "Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices," *arXiv:1909.05073*, 2019.
[18] A. Jochems, T. M. Deist, I. El Naqa, M. Kessler, C. Mayo, J. Reeves, S. Jolly, M. Matuszak, R. Ten Haken, J. van Soest *et al.*, "Developing and validating a survival prediction model for nsclc patients through distributed learning across 3 countries," *International Journal of Radiation Oncology* Biology* Physics*, vol. 99, no. 2, pp. 344–352, 2017.
[19] A. Jochems, T. M. Deist, J. Van Soest, M. Eble, P. Bulens, P. Coucke, W. Dries, P. Lambin, and A. Dekker, "Distributed learning: developing a predictive model based on data from multiple hospitals without data leaving the hospital–a real life proof of concept," *Radiotherapy and Oncology*, vol. 121, no. 3, pp. 459–467, 2016.
[20] S. Boyd, N. Parikh, E. Chu *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.
[21] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," in *BMVC*, 2015.
[22] W. F. Tinney and J. W. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization," *Proc of the IEEE*, vol. 55, no. 11, pp. 1801–1809, 1967.
[23] S. Muchnick, *Advanced Compiler Design and Implementation.* Morgan Kaufmann Publishers Inc., 1997.
[24] Coleman, Stephanie and McKinley, Kathryn S., "Tile Size Selection Using Cache Organization and Data Layout," in *PLDI*, 1995.
[25] https://www.tensorflow.org/lite/performance/model_optimization.
[26] T. Chen, T. Moreau, Z. Jiang *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," in *OSDI*, 2018.
[27] https://github.com/alibaba/MNN.
[28] C. Min, A. Wang, Y. Chen *et al.*, "2pfpce: Two-phase filter pruning based on conditional entropy," *arXiv:1809.02220*, 2018.