2022 23rd International Symposium on Quality Electronic Design (ISQED) | 978-1-6654-9466-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISQED54688.2022.9806237

BLCR: Towards Real-time DNN Execution with Block-based Reweighted Pruning

Xiaolong Ma⋆[1], Geng Yuan⋆[1], Zhengang Li⋆[1], Yifan Gong[1], Tianyun Zhang[2], Wei Niu[3], Zheng Zhan[1], Pu Zhao[1], Ning Liu[4], Jian Tang[4], Xue Lin[1], Bin Ren[3], Yanzhi Wang[1]
[1]*Northeastern University,* [2]*Cleveland State University,*
[3]*College of William and Mary,* [4]*Midea Group,*
[1]{ma.xiaol, yanz.wang}@northeastern.edu,
⋆*These authors contributed equally.*

***Abstract*—Accelerating DNN execution on resource-limited computing platforms has been a long-standing problem. Prior works utilize ℓ_1-based group lasso or dynamic regularization such as ADMM to perform structured pruning on DNN models to leverage the parallel computing architectures. However, both of the pruning schemes and pruning methods lack universality, which leads to degraded performance and limited applicability. Considering mobile devices are becoming an important carrier for deep learning tasks, current approaches are not ideal for fully exploiting mobile parallelism while achieving high inference accuracy. To solve the problem, we propose BLCR, a novel block-based pruning framework that comprises a general and flexible structured pruning scheme that enjoys higher flexibility while exploiting full on-device parallelism, as well as a powerful and efficient reweighted regularization method to achieve the proposed sparsity scheme. Our framework is universal, which can be applied to both CNNs and RNNs, implying complete support for the two major kinds of computation-intensive layers (i.e., CONV and FC layers). To complete all aspects of the pruning-for-acceleration task, we also integrate compiler-based code optimization into our framework that can perform DNN inference on mobile devices in real-time. To the best of our knowledge, it is the first time that the weight pruning framework achieves universal coverage for both CNNs and RNNs with real-time mobile acceleration and no accuracy compromise.**

I. Introduction

Deep Neural Networks (DNNs) such as Convolutional Neural Networks (CNNs) [1]–[3] and Recurrent Neural Networks (RNNs) [4] [5] have been extensively adopted in various artificial intelligence (AI) systems. However, accelerating the computational intensive DNN inference is very challenging for many AI applications, especially those with critical time constraints, such as self-driving cars [6] and real-time translation [7].

Weight Pruning [8]–[12] has gained its popularity due to the effectiveness in reducing model size and computation cost. In order to remove redundant weights while maintaining accuracy, many studies have been proposed regarding both pruning schemes (DNN structure level) and pruning method (algorithm level). According to the structure of pruned models, there are mainly two DNN pruning approaches: *non-structured pruning* and *structured pruning*. However, non-structured pruning [9] [13] has been proven by many recent studies [8] [14] that it is not compatible with the parallelism in hardware accelerations due to the imbalanced computation and significant overhead. Structured pruning [8], [10]–[12] has been proposed to conquer the challenge. A structured pruned model maintains the regularity of the weight matrix, which eliminates the overhead and facilitates on-device acceleration. However, the aggressive pruning strategy causes severe information loss, making accuracy degradation non-negligible. Achieving both high accuracy and fast inference with DNN pruning is an ideal but very challenging goal.

Efforts have been made to achieve this goal. At algorithm level, many pruning techniques have been proposed to find the uncritical weights. For non-structured pruning, prior works leverage a magnitude-based pruning method that prunes weights with small magnitudes or use ℓ_1 regularization to explore sparsity in DNN models. For structured pruning, the static ℓ_1-based group lasso regularization is used to find the regular sparse pattern in DNN models. However, the above approaches fail to find a satisfactory solution for the pruning problem due to the poor solution quality for the non-convex ℓ_0 problem. With a significant improvement in the solution quality, ADMM [15] pruning supersedes (almost) every pruning framework and becomes the state-of-the-art method. Nevertheless, ADMM still suffers from sub-optimal solution quality and long convergence time, especially for the long-standing problem of finding structured sparsity solution for the Fully Connected (FC) layer. This will certainly limit the usage of ADMM solutions on many CNNs and almost all RNNs since they are majorly composed of FC layers.

In this paper, we present a unified pruning framework – block-based structured pruning with reweighted regularization (BLCR), and the design of the associated compiler-aided acceleration, for off-the-shelf mobile devices. We focus on two aspects: pruning scheme and pruning method.

Aspect 1: From the pruning scheme aspect, we propose block-based structured pruning which divides DNN layers into multiple blocks and applies structured pruning independently to each block. Our design takes a unique perspective on structured pruning, which greatly enlarges the design space by introducing a higher degree of flexibility with a changeable block shape. More importantly, the proposed pruning scheme is applicable to both CNNs and RNNs without obvious accuracy degradation, and outperforms the existing pruning schemes. It achieves similar or even higher accuracy compared with non-structured pruning, and preserves the hardware compatibility advantage of structured pruning, with the compiler-based code optimization embedded in our pruning-acceleration framework.

Aspect 2: From the pruning method aspect, we propose to use reweighted group lasso regularization method to generate structured sparsity (we will use *reweighted method* in the rest of the paper for concision). By introducing a reweighted term into regularization, our method can perform group regularization at a more precise location in DNN with an appropriate degree. Compared to the traditional ℓ_1-based group lasso and the recently developed ADMM regularization method, the reweighted method exhibits a significant improvement in the regularization effect (i.e., facilitating better pruning results) with a desirable short convergence time (i.e., efficient training process), which makes it a favorable approach that naturally fit for the DNN pruning problems.

We show the performance improvements of BLCR framework in three ways. **I.** the proposed reweighted method can efficiently find uncritical weights. Compared to other methods, the reweighted method achieves better weight regularization effect using significantly shorter training time. **II.** the proposed block-based pruning scheme is more general and achieves extremely high compression rates in both CNN and RNN. **III.** the proposed BLCR pruning naturally fits for the compiler optimization. Our designed compiler-aided acceleration framework achieves real-time inference on the resource-limited mobile devices.

II. Background and Motivation

A. Structured Pruning Scheme

Recent works [8], [10], [12], [14], [16], [17] considered to incorporate regularity (i.e., filter-wise, channel-wise, etc.) in weight pruning, which generates regular and smaller weight matrices for faster executions on CPUs/GPUs. For convolution computations, weight matrices are usually transformed into general matrix multiplication (GEMM) form. As a result, filter pruning can also be termed as row pruning since it corresponds to removing one row of the weight matrix, and channel pruning corresponds to reducing multiple consecutive columns (column pruning). Current structured pruning approaches suffer from notable accuracy loss when the compression rate is high because the entire information of the pruned filter(s)/channel(s) is lost. As a result, it usually has limited compression rates and low accuracy, as well as limited applicability as most works focus on CONV layers only. For FC layers (applied partially in CNN and majorly in RNN), structured pruning is applicable but not desirable due to the same reason above. The drawback is obvious, especially for time-based RNNs since one pruned row/column in an RNN will not be utilized for all timestamps, causing server accuracy degradation.

B. Regularization-based Pruning Methods

Finding structured sparsity in a DNN model is intrinsically solving an ℓ_0 optimization problem with structured constraints. The following two mainstream methods have been proposed to solve this problem:

Static regularization is firstly utilized in solving non-structured pruning problems by incorporating ℓ_1 regularization into DNN training. By extending ℓ_1 regularization into group lasso [8], [14], [18] form, structured pruning on DNN models can also be achieved. With specified regularization dimensions (groups), it can perform different types of structured pruning (i.e., filter pruning, channel pruning and the combination of them). However, this method yields limited compression rates and non-negligible accuracy degradation due to the intrinsically heuristic and non-optimized approach.

Dynamic regularization method such as ADMM pruning [19], [20] usually reforms pruning problems into optimization problems with dynamically updated regularization terms bounded by the designated constraint (i.e., pruning with specific dimensions or with any desired weight matrix shapes) sets. During training, ADMM can separately and iteratively solve the pruning problem. Although this method is revolutionary in its functionality and outperforms the former ones in terms of pruning rate/accuracy, a satisfactory solution cannot always be guaranteed for the non-convex (i.e., DNN loss function) problem, not to mention that this method suffers from a time-consuming training process. Another approach prunes DNN with dynamic regularization via Generative Adversarial Learning (GAL) technique [16]. Specifically, GAL uses adversarial regularization to learn a soft mask to find the sparse structure, and FISTA [21] [22] is introduced to solve the adversarial regularization problem via two alternating steps. However, GAL suffers from convergence quality since the FISTA framework is not focused on the non-convexity of the loss function, thus the method is still heuristic and can not guarantee solution quality.

To fully utilize the regularization-based method to find sparsity, neural architectures search (NAS) is also becoming popular. Prevailing NAS methods [23]–[25] optimize the network topology, which greatly improves the performance of neural networks. Regularized by minimization of the computation cost, NAS aims to search for the best topology, or search for the best size of a network directly [10]. However, the search space of these methods is extremely large, which causes significant computational overhead to search and select the best model from hundreds of models.

C. Motivation

From the pruning scheme aspect, the current structured pruning schemes suffer from major information loss. The accuracy drop is especially significant in RNN pruning. The motivation of our study is to seek an approach to maintain the regularity in the pruned model (for facilitating hardware acceleration), while restoring the flexibility of the spatial distribution of the weights (to re-gain high accuracy). In our proposed block-based pruning which is applicable to both

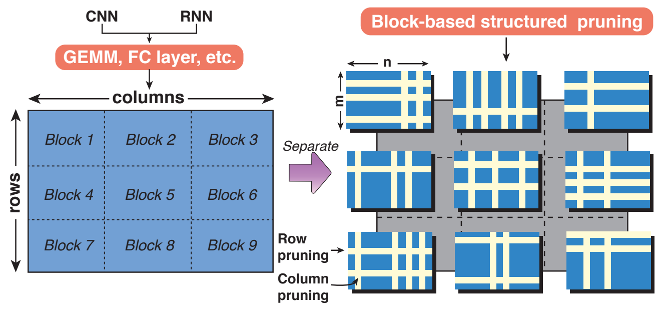

Fig. 1: Proposed flexible, block-based structured pruning.

CNNs and RNNs, we take a unique step towards this goal by introducing a new pruning perspective, and avoid the pitfall of making this approach "a mere trade-off" between model accuracy and regularity. We also take a further step of compiler optimization to establish the connection between the general, block-based sparsity and the on-device speedups.

From the regularization aspect, we emphasize that both current static and dynamic regularization methods are limited by their intrinsic shortcomings. For static regularization, the ℓ_1 or group lasso regularization penalizes all weights in its dimension scope through the entire network, which means some important weights are penalized to near-zero values, thereby resulting in highly impaired solutions. On the other hand, the dynamic regularization method reforms pruning problem as an optimization problem with hard constraints on ℓ_0 norm, and then use ADMM to solve it. However, this method suffers from long convergence time due to the strong non-convexity of ℓ_0 norm, especially with structured hard constraints. ADMM involves a large amount of hyper-parameters that need to be tuned manually for each layer, which is very inefficient. It is imperative to find an effective method to solve the ℓ_0 optimization problem with self-adaptive regularization and soft constraints.

III. Unified and Flexible Framework of DNN Pruning - Acceleration

In this section, we propose a unified framework of DNN weight pruning, supporting (i) the flexible, *block-based structured pruning* that applies to both CNN and RNN architectures, and (ii) highly effective weight pruning algorithm with *reweighted method*. Our framework also includes a general method to accelerate DNN execution by utilizing compiler-based code optimization, achieving holistic supports for the DNN pruning-acceleration studies.

A. Block-based Structured Pruning – A Unique Perspective on Structured Weight Pruning

Conventional, structured pruning treats the DNN weight matrix in each layer as a whole, and selects to prune a whole row or column of the entire weight matrix. However, the accuracy performance is hindered by this limited, inflexible view of structured pruning.

In our perspective, we consider the weight matrix in each layer (e.g., GEMM or FC that represent different types of layer-wise computation) to be composed of multiple weight blocks with the same size $m \times n$ as Figure 1 shows. We apply independent row and column pruning on each block, with potentially different pruning rates (number of pruned rows/columns) in each block, to ensure high flexibility. The remaining weights in each block still form a full matrix with a smaller size. Within our perspectives, the aforementioned non-structured pruning and the state-of-the-art structured pruning are two extremes in our design with the block size 1×1 (i.e., non-structured pruning) and the size of the whole matrix (i.e., structured pruning).

B. Effective Regularization-based Pruning Algorithm with Reweighted Method

For an N-layer DNN of interest, let $\mathbf{W}$ denote the collection of weights for i-th layer, i.e., $\mathbf{W} = \{\mathbf{W}_i\}_{i=1}^{N}$. According to our design of the flexible, block-based sparsity, we propose the following ℓ_0 constraints on the pruning of $\mathbf{W}_i$.

`Constraints`: Each $\mathbf{W}_i$ will be uniformly divided into K blocks with the size of $m \times n$ in each of the GEMM or FC matrix, namely, $\mathbf{W}_i = [\mathbf{W}_{i1}, \mathbf{W}_{i2}, ..., \mathbf{W}_{iK}]$, where $\mathbf{W}_{ij} \in \mathbb{R}^{m\times n}$. Let $[\mathbf{W}_{ij}]_{p,:}$ and $[\mathbf{W}_{ij}]_{:,q}$ denote the p-th row and the q-th column of $\mathbf{W}_{ij}$, respectively.

Towards training of the DNN, we minimize the loss function of the network to increase accuracy. In order to achieve structured sparsity, the common method is to add group lasso regularization [18] to the loss function. In fact, achieving block-based row and column sparsity is also a special group lasso problem. Let $f(\mathbf{W})$ denote the training loss. The classic optimization with group lasso regularization on the block-based sparsity can be formulated as

$$\underset{\mathbf{W}}{\text{minimize}}\ f(\mathbf{W}) + \lambda \sum_{i=1}^{N}\sum_{j=1}^{K} \|[\mathbf{W}_{ij}]\|_g \tag{1}$$

where λ is the penalty parameter to adjust the relative importance of accuracy and sparsity degree, and $\|\cdot\|_g$ denotes group lasso computation. It is difficult to find high quality solution using this fixed regularization method (please refer to the explanation in Section II-C). Instead, an effective dynamic regularization method dealing with such soft constraints is in need. To achieve this goal, we propose to use reweighted method [26] to solve group lasso regularization, thereby eliminating the previous shortcoming of applying the same penalty on important and less significant weights. We formulate the following two optimization problems for block-based row pruning and column pruning.

For block-based row pruning, we solve

$$\underset{\mathbf{W}}{\text{minimize}}\ f(\mathbf{W}) + \lambda \sum_{i=1}^{N}\sum_{j=1}^{K} \left(\mathcal{P}_i^{(t)} \circ \|[\mathbf{W}_{ij}]_{p,:}\|_2\right) \tag{2}$$

where $\circ$ denotes element-wise multiplication, $\|\cdot\|_2$ denotes the Frobenius norm and $\mathcal{P}_i^{(t)}$ is the collection of penalty weights[1],

[1]$\mathcal{P}$ is initialized by the original weights in the pre-trained model.

Algorithm 1: Reweighted regularization for block-based structured pruning

```
1 Initialization: Pretrained DNN model with initialized
    $\mathcal{P}_i$; Set $t \leftarrow 1$ and total iteration number $T$;
    Pre-define block size $m$ and $n$
  Result: Block-based structured pruned model;
2 Each layer $K \leftarrow$ (the size of $\mathbf{W}_i$)/($m \times n$);
3 while $t \leq T$ do
4   | Solve (2) and/or (3) using standard SGD solver ;
5   | Update $\mathcal{P}_i^{(t+1)}$ using the solution of $\mathbf{W}_i^{(t)}$
6 end
7 Remove the group of weights close to zero and
    fine-tune the rest of non-zero weights.
```

which is updated in every iteration t to help increase the degree of sparsity beyond group lasso regularization. In each iteration, the solution of $\mathbf{W}_i$ is given by $\mathbf{W}_i^{(t)}$ and we update $\mathcal{P}_i$ by setting

$$\mathcal{P}_i^{(t+1)} = \frac{1}{\|[(\mathbf{W}_{ij})]^t_{p,:}\|_2^2 + \epsilon}$$

where ϵ is a parameter with small value to prevent the division by zero denominator.

For block-based column pruning, we solve

$$\underset{\mathbf{W}}{\text{minimize}}\, f(\mathbf{W}) + \lambda \sum_{i=1}^{N}\sum_{j=1}^{K}\left(\mathcal{P}_i^{(t)} \circ \|[\mathbf{W}_{ij}]_{:,q}\|_2\right) \quad (3)$$

and update $\mathcal{P}_i$ by

$$\mathcal{P}_i^{(t+1)} = \frac{1}{\|[(\mathbf{W}_{ij})]^t_{:,q}\|_2^2 + \epsilon}$$

Please note that (2) and (3) can be solved separately or simultaneously using the standard solver.

Algorithm 1 describes the general steps that are used in the proposed reweighte method. We first initialize $\mathcal{P}_i$ using the pretrained model, and pre-define the block size for the pruned model. During DNN training, we incorporate the reweighted group lasso regularization in (2) and (3), and update the penalty parameter $\mathcal{P}_i$ iteratively. By updating the penalty, we "*reweight*" the regularization term(s) that is (are) bounded in the optimization problems. After reweighted steps, we remove the weights (or group of weights) which are close to zeros and fine-tune the DNN using the non-zero weights.

Reweighted regularization analysis: Consider that two weights w_i and w_j ($w_i < w_j$) are penalized by certain regularization. The larger w_j is inevitably being penalized more heavily than the smaller w_i. Although it is easier for w_i to become zero, the fact that w_j is penalized still violates the original intention of weight pruning, which is to remove the "uncritical" weights. Larger weights typically serve a critical role in generating stronger activation for a more confident decision. In the reweighted method, w_j remains un-penalized or even being rewarded while w_i's penalty is amplified. Interestingly, our experimental results in Section IV-A show that the importance of a (group of) weight is also related to its location, and the reweighted method can effectively separate those locations. We claim that this characteristic is attributed to the globally and iteratively updating the potential sparse pattern by reweighted method, such that the algorithm converges to a better sparse solution, while other methods (static regularization or ADMM, etc.) only target at a fixed sparse pattern.

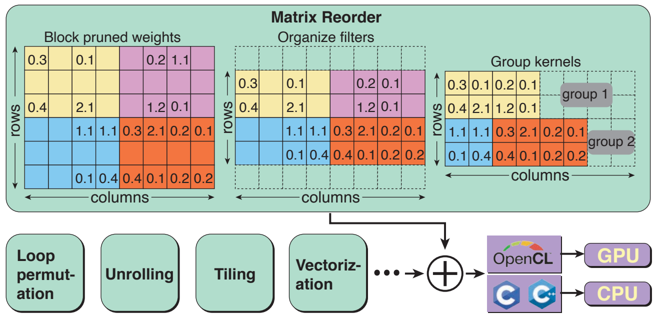

Fig. 2: Matrix reorder and optimizations for efficient execution code generation.

Reweighted training: Compared with ADMM training which also uses an iteratively updating scheme for the regularization term, reweighted method uses fewer training epochs for the loss to converge. For example, when pruning VGG-16 on CIFAR-10, the ADMM method usually requires 1,000 - 1,200 epochs to converge when the compression rate is around 20×. Additionally, the retraining step also requires the same amount of epochs to restore accuracy. In reweighted training, we only need 150 - 200 epochs for reweighted step and 200 epochs for retraining. In the meantime, ADMM requires setting pruning ratio and other hyper-parameters (e.g., layer-wise penalty) for each layer manually, while reweighted method only requires one penalty parameter for all layers. Also, the soft constraints in reweighted method determine pruning ratio for the whole network automatically, which eliminates a lot of parameters that need to be set empirically.

Multiple objective functions: The original objective function in the proposed reweighted method is targeting at DNN weight reduction. However, our objective function can also be formulated for operation (FLOPS) reduction, storage reduction, etc., and solved using the same reweighted method. We will not discuss those formulations since they are not the focuses of this paper.

C. Compiler-aided Mobile Acceleration Framework for Block-based DNN Sparsity

To fully leverage the block-based sparsity, we design a compiler-aided acceleration framework to deploy DNN models on the computing platform. We adopt code generation to convert a DNN model into computational graph embodied by static C++ (for CPU execution) or OpenCL (for GPU execution) code, and with the optimization techniques to guarantee end-to-end execution efficiency. We use mobile devices as the

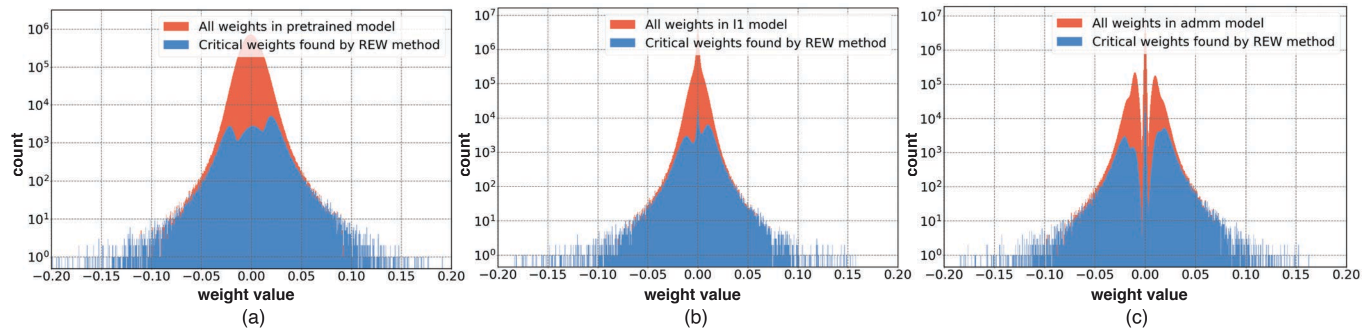

Fig. 3: Critical weights distribution (logarithmic scale) found by reweighted method in the first FC layer of a VGG-16 model. The comparison includes (a) a pretrained model, (b) an ℓ_1-based group lasso regularized model and (c) an ADMM regularized model.

computing platform. However, the concept and principle of using compiler to execute DNN is universal and can be utilized in (almost) every computing device.

The compiler optimization aims to address the following performance challenges in pruned DNN executions: *thread divergence* and *load imbalance* caused by the well-known challenges of the sparse matrix multiplications. The core of our solution is the *matrix reorder* technique.

Matrix reorder: At first glance, the block-based sparsity has a disordered weight distribution, which incurs significant thread divergence and load imbalance if rows are processed by different threads. Figure 2 illustrates our proposed matrix reorder technique. As the remaining weights that appear in certain rows and columns in each block have a certain degree of regularity, we first reorder the rows (e.g., filters in CNN) by arranging the ones with the same or similar patterns together. Next, we compact the weights in the column direction (e.g., kernels in CNN). At last, the rows with the same or similar computations are grouped together. As a result, each group is processed by all threads in parallel, and each thread is in charge of multiple consecutive rows. Thus, the computation divergence among these threads is significantly reduced. On the other hand, since the weight distribution pattern in each block is regular and *known* after grouping, the input matrix that corresponds to each weight group will be loaded only once. The load imbalance can be relieved thanks to the register level loading operation reduction. Based on matrix reordering result, other standard optimizations (e.g., loop permutation, unrolling, tiling, vectorization, etc.) effects can be maximized, and our compiler-aided framework generates more efficient codes comparing to other DNN acceleration frameworks.

IV. Experimental Results

Methodology: In our experiment, the proposed BLCR pruning framework is utilized on two different machine learning tasks – image classification and natural language processing (NLP). In image classification tasks, our experiments are based on four widely used CNNs, VGG-16 [2], ResNet-18/20/50/56 [1] and MobileNet-V2 (MBNT) [27] on CIFAR-10 and ImageNet [28] datasets; and for NLP task, we test our proposed pruning framework for GRU [5] on TIMIT dataset. We train the networks on an eight NVIDIA Titan RTX GPUs server using PyTorch [29]. We train CIFAR-10 for 160 epochs and train ImageNet for 100 epochs with standard data preprocessing.

In order to show the acceleration of block-based sparsity on mobile devices, we compare it with three state-of-the-art DNN acceleration frameworks, TensorFlow-lite (TF-Lite) [30], TVM [31], and MNN [32]. Our evaluations are conducted on a Samsung Galaxy S10 phone with the latest Qualcomm Snapdragon 855 that consists of a Qualcomm Kryo 485 Octa-core CPU and a Qualcomm Adreno 640 GPU.

A. Critical Weights Analysis on Different Regularization Methods

We state that the proposed reweighted method can achieve better pruning result. The reason is that our method can effectively separate the uncritical weights from critical ones. We use VGG-16 on ImageNet to generate a sparse model based on the proposed reweighted regularization method, and compare it with ℓ_1-based regularization as well as ADMM regularization. To ensure absolute fairness, all the models in the comparison use the same pruning scheme and compression rate. In this case, we use one block (i.e., prune entire columns and rows) in each layer for all methods.

Figure 3 illustrates the difference of critical weights distribution between reweighted method and others. We first find the non-zero value *positions* in the sparse model generated by our reweighted method. Through using those *positions*, we find the corresponding weights and their distribution in (i) a pretrained model, (ii) an ℓ_1-based group lasso regularized model and (iii) an ADMM regularized model. The critical weight distribution is shown in Figure 3, with the orange color denoting original weights distribution and the blue color indicating the "critical" weights found and preserved by our method. According to the figure, we have the following analyses:
(a). In a pretrained DNN model, some weights with small magnitude are critical to maintain accuracy. Therefore, some pruning works that only prune small weights are very subjective and hard to achieve good results.

	Method	Dense Acc. (%)	Prune Acc. (%)	Comp. Rate	Sparsity Scheme
ResNet-20	AMC	90.5	90.2	2.0×	Chl (Lasso)
	SFP	92.2	90.8	1.7×	Chl (Heuristic)
	TAS	92.8	92.8	1.8×	Chl (NAS)
	FPGM	92.2	91.9	2.5×	Chl (Lasso)
	BLCR	**92.5**	**92.4**	**2.4**×	1-BLK (REW)
	BLCR	**92.5**	**92.3**	**6.0**×	BLK (REW)
	BLCR	**92.5**	**91.1**	**11.6**×	BLK (REW)
ResNet-56	TAS	94.5	93.7	2.0×	Chl (NAS)
	SFP	93.6	93.4	1.7×	Chl (Heuristic)
	GAL	93.3	90.4	2.9×	Chl (Adv)
	FPGM	93.6	93.5	2.5×	Chl (Lasso)
	BLCR	**94.3**	**94.0**	**2.5**×	1-BLK (REW)
	BLCR	**94.3**	**93.6**	**5.5**×	BLK (REW)
	BLCR	**94.3**	**92.3**	**11.0**×	BLK (REW)
MBNT	DCP	94.5	94.7	1.4×	Chl (Heuristic)
	BLCR	**94.5**	**94.5**	**7.1**×	1-BLK (REW)
	BLCR	**94.5**	**94.5**	**8.9**×	BLK (REW)
	BLCR	**94.5**	**93.4**	**10.3**×	BLK (REW)
VGG-16	2PFPCE	92.9	92.8	4.0×	Row (Lasso)
	ADMM	93.7	92.7	50.0×	R+C (ADMM)
	Decor	93.5	93.3	8.5×	Chl (Lasso)
	GAL	93.9	90.8	5.6×	Chl (Adv)
	BLCR	**93.5**	**93.0**	**50.0**×	1-BLK (REW)
	BLCR	**93.5**	**93.5**	**50.1**×	BLK (REW)
	BLCR	**93.5**	**93.0**	**69.7**×	BLK (REW)

TABLE I: BLCR pruning results on CIFAR-10 using VGG-16, ResNet-20/56 and MobileNet-V2 (MBNT). Comparison baselines: AMC [33], Decor [34], 2PFPCE [35], SFP [36], TAS [10], FPGM [12], DCP [11], GAL [16], ADMM [37].

(b). In an ℓ_1-based group lasso or ADMM regularized model, part of the weighs are penalized to zero or near-zero values, and then those close-to-zero values are pruned and the rest non-zero values are fine-tuned. However, the reweighted method considers some weights that have been penalized are critical, thus should not be pruned.

We conclude that reweighted method separates critical weights in a very different way, in which the importance of weight(s) is not only based on its value, but also associated with its position. To prove and reinforce our conclusion, we need to show a strong accuracy improvement of the reweighted method compared with others, which is reported in the following section.

B. Accuracy Analysis on Overall Pruning Results

Terminology description: In Table I, Table II and Table III, we use *BLK* to denote block-based pruning scheme, and *REW* to represent reweighted method. Since the conventional pruning scheme is performed on the whole weight matrix, we use *1-BLK* to denote it. Beyond one block structured pruning, we also divide weights into several blocks to show BLCR pruning results. We use $m \times n$ *BLK* in our experiments to denote the proposed block-based pruning scheme. We use *Comp. Rate* as weight reduction criterion which can be obtained by the ratio of the number of all weighs to the number of the non-zero weights. We use *Chl* for channel pruning and *R+C* for row and column pruning.

	Method	Top-1 Acc. (%)	Top-5 Acc. (%)	Comp. Rate	Sparsity Scheme
ResNet-18	DCP	69.6→64.1	88.9→85.7	3.3×	Chl (Heuristic)
	ADMM	*N/A*	89.1→88.4	3.3×	R+C (ADMM)
	SFP	70.3→67.1	89.6→87.8	1.7×	Chl (Heuristic)
	TAS	70.6→69.1	89.8→89.2	1.5×	Chl (NAS)
	FPGM	70.2→68.3	89.6→88.5	3.3×	Chl (Lasso)
	BLCR	**70.1→69.2**	**89.3→88.6**	**4.0**×	1-BLK (REW)
	BLCR	**70.1→69.4**	**89.3→89.0**	**4.0**×	BLK (REW)
	BLCR	**70.1→66.9**	**89.3→87.2**	**7.6**×	BLK (REW)
ResNet-50	TAS	77.5→76.2	93.5→93.1	1.7×	Chl (NAS)
	SFP	76.2→74.6	92.9→92.1	1.7×	Chl (Heuristic)
	FPGM	76.2→75.6	92.8→92.6	3.3×	Chl (Lasso)
	CP	*N/A*	92.2→90.8	2.0×	Chl (Lasso)
	GBN	75.8→75.2	92.7→92.4	2.2×	Chl (Lasso)
	BLCR	**77.5→77.0**	**93.8→93.3**	**3.4**×	1-BLK (REW)
	BLCR	**77.5→76.7**	**93.8→93.1**	**4.5**×	BLK (REW)
	BLCR	**77.5→76.2**	**93.8→92.8**	**5.1**×	BLK (REW)
MBNT	AMC	*N/A*	71.8→70.8	1.4×	Chl (Lasso)
	BLCR	**70.9→70.5**	**90.4→89.8**	**1.6**×	1-BLK (REW)
	BLCR	**70.9→70.0**	**90.4→89.7**	**2.0**×	BLK (REW)
VGG-16	Decor	73.1→73.2	*N/A*	3.9×	Row (Lasso)
	APoZ	68.4→66.2	88.4→87.6	2.0×	Chl (Heuristic)
	BLCR	**74.5→74.0**	**91.7→91.5**	**5.5**×	1-BLK (REW)
	BLCR	**74.5→74.4**	**91.7→91.6**	**3.1**×	BLK (REW)
	BLCR	**74.5→73.8**	**91.7→91.2**	**7.8**×	BLK (REW)

TABLE II: BLCR pruning results on ImageNet using VGG-16 and ResNet-18/50 and MobileNet-V2 (MBNT). The arrow (→) indicates accuracy before and after pruning. Extra comparison baselines: APoZ [38], CP [14], GBN [17].

Method	Dense PER	Prune PER	Comp. Rate	Sparsity Scheme	Speed(ms) (CPU/GPU)
ESE	20.40	20.70	8.0×	Irr. (Heuristic)	*N/A*
C-LSTM	24.15	25.48	16.0×	Block-circ.	*N/A*
E-RNN	20.02	20.20	8.0×	Block-circ.	*N/A*
BLCR	**18.8**	**18.8**	**19.1**×	BLK (REW)	**0.97/0.50**
BLCR	**18.8**	**23.2**	**112.9**×	BLK (REW)	**0.35/0.25**
BLCR	**18.8**	**24.0**	**231.3**×	BLK (REW)	**0.21/0.09**

TABLE III: BLCR pruning and speed results on GRU with TIMIT dataset. PER denotes phone error rate (%).

In our previous analysis, we stress that reweighted regularization can effectively separate critical weights, thus achieving better pruning solutions. In this part, we demonstrate the overall compression results to support our conclusion. To guarantee fairness, we first use the conventional pruning schemes (i.e., prune entire rows and columns) in 1-BLK scheme to show the proposed pruning algorithm outperforms other methods such as Lasso, ADMM, NAS, Adversarial (Adv) and other heuristics. Please note that row pruning is equivalent to channel pruning since each row in GEMM indicates a convolution filter which corresponds to a channel in its adjacent layer, and the consecutively pruned columns are also equivalent to channel pruning.

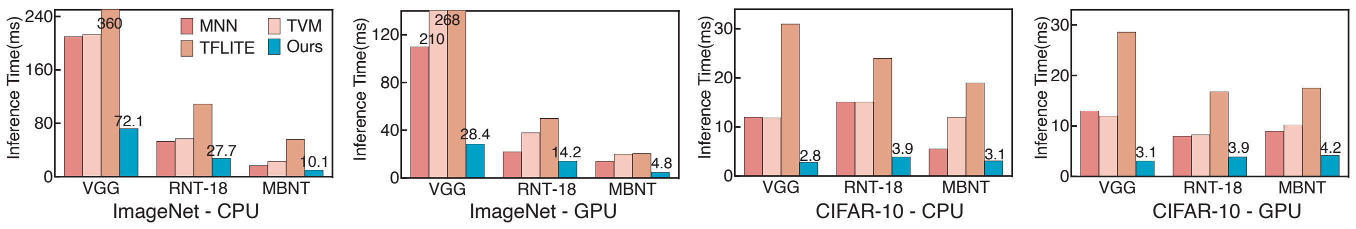

Fig. 4: Mobile CPU/GPU inference time (ms) on different network structures with CIFAR-10 and ImageNet images.

Table I,II show our pruning results using different CNNs with CIFAR-10 and ImageNet datasets with 1-BLK and size 4×16 BLK. Table III shows RNN pruning results using GRU on TIMIT dataset with size 2×32 BLK. A valuable observation in Table I,II is that by using reweighted method, the traditional pruning scheme achieves better results compared to other methods, including the state-of-the-art ADMM pruning. On CIFAR-10 and ImageNet dataset, our pruning results consistently outperform the recent approaches in all networks. For GRU pruning on TIMIT, we compare our method with the SOTAs that are ESE [39], C-LSTM [40] and E-RNN [41]. Particularly, ESE prunes weight in a irregular scheme, and C-LSTM and E-RNN compress weight matrix into block-circulant format and use Fast Fourier Transform (FFT) operation to perform DNN computation. We achieve $19.1\times$ compression for GRU without accuracy loss, and the maximum $231.3\times$ with 24% phone error rate (PER), which still outperforms C-LSTM. To sum up, BLCR pruning achieves better compression results for both CNNs and RNNs, leading to lightweight model size and computation.

C. Performance Evaluation on Mobile Devices

Execution time results are shown in Figure 4. We test the block-based sparse models on mobile CPU/GPU. To ensure fairness, all frameworks are using the same sparse model, and we also enable the fully optimized configurations of TF-Lite, TVM and MNN (e.g., Winograd optimization is turned on). All test models are the ones with the largest compression rates in Table I and Table II. We omit ResNet-50 results because they are similar to VGG-16 results. For GRU RNN execution, since other frameworks do not support end-to-end execution on mobile devices, we only report the execution time of the proposed block-based sparse model with block size 2×32 in Table III. We can see our approach achieves significant acceleration on mobile devices compared with other frameworks. On CPU, BLCR model on compiler-aided framework achieves $1.29 - 4.18\times$ speedup over MNN, $2.12 - 4.19\times$ speedup over TVM, $3.93 - 11.42\times$ speedup over TF-Lite. On GPU, we achieves $1.53 - 4.26\times$ speedup over MNN, $2.38 - 7.39\times$ speedup over TVM, $3.66 - 9.51\times$ speedup over TF-Lite. For image classification tasks, all of our results on mobile GPU exceed the real-time requirements (usually $33ms$/frame). For NLP tasks, the proposed framework also achieves real-time speech recognition.

Discussion on the comparison fairness. Our comparisons on mobile acceleration are fair. Using ImageNet data and VGG-16 as an example, our method achieves $7.8\times$ parameter reduction and $6 - 10\times$ CPU/GPU acceleration compared to the *dense* model (both using our compiler). Under the same accuracy, channel pruning achieves much lower compression rate compared with our pruning scheme. For example, to achieve 91.5% top-5 accuracy for VGG-16 on ImageNet, channel pruning achieves at most $5.5\times$ compression rate while we achieve $7.8\times$ compression rate. On mobile CPU/GPU, this $5.5\times$ compression rate at most translates into $3 - 5\times$ acceleration rate using existing (also the SOTAs) frameworks like TVM or MNN. Furthermore, we have integrated full system-level optimization to our compiler acceleration framework, which makes our compiler outperform other frameworks when executing *dense* models. For example, when DNN models are dense, our compiler achieves $2 - 3\times$ acceleration to TF-Lite, $1.5 - 2\times$ acceleration to TVM and $1.1 - 1.3\times$ acceleration to MNN.

To sum up, we can see that (i) our compiler already runs faster on dense models, compared to the SOTAs, and (ii) our compression + compiler combination is also better in performance. So overall we have a very significant speedup.

V. Conclusion

This paper presents the block-based DNN structured pruning framework using reweighted regularization method (BLCR). The proposed block-based structured sparsity is flexible and can be used in both CNN and RNN applications. With the support of the compiler code generation and optimization, our framework can achieve real-time acceleration on mobile devices. The proposed framework also uses reweighted method to dynamically update the regularization process, which improves the pruning performance significantly within considerably shorter training time. Compared to the state-of-the-art DNN pruning methods and acceleration frameworks, the proposed framework is general and achieves higher performance.

VI. Acknowledgement

This work is partly supported by the National Science Foundation CNS-1909172, CCF-1919117, and CNS-1929300. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

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

[3] 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*, 2018, pp. 4510–4520.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[5] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[6] B. T. Nugraha, S.-F. Su *et al.*, "Towards self-driving car using convolutional neural network and road lane detector," in *ICACOMIT*, 2017.

[7] J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin, "A convolutional encoder model for neural machine translation," *arXiv preprint arXiv:1611.02344*, 2016.

[8] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *NeurIPS*, 2016.

[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[10] X. Dong and Y. Yang, "Network pruning via transformable architecture search," in *Advances in Neural Information Processing Systems*, 2019, pp. 759–770.

[11] 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 *NeurIPS*, 2018.

[12] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4340–4349.

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

[14] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017.

[15] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, 2011.

[16] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann, "Towards optimal structured cnn pruning via generative adversarial learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2790–2799.

[17] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 2130–2141.

[18] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Statistical Methodology*, 2006.

[19] 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," in *ECCV*, 2018.

[20] 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 *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 925–938.

[21] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[22] T. Goldstein, C. Studer, and R. Baraniuk, "A field guide to forward-backward splitting with a fasta implementation," *arXiv preprint arXiv:1411.3406*, 2014.

[23] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *International Conference on Learning Representations (ICLR)*, 2019.

[24] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[25] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1761–1770.

[26] E. J. Candes, M. B. Wakin, and S. P. Boyd, "Enhancing sparsity by reweighted l1 minimization," *Journal of Fourier analysis and applications*, 2008.

[27] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019.

[30] Google, https://www.tensorflow.org/mobile/tflite/, 2015.

[31] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," in *OSDI*, 2018.

[32] X. Jiang, H. Wang, Y. Chen, Z. Wu, L. Wang, B. Zou, Y. Yang, Z. Cui, Y. Cai, T. Yu, C. Lv, and Z. Wu, "Mnn: A universal and efficient inference engine," in *MLSys*, 2020.

[33] 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 *ECCV*, 2018.

[34] X. Zhu, W. Zhou, and H. Li, "Improving deep neural network sparsity through decorrelation regularization," in *IJCAI*, 2018.

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

[36] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[37] X. Ma, G. Yuan, S. Lin, C. Ding, F. Yu, T. Liu, W. Wen, X. Chen, and Y. Wang, "Tiny but accurate: A pruned, quantized and optimized memristor crossbar framework for ultra efficient dnn implementation," *ASP-DAC*, 2020.

[38] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[39] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. J. Dally, "Ese: Efficient speech recognition engine with sparse lstm on fpga." in *FPGA*, 2017.

[40] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang, and Y. Liang, "C-lstm: Enabling efficient lstm using structured compression techniques on fpgas," in *FPGA*, 2018.

[41] Z. Li, C. Ding, S. Wang, W. Wen, Y. Zhuo, X. Lin, X. Qian, and Y. Wang, "E-rnn: design optimization for efficient recurrent neural networks in fpgas," in *HPCA*, 2019.