Automatic Mapping of the Best-Suited DNN Pruning Schemes for Real-Time Mobile Acceleration

YIFAN GONG, GENG YUAN, and ZHENG ZHAN, Northeastern University
WEI NIU, College of William and Mary
ZHENGANG LI, PU ZHAO, and YUXUAN CAI, Northeastern University
SIJIA LIU, Michigan State University
BIN REN, College of William and Mary
XUE LIN, Northeastern University
XULONG TANG, University of Pittsburgh
YANZHI WANG, Northeastern University

Weight pruning is an effective model compression technique to tackle the challenges of achieving real-time deep neural network (DNN) inference on mobile devices. However, prior pruning schemes have limited application scenarios due to accuracy degradation, difficulty in leveraging hardware acceleration, and/or restriction on certain types of DNN layers. In this article, we propose a general, fine-grained structured pruning scheme and corresponding compiler optimizations that are applicable to any type of DNN layer while achieving high accuracy and hardware inference performance. With the flexibility of applying different pruning schemes to different layers enabled by our compiler optimizations, we further probe into the new problem of determining the best-suited pruning scheme considering the different acceleration and accuracy performance of various pruning schemes. Two pruning scheme mapping methods—one -search based and the other is rule based—are proposed to automatically derive the best-suited pruning regularity and block size for each layer of any given DNN. Experimental results demonstrate that our pruning scheme mapping methods, together with the general fine-grained structured pruning scheme, outperform the state-of-the-art DNN optimization framework with up to 2.48× and 1.73× DNN inference acceleration on CIFAR-10 and ImageNet datasets without accuracy loss.

CCS Concepts: • **Computing methodologies → Machine learning**; **Neural networks**; **Machine learning approaches**; **Machine learning algorithms;**

Additional Key Words and Phrases: Network pruning, mobile acceleration, neural architecture search

Y. Gong and G. Yuan contributed equally to this research.

This research was partially funded by the National Science Foundation (CCF-1919117, CNS-1909172, CCF-2047516 (CAREER), and CCF-1901378), and Jeffress Trust Awards in Interdisciplinary Research to William & Mary. 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 or Thomas F. and Kate Miller Jeffress Memorial Trust.

Authors' addresses: Y. Gong, G. Yuan, Z. Zhan, Z. Li, P. Zhao, Y. Cai, X. Lin, and Y. Wang, Northeastern University, Boston, MA; emails: gong.yifa@northeastern.edu, yuan.geng@northeastern.edu, zhan.zhe@northeastern.edu, li.zhen@northeastern.edu, zhao.pu@northeastern.edu, cai.yuxu@northeastern.edu, xue.lin@northeastern.edu, yanz.wang@northeastern.edu; W. Niu and B. Ren, College of William and Mary, Williamsburg, VA; emails: wniu@email.wm.edu, bren@cs.wm.edu; S. Liu, Michigan State University, East Lansing, MI; email: liusiji5@msu.edu; X. Tang, University of Pittsburgh, Pittsburgh, PA; email: tax6@pitt.edu.

1084-4309/2022/06-ART47 $15.00
https://doi.org/10.1145/3495532

ACM Reference format:
Yifan Gong, Geng Yuan, Zheng Zhan, Wei Niu, Zhengang Li, Pu Zhao, Yuxuan Cai, Sijia Liu, Bin Ren, Xue Lin, Xulong Tang, and Yanzhi Wang. 2022. Automatic Mapping of the Best-Suited DNN Pruning Schemes for Real-Time Mobile Acceleration. *ACM Trans. Des. Autom. Electron. Syst.* 27, 5, Article 47 (June 2022), 26 pages.
https://doi.org/10.1145/3495532

1 INTRODUCTION

Model compression techniques have been proposed to reduce computation and memory intensity without compromising accuracy [22, 26, 40, 57, 69, 81, 82]. It is a promising solution for achieving various practical **deep learning (DL)** based methods such as fingerprinting [31], YOLO [9], super-resolution [79], and speech recognition [18] in real time on resource-limited platforms, especially mobiles and embedded devices [37, 58, 84]. Among the compression techniques, weight pruning [22, 26, 27, 57, 69] explores and reduces the vast redundancy in the number of weights and results in structural sparsity of **deep neural network (DNN)** models with fewer memory references and less power consumption during inference.

The design of a weight pruning method includes two fundamental aspects: pruning regularity and pruning algorithm. The former refers to the structural characteristics of the DNNs after pruning, whereas the latter determines the rule to identify the weights to be pruned. From the *pruning regularity* aspect, the widely adopted pruning schemes include unstructured pruning, structured pruning, and pattern-based pruning. Specifically, unstructured pruning is flexible to prune any weights and generally yields promising accuracy, but they are not compatible with hardware accelerations due to the irregular computation after pruning [22, 23, 48]. However, structured pruning eliminates weights while maintaining a full matrix format. It is hardware-friendly but suffers from notable accuracy degradation due to the coarse-grained nature in pruning whole filters/channels [45, 54, 57, 83, 86, 87]. Recently proposed pattern-based pruning overcomes the shortcomings of prior works by incorporating fine-grained structured pruning in a hardware-aware fashion [50, 59], with the aid of compiler. However, pattern-based pruning is only applicable to 3×3 **convolutional (CONV)** layers and is difficult to be generalized to **fully connected (FC)** layers and CONV layers with other kernel sizes. There is a lack of pruning regularity that is general and achieves high accuracy and hardware performance simultaneously.

From the *pruning algorithm* aspect, heuristic-based pruning was first proposed by Han et al. [23] and gets improvements with more sophisticated designed heuristics [19, 27, 36, 49, 74, 87]. Regularization-based pruning [21, 26, 39, 41, 43, 55, 56, 62, 69, 76, 77, 81], however, is more mathematics oriented. Recent works [39, 51, 62, 81, 82] achieve substantial weight reduction without hurting the accuracy by leveraging **alternating direction methods of multipliers (ADMM)** with dynamic regularization penalties, but these methods require the manual setting of the compression rate for each layer.

To fully exploit the potential of the pruned models on mobile devices for inference accelerations, it is necessary to incorporate compiler optimizations to support efficient sparse computation and storage. However, state-of-the-art compiler-based DNN execution frameworks such as **TensorFlow Lite (TFLite)** [1], **Alibaba Mobile Neural Network (MNN)** [2], and TVM [11] do not support sparse (pruned) model inference acceleration on the mobile platforms, whereas the recent works PCONV [50] and PatDNN [59] only have limited sparse inference support for 3×3 CONV layers.

Apart from the individual limitations mentioned previously, there is one additional deficiency that prevents DNN models from taking full advantage of weight pruning. Different pruning schemes result in different acceleration and accuracy performance, but prior works simply apply

the same pruning scheme to the entire model, undermining the flexibility to select the best-suited pruning scheme for each layer to achieve better accuracy and acceleration performance.

This work aims to overcome the preceding limitations of prior works. More specifically, we make the following contributions toward a general, fine-grained structured pruning scheme and two automatic pruning scheme mapping methods.

For the pruning scheme part:

- We propose a novel and general pruning regularity, block-based pruning for FC layers and block-punched pruning for CONV layers with different kernel sizes, which can achieve high accuracy and high hardware inference performance simultaneously.
- We adopt a reweighted dynamic regularization algorithm to derive the structured sparsity with an automatically determined compression rate for each layer and each block without compromising accuracy.
- To extract the fine-grained structure information and exploit hardware parallelism, we propose a compiler-based mobile acceleration framework that supports the proposed pruning regularity as well as other pruning regularities. It provides the flexibility to apply different pruning schemes to different layers for better performance of the pruned model.

With regard to the automatic pruning scheme mapping methods part, taking the different acceleration and accuracy performance of various pruning schemes into consideration, we probe into the new problem of determining the best-suited pruning scheme for each layer of any given DNN. We propose two automatic pruning scheme mapping methods to address this problem. More specifically:

- The first is a *search-based* method leveraging the recent concept of network architecture search [8, 67, 70, 85, 88], which employs **reinforcement learning (RL)** technique to yield close-to-optimal pruning scheme mappings.
- The second is a training-free, *rule-based* method leveraging an offline-generated latency model. It is efficient and more useful in practice.

We perform comprehensive evaluations of the proposed general pruning scheme and the two mapping methods on representative DNN models and benchmark datasets. Experimental results demonstrate that our methods significantly outperform the state-of-the-art DNN pruning framework PatDNN in terms of accuracy and latency performance. We achieve 17.22-, 18.17-, and 3.90-ms ImageNet inference time with negligible accuracy loss on an off-the-shelf mobile phone for ResNet-50, VGG-16, and MobileNetV2, respectively. Furthermore, the search-based method only shows a slightly better performance than the rule-based method, whereas the rule-based method is training-free in pruning scheme mapping.

2 BACKGROUND AND RELATED WORKS

2.1 DNN Pruning: Regularity and Algorithm

2.1.1 Pruning Regularity. From the pruning regularity aspect, existing pruning schemes can be divided into three categories: fine-grained unstructured pruning, coarse-grained structured pruning, and pattern-based pruning. We show the different pruning regularities in Figure 1, with colored grids representing remaining weights. The left and middle columns in the figure illustrate pruning regularities in the 4D weight tensor format and 2D weight matrix format for CONV layers, respectively. The right column shows the different regularities for FC layers.

Unstructured pruning is fine-grained and flexible in removing weights at arbitrary locations [15, 16, 20, 22], as shown in Figure 1(a) and (b). Although having the advantage in maintaining accuracy, unstructured pruning leads to sparse and irregular weight matrices, and as a result, indices are

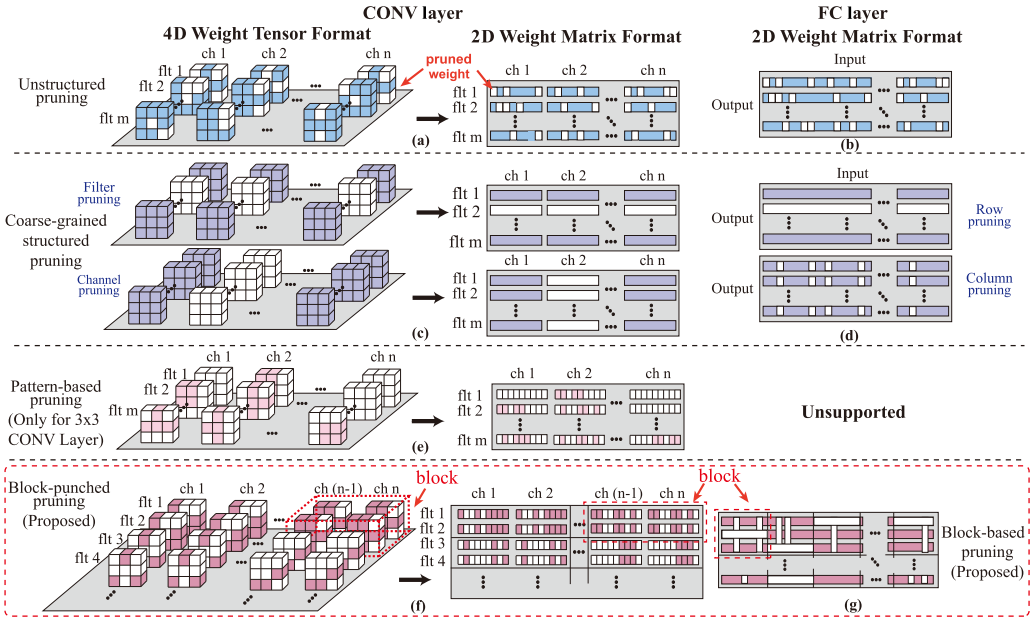

Fig. 1. Different weight pruning schemes for CONV and FC layers using 4D tensor and 2D matrix representation.

required to locate the non-zero weights in the sparse matrix storage format (e.g., **compressed sparse row (CSR)** format). Therefore, it cannot effectively and efficiently leverage the hardware parallelism provided by the underlying system. Consequently, unstructured pruning is generally not compatible with GPU acceleration for DNN inference, and speed degradation can often be observed [52].

Structured pruning [27, 28, 69, 74, 75] focuses on CONV layers and maintains structured regularity. It consists of filter pruning and channel pruning that prune the entire filter(s)/channel(s). In the weight matrix format representation as shown in Figure 1(c), filter pruning corresponds to reducing one row of the weight matrix, and it is also called *row pruning*. Accordingly, channel pruning corresponds to reducing multiple consecutive columns. The key advantage of structured pruning is that a full matrix will be maintained with dimension reduction, thereby facilitating hardware acceleration. However, structured pruning is coarse-grained and often leads to certain accuracy degradation [59, 68].

Pattern-based pruning [50, 59, 78] alleviates the shortcomings of prior works by incorporating the benefits from fine-grained pruning while maintaining structures that can be exploited for hardware accelerations with the help of compiler. Pattern-based pruning is a combination of kernel pattern pruning and connectivity pruning as shown in Figure 1(e). Kernel pattern pruning prunes weights at an intra-kernel level by enforcing the locations of the remaining weights in a kernel to form a specific kernel pattern. Different kernels can have different kernel patterns, but the total types of kernel patterns are restricted to a fixed-size set. Each kernel pattern reserves four non-zero weights to match the single-instruction multiple-data (SIMD) architecture of embedded CPU/GPU processors to maximize the hardware throughput. As a *fixed* number of weights are pruned, the compression rate is constant for kernel pattern pruning. For a higher compression rate, connectivity pruning is adopted as the supplementary to kernel pattern pruning.

Connectivity pruning prunes weights at an inter-kernel level via cutting the connections between certain input and output channels.

However, pattern-based pruning is designed for 3×3 CONV layers and suffers difficulty when generalized to CONV layers with other kernel sizes and FC layers. To avoid the execution overhead of branching conditions caused by using different pattern types, pattern-based pruning requires limiting the maximum number of different pattern types to be used. Generally, 8 or 16 different pattern types are allowed to be selected from all possible 4-entry pattern combinations to ensure a decent acceleration while not hurting accuracy. For larger kernel sizes such as $5 \times 5/7 \times 7$, 4-entry patterns need to be selected from 25/49 weights (instead of 9 weights in the 3×3 case), making the pattern have too many potential candidates. As a result, if only 8 or 16 patterns are used, there will be an accuracy degradation. Moreover, as studied in the work of Ma et al. [53], the Gaussian filter-like patterns and the enhanced Laplacian of Gaussian (ELoG) filter-like patterns (as shown in Figure 1(e)) are more preferred since they can provide an enhancement on feature extraction. But such 4-entry patterns in $5 \times 5/7 \times 7$ kernels cannot provide the receptive field size that the large kernels are supposed to have. For the 1×1 CONV layer, there is only one weight in a kernel, making the pattern-based pruning same as unstructured pruning, which is hard to achieve actual acceleration. Therefore, the existing pattern-based pruning is only suitable for 3×3 kernels, which significantly restricts the application scenarios of pattern-based pruning.

2.1.2 Pruning Algorithm. There are two main categories of the pruning algorithm: heuristic based and regularization based. The heuristic-based pruning algorithm was first proposed to achieve unstructured pruning by pruning weights with small magnitudes in an iterative manner [23]. Later heuristic works were improved in multiple directions including structured-preserving pruning [36, 49, 74], combining growth of neurons and connections with pruning [16], and introducing meticulously designed criteria [27, 49, 74, 87] to replace magnitudes for the pruning.

The regularization-based algorithm deals with the pruning problem using a more mathematics-oriented method. To solve filter/channel pruning problems, early works [28, 69] incorporate ℓ_1 or ℓ_2 structured regularization in the loss function. The work of Liu et al. [46] introduces a scaling factor to each channel while imposing ℓ_1 regularization on the scaling factors in batch normalization to prune channels with near-zero scaling factors. However, these works directly apply fixed regularization terms that penalize all weights equally, incurring potential accuracy loss. Later works [21, 62, 81] adopt ADMM to reform the pruning problem as optimization problems with dynamic regularization penalties, thus preserving accuracy. One drawback of these methods is the requirement for the manual setting of the compression rate for each layer.

2.2 Compiler-Based DNN Frameworks on Mobile

Mobile devices become key carriers of DL [29, 34, 60, 80] to enable the widespread of machine intelligence. To facilitate the deployment of various DNN models on mobile devices, multiple mobile DNN execution frameworks from both industry and academia attract broad attention [24, 30, 33, 35, 71, 73]. TFLite [1], MNN [2], and TVM [11] are three representative state-of-the-art end-to-end DNN execution frameworks with high execution efficiency. They employ several performance optimization techniques, such as various computation graph optimizations, tensor optimizations, and half-float support. Particularly, TVM includes a more advanced parameter auto-tuning technique. However, none of these frameworks support sparse (pruned) DNN models on mobile platforms.[1] This is the essential drawback that obstructs the real-time DNN inference on mobile devices. Taking the VGG-16 network—one of the key DNN models in transfer learning—as

[1]TVM considers sparsity recently for desktop processors.

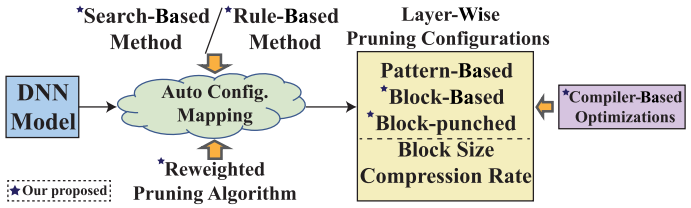

Fig. 2. High-level overview of the proposed automatic pruning scheme mapping framework.

an example, TVM takes 200 ms to perform an inference on the embedded GPU (Adreno 640), and TFLite takes even longer time (270 ms).

Previous efforts based on fine-grained pattern-based pruning such as PatDNN [59] and PCONV [50] employ a set of compiler-based optimizations to support sparse DNN models, significantly accelerating the end-to-end DNN inference on mobile devices. However, they mainly accelerate the square and small convolution kernels used in 3×3 CONV layers. A larger kernel size (e.g., 5×5, 7×7) will introduce huge code execution overhead due to the increasing number of branches in generated code. In addition, they cannot support FC layers and 1×1 CONV layers that are commonly used in DNNs.

3 OVERVIEW OF THE AUTOMATIC PRUNING SCHEME MAPPING FRAMEWORK

To achieve real-time mobile acceleration for various modern DNNs, we propose an automatic pruning scheme mapping framework, which is illustrated in Figure 2. Given an arbitrary DNN model, the framework can automatically map the best pruning configurations to each layer and leverage compiler-based optimizations to achieve inference speedup. The layer-wise configurations include the pruning regularity, compression rate, and block size.

To achieve the design objective, our framework contains the following innovations. We first propose a general, fine-grained pruning regularity that is applicable to *different* types of layers while achieving both high accuracy and hardware acceleration performance to overcome the limitations of prior pruning regularities in Section 4.1. To determine the compression rate for each layer automatically without compromising accuracy, we introduce a reweighted pruning algorithm in Section 4.2. For the goal of transforming compression to real inference speedup on mobile devices, we propose corresponding compiler-based optimizations that support the proposed pruning regularity as well as other pruning regularities in Section 4.3. As directly applying the same pruning scheme to the entire model cannot yield the optimal performance, we further propose to map the best-suited pruning configurations to each layer of any given DNN for mobile devices thanks to the flexibility enabled by our compiler optimizations. The mapping methods include a comprehensive search-based method that can provide close-to-optimal results in Section 5.1 and a training-free rule-based method that is more useful in practice while reaching similar performance as the search-based method in Section 5.2.

4 GENERAL FINE-GRAINED STRUCTURED PRUNING SCHEME

In this section, we present a novel fine-grained structured pruning scheme and corresponding compiler optimizations to (i) achieve high accuracy and hardware inference performance simultaneously while applicable to different types of layers, (ii) determine the compression rate for each layer automatically without compromising the accuracy, and (iii) provide the supports to the proposed pruning regularity and other pruning regularities for the exploitation of the hardware parallelism.

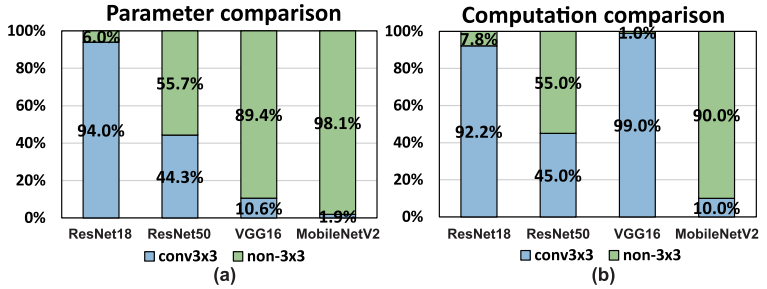

Fig. 3. Comparisons of parameter ratio (a) and computation ratio (b) for 3 × 3 CONV layers and non-3 × 3 layers for different networks on the ImageNet dataset.

We start by providing a general fine-grained structured pruning regularity that includes block-based pruning for FC layers and block-punched pruning for CONV layers with different kernel sizes in Section 4.1. Next, a reweighted dynamic regularization algorithm that allows the automatic determination of the per-layer and per-block compression rate is introduced to derive the sparse regularity in Section 4.2. Then we provide corresponding compiler optimizations for the proposed pruning scheme to enable efficient on-device inference of the pruned model in Section 4.3.

4.1 General Fine-Grained Structured Pruning Regularity

Although state-of-the-art pattern-based pruning strikes a desirable balance between accuracy and hardware efficiency, it *only* works for CONV layers with 3 × 3 kernels and suffers difficulty when generalized to layers with other kernel sizes and FC layers. Note that not all of the layers only operate on 3 × 3 kernels in a given DNN model. As a result, the number of layers using 3 × 3 kernels affect the effectiveness of pattern-based pruning. Figure 3 illustrates the percentage of the parameters and **multiply-and-accumulates (MACs)** in 3×3 CONV layers of four representative networks. The large portion of non-3 × 3 CONV layers leaves great space for higher compression rate and faster inference that cannot be achieved by pattern-based pruning alone.

To alleviate the deficiencies, we propose a *general* pruning scheme with fine-grained structured pruning regularity, including block-based pruning for FC layers and block-punched pruning for CONV layers with different kernel sizes.

4.1.1 Block-Based Pruning for FC Layers. Block-based pruning is an extension of the coarse-grained structured pruning that prunes rows/columns in matrix-based computation for FC layers. As shown in Figure 1(g), we divide a whole weight matrix of an FC layer to a number of equal-sized blocks (4 × 4, 16 × 32, 64 × 128, etc.), and apply independent row and column pruning for each block. The compression rate (the number of pruned rows/columns) for each block can either be the same or different, which depends on the design requirements.

4.1.2 Block-Punched Pruning for CONV Layers. Compared with matrix-based representation and computation, tensor-based representation and computation are more suitable for CONV layers. Thus, inspired by block-based pruning, we further propose block-punched pruning that is tailored for CONV layers and can be accelerated with the same compiler optimizations. As shown in Figure 1(f), block-punched pruning first partitions the weight tensor of a CONV layer into groups (blocks) of kernels along the filter and input channel dimensions. For each block, the weights at the same locations for all kernels within the block are pruned. With effective compiler-level executable code generation, high hardware parallelism and inference acceleration on mobile can be achieved.

Table 1. Comparison of Different Pruning Algorithms

	Accuracy	Compression Rate
GroupLasso	Low	Auto
ADMM	High	Manual
Reweighted	High	Auto

Compared with state-of-the-art pattern-based pruning, the proposed fine-grained structured pruning regularity is *general and flexible*, as it can adaptively prune FC layers and CONV layers with different kernel sizes. In addition, block-based pruning and block-punched pruning can simultaneously achieve high accuracy and high hardware inference performance like pattern-based pruning. The *high accuracy* is attributed to the fine-grained property of pruning regularity, which allows higher flexibility when searching the pruned model structure compared to coarse-grained structured pruning that prunes entire rows/columns in weight matrices. However, the *high hardware inference performance* is attributed to the appropriate degree of structural regularity, which can be exploited by compiler-level code generation to achieve high or even maximum hardware parallelism. With an appropriate selection of the block size, the remaining entries in each block can still be sufficient to exploit high hardware parallelism. The block size for each layer is an important hyperparameter that influences hardware performance and accuracy. We will elaborate on how to select the appropriate block size for each layer in Section 5.2.2.

4.2 Reweighted Dynamic Regularization Algorithm

Another important design aspect of a pruning scheme is the pruning algorithm. Prior pruning algorithms, such as using group Lasso regularization [28, 46, 69] or ADMM [39, 61, 81], either suffer from potential accuracy loss or require maual compression rate tuning. To overcome the limitations, we propose to adopt the reweighted group Lasso [10] method to discover the structured sparsity with systematically and dynamically reweighted penalties. More specifically, the reweighted method reduces the penalties on weights with larger magnitudes, which are likely to be more critical weights, and increases the penalties on weights with smaller magnitudes. A comparison of the characteristics of different regularization-based pruning algorithms is shown in Table 1.

For the i-th layer in the DNN, if the layer is an FC layer, let $\boldsymbol{W}_i \in \mathbb{R}^{P_i \times Q_i}$ denote the 2D weight matrix, with P_i and Q_i indicating the rows and columns of the weight matrix; otherwise, $\boldsymbol{W}_i \in \mathbb{R}^{P_i \times Q_i \times K_i^h \times K_i^w}$ represents the 4D weight tensor of a CONV layer, where P_i is the number of filters, Q_i is the number of input channels, and K_i^w and K_i^h are the kernel width and kernel height. Let $\boldsymbol{b}_i \in \mathbb{R}^{P_i}$ represent the bias for the i-th layer. We also define $\boldsymbol{W} := \{\boldsymbol{W}_i\}_{i=1}^N$ and $\boldsymbol{b} := \{\boldsymbol{b}_i\}_{i=1}^N$ as the set of all weights and biases in the DNN. We denote the loss of the DNN under dataset $\mathcal{D}$ by $f(\boldsymbol{W}, \boldsymbol{b}; \mathcal{D})$. Each $\boldsymbol{W}_i$ is divided into J blocks with the same size, $p_i \times q_i$ for an FC layer and $p_i \times q_i \times K_i^h \times K_i^w$ for a CONV layer, namely $\boldsymbol{W}_i = [\boldsymbol{W}_{i1}, \boldsymbol{W}_{i2}, \ldots, \boldsymbol{W}_{iJ}]$, where $\boldsymbol{W}_{ij} \in \mathbb{R}^{p_i \times q_i}$ for a FC layer and $\boldsymbol{W}_{ij} \in \mathbb{R}^{p_i \times q_i \times K_i^h \times K_i^w}$ for a CONV layer. The general reweighted pruning problem is formulated as

$$\underset{\boldsymbol{W}, \boldsymbol{b}}{\text{minimize}} \quad f(\boldsymbol{W}, \boldsymbol{b}; \mathcal{D}) + \lambda \sum_{i=1}^{N} R\left(\boldsymbol{\alpha}_i^{(t)}, \boldsymbol{W}_i\right), \tag{1}$$

where λ is the hyperparameter to adjust the relative importance between accuracy and sparsity. Let $\boldsymbol{\alpha}_i^{(t)}$ denote the collection of penalty values applied on the weights $\boldsymbol{W}_i$ for layer i at step t. Note that each element in $\boldsymbol{\alpha}_i^{(t)}$ is a positive value that is determined by the reweighted ℓ_1 algorithm [10].

For block-based row pruning, the regularization term is

$$R\left(\boldsymbol{\alpha}_i^{(t)}, \boldsymbol{W}_i\right) = \sum_{j=1}^{J} \sum_{m=1}^{p_i} \left\| \alpha_{ijm}^{(t)} \circ [\boldsymbol{W}_{ij}]_{m,:} \right\|_F^2, \tag{2}$$

where the operator $\circ$ represents element-wise multiplication, $[\boldsymbol{W}_{ij}]_{m,:}$ denotes the m-th row of $\boldsymbol{W}_{ij}$, and $\alpha_{ijm}^{(t)}$ is updated by $\alpha_{ijm}^{(t)} = \frac{1}{\|[\boldsymbol{W}_{ij}]_{m,:}^t\|_F^2 + \epsilon}$ to help increase the degree of sparsity beyond group Lasso regularization.

For block-based column pruning, the regularization term is

$$R\left(\boldsymbol{\alpha}_i^{(t)}, \boldsymbol{W}_i\right) = \sum_{j=1}^{J} \sum_{n=1}^{q_i} \left\| \alpha_{ijn}^{(t)} \circ [\boldsymbol{W}_{ij}]_{:,n} \right\|_F^2, \tag{3}$$

where $[\boldsymbol{W}_{ij}]_{:,n}$ is the n-th column of $\boldsymbol{W}_{ij}$ and $\alpha_{ijn}^{(t)}$ is updated by $\alpha_{ijn}^{(t)} = \frac{1}{\|[\boldsymbol{W}_{ij}]_{:,n}^t\|_F^2 + \epsilon}$. The block-based row pruning problem (2) and column pruning problem (3) can be solved separately or simultaneously using a standard DL solver.

For block-punched pruning, the regularization term is formulated as

$$R\left(\boldsymbol{\alpha}_i^{(t)}, \boldsymbol{W}_i\right) = \sum_{j=1}^{J} \sum_{m=1}^{K_i^h} \sum_{n=1}^{K_i^w} \left\| \alpha_{ijmn}^{(t)} \circ [\boldsymbol{W}_{ij}]_{:,:,m,n} \right\|_F^2, \tag{4}$$

where $[\boldsymbol{W}_{ij}]_{:,:,m,n}$ indicates the weight located at the m-th row and n-th column in a kernel for all kernels in the block and $\alpha_{ijmn}^{(t)} = \frac{1}{\|[\boldsymbol{W}_{ij}]_{:,:,m,n}^t\|_F^2 + \epsilon}$. The reweighted method only requires the hyperparameter λ, and the soft constraints formulation allows the automatic determination of the compression rate for each layer and each block.

4.3 Compiler Optimizations for Proposed Pruning Regularity

Compiler optimizations can turn the sparsity of pruned models into higher speedups. Without compiler optimizations, the pruned weights (with zero values) still participate in the inference computations, resulting in minor inference speedup. Hence, we develop a comprehensive compiler-based automatic code generation framework to extract the fine-grained structure information in block-punched and block-based pruning. The framework also supports other pruning regularities including unstructured pruning, structured pruning, and pattern-based pruning. Our proposed compiler-based mobile acceleration framework first compacts the model storage with a novel compression format called **blocked compressed storage (BCS)** format, as shown in Figure 4. Then, it performs computation reordering to reduce the branches within each thread and eliminate the load imbalance among threads.

BCS stores non-zero weights as CSR format with a better compression rate by further compressing the index with a hierarchical structure. Traditional CSR format has to store each non-zero weight with an explicit column index. Our proposed block-based/block-punched pruning preserves non-zero weights in identical columns within each block, inducing many repeated column indices if we use CSR format. BCS eliminates this redundancy with a hierarchical compression on the column index only.

Figure 4 shows a simplified example. The `weights array` stores all non-zero weights. The `compact column array` stores the compressed column index—for example, [0, 3, 6] denotes the column id of the first three weights [1, 2, 3]. The `column stride array` denotes the start and end index of each row in the compact column array—for example, [0, 3] denotes that the column index for the first row starts from index 0 and ends at index 2 in the compact column array. The same column indices may be used for multiple rows. The `occurrence array` is used to specify the

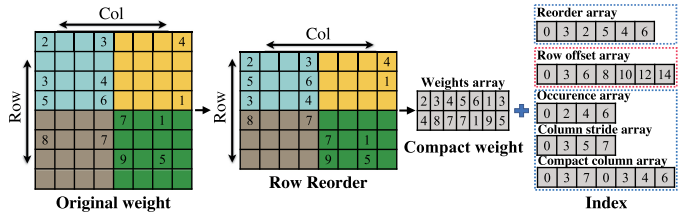

Fig. 4. BCS for weights.

start and end rows with the identical column index—for example, [0, 2] means that rows 0 and 1 share the same column index. BCS also contains a row offset array to specify the starting location of each row in the weights array.

Usually, the weight distribution is not as regular as the preceding simplified example, thus a row reordering optimization is also included to further improve the regularity of the weight matrix. After this reordering, the continuous rows with identical or similar numbers of non-zero weights are processed by multi-threads simultaneously, thereby eliminating thread divergence and achieving load balance. Each thread processes more than one row, thus eliminating branches and improving instruction-level parallelism. We also incorporate other compiler-based optimizations for on-mobile DNN inference acceleration, such as layer fusion, auto-tuning, and high-level **domain-specific language (DSL)**. More details are provided in the Appendix.

4.4 Effectiveness of the Proposed Pruning Scheme

We show an example of the inference accuracy and acceleration performance of the proposed pruning scheme on ResNet-50 using the ImageNet dataset in Figure 5. More thorough evaluation results are presented in Section 6.2. Here, block-based pruning is applied for *all* FC layers and block-punched pruning is applied for *all* CONV layers. The compression rate for each layer is derived by the reweighted dynamic regularization algorithm. As can be seen from the figure, unstructured pruning, which is equivalent to setting the block size as 1×1 for each layer, achieves the highest accuracy but the worst performance in latency. In contrast, structured pruning (i.e., using the whole matrix as the block size) achieves the fastest inference but degrades the accuracy the most. With a suitable block size, our proposed fine-grained structured pruning scheme achieves high accuracy and inference speed simultaneously. The reason is that the maximal hardware parallelism is limited by the computation resource. Since the weight matrix/tensor is typically very large, the remaining entries in each block are still sufficient to exploit high hardware parallelism. With parallelism maximally exploited, the hardware inference performance can be almost the same as coarse-grained structured pruning.

Takeaway. In this section, we first introduced a general fine-grained structured pruning regularity, which can work for CONV layers with any kernel size and FC layers. Second, we proposed the reweighted group Lasso with block-based constraints as the pruning algorithm to derive the structured sparsity with an *automatically* determined compression rate for each layer and each block. Third, we developed the first compiler-based mobile acceleration framework that supports general block-based/block-punched sparsity as well as other pruning regularities, which is flexible and allows different layers to adopt different pruning regularities and block sizes.

5 AUTOMATIC PRUNING SCHEME MAPPING METHODS FOR MOBILE DEVICES

Although the general fine-grained pruning scheme proposed in Section 4 can achieve high accuracy and hardware acceleration performance, it is not optimal to directly apply the same pruning

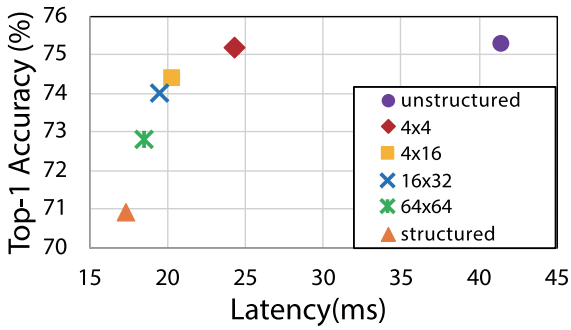

Fig. 5. Accuracy and latency performance with different block sizes on ResNet-50 using the ImageNet dataset.

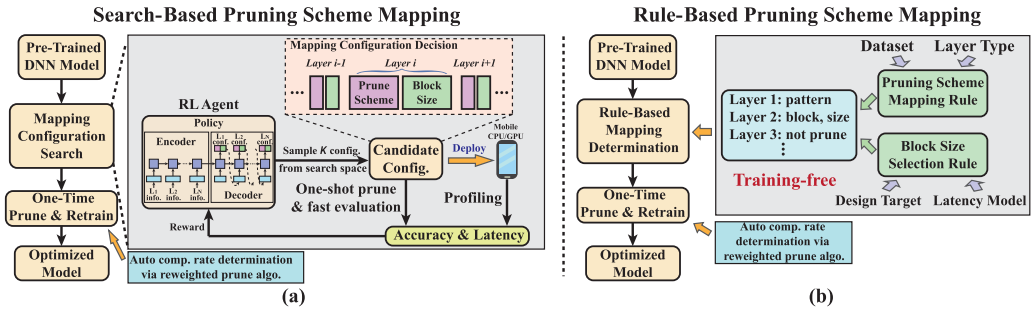

Fig. 6. Overview of search-based pruning scheme mapping (a) and rule-based pruning scheme mapping (b).

scheme to the entire model, as different layers may prefer different pruning regularities and configurations (e.g., the compression rate and block size). Fortunately, effective compiler optimization techniques provide the flexibility to apply different pruning regularities and block sizes to different layers. As different weight pruning schemes have different acceleration and accuracy performance under the same mobile acceleration framework, it is important to have a pruning scheme mapping method to determine the pruning configurations for each layer. Therefore, we further probe into the problem of mapping the best-suited pruning scheme for each layer of any given DNN to obtain a pruned model with better performance in terms of accuracy and latency in this section.

The performance of a pruned model is influenced by the compression rate, pruning regularity, and block size when block-based/block-punched pruning is selected, of each layer. This is a new challenge that resulted from the new dimension of compiler-aware pruning scheme optimizations. To find the appropriate pruning schemes in such a large design space, we propose two automatic pruning scheme mapping methods: one is search based and the other is rule based, as shown in Figure 6. The former is a more comprehensive framework to yield close-to-optimal pruning scheme mapping results, whereas the latter is a *training-free* procedure that is efficient and more useful in practice. Note that with our proposed reweighted dynamic regularization algorithm in Section 4.2, the compression rate can be obtained automatically for each layer and each block. Thus, the search space of the pruning scheme mapping problem can be reduced to finding the appropriate pruning regularity and the block size for each layer in the given DNN.

5.1 Search-Based Pruning Scheme Mapping Method

Although we simplify the search space with the reweighted dynamic regularization algorithm to determine the per-layer and per-block compression rate automatically, there is still a huge amount

of combinations of pruning regularities and block sizes to seek. Inspired by recent advances in network architecture search [8, 67, 70, 85, 88], we consider to leverage a search-based method by employing RL [38, 65] to map the appropriate pruning scheme for each layer of a given DNN.

In RL, an agent interacts with the environment by taking an action $a_t \in A$ according to a policy π upon the observation of a state $s_t \in S$ at timestep t. For our problem, each timestep t corresponds to the pruning scheme mapping of one layer. The state $s_t \in S$ represents the information of current layer, which is defined as a 4D vector {*layer type*, *kernel size*, *input channel number*, *output channel number*}. The action $a_t \in A$ is the mapping decision for the current layer, which is a 2D vector {*pruning regularity*, *block size*}. For an N-layer DNN with information $\mathcal{I} = \{s_1, \ldots, s_N\}$, an entire mapping $\mathcal{M} = \{a_1, \ldots, a_N\}$ can be found with N timesteps. Let $R(\mathcal{M})$ denote the cumulative reward for $\mathcal{M}$, which is the optimization target of the RL agent. A good pruning scheme mapping should achieve high accuracy and hardware performance jointly, thus we define $R(\mathcal{M})$ as the weighted sum of the accuracy and the negative of the latency of the pruned model with information $\mathcal{I}$ under the mapping $\mathcal{M}$.

We leverage the policy gradient method [66] to directly learn a parameterized policy for the pruning scheme mapping, and the training objective of the policy is defined as follows:

$$J(\theta) = \mathbb{E}_{\mathcal{M}\sim\pi(\mathcal{M}|\mathcal{I};\theta)}[R(\mathcal{M})|\mathcal{I}], \tag{5}$$

where $\pi(\mathcal{M}|\mathcal{I};\theta)$ is a sequence-to-sequence model in our work. The input to the encoder recurrent neural network is the sequence of the information of each layer in the target DNN, and the decoder is an LSTM with N timesteps to output the mapping decision for each layer at the same encoder timestep. We estimate the gradient of the objective function by drawing K mapping decision samples from $\mathcal{M}_k \sim \pi(\mathcal{M}|\mathcal{I};\theta)$ and reduce the variance of the estimate with a baseline term B, leading to

$$\nabla_\theta J(\theta) \approx \frac{1}{K}\sum_{k=1}^{K}(R(\mathcal{M}_k) - B)\cdot\nabla_\theta \log \pi(\mathcal{M}_k|\mathcal{I};\theta). \tag{6}$$

For each mapping decision sample M_k in a training iteration of the policy, we need to compress the target DNN to obtain the accuracy and latency performance for the calculation of the reward $R(\mathcal{M}_k)$. The latency is obtained via deploying the pruned model with compiler code generation on target device and measuring the real execution time. To accelerate the policy training, we adopt magnitude-based one-shot pruning and early stopping for faster accuracy evaluation during the policy training process. More specifically, once a mapping $\mathcal{M}_k$ is obtained, we conduct a one-shot pruning for each layer of the DNN based on the weight magnitude and retrain the DNN for two epochs to regain accuracy. This partially regained accuracy can be used to predict the final model accuracy and compare the performance between different schemes [67, 85]. Furthermore, as compiler code generation and latency measurement do not depend on absolute weight values and are faster than DNN training, we overlap the compiler code generation and latency measurement with the accuracy evaluation of the pruned model.

5.2 Rule-Based Pruning Scheme Mapping Method

The advantage of the search-based method is that it can find the globally close-to-optimal pruning configurations for each of the layers in a given DNN. Although it works perfectly for small DNN models, the searching overheads increase exponentially when the model size increases, making it unsuitable for large-size DNN models. Therefore, we design a *training-free rule-based* method that maps the best-suited pruning schemes in a layer-wise fashion to avoid the time-consuming search process for the best mapping. We consider the search-based solution as the performance upper

bound, and we target to make the rule-based method perform as well as the search-based one, yet highly efficient and practical.

5.2.1 Latency Model. To obtain the latency performance without the pruning and retraining of the given DNN, we build latency models for different types of layers (e.g., 1×1 CONV, 3×3 CONV, 5×5 CONV, and **3×3 depth-wise (3×3-DW)** CONV) on the target device (e.g., a Samsung S10 smartphone). Each latency model contains latency results for different settings, including block size, number of filters, input and output feature map size, pruning scheme, and compression rate. The results are measured on the target device by running test models with each setting for 100 runs. Each test model has 10 cascaded layers with the same setting. Since building the latency model does not involve DNN training, it will not take a very long time. The testing time for each run of each setting is in the milliseconds level. For instance, our latency model including 512 different layer settings can be built in around 30 minutes. Such a building time is negligible compared to the DNN training or the searching process, which usually counts in days. The latency model only needs to be built once for a target device and is universal to different DNN models.

5.2.2 Block-Size Selection. Block size has a significant impact on the accuracy and hardware performance for block-based/block-punched pruning. A larger block size is typically more hardware-friendly and easier to leverage the built-in hardware acceleration, yet it may cause more severe accuracy degradation due to the coarse granularity. On the contrary, a smaller block size typically leads to higher accuracy but also increases the latency. An appropriate setting of the block size can achieve high accuracy as unstructured pruning (essentially with block size 1×1) and high hardware acceleration performance as structured pruning (essentially with the block size of the whole weight tensor/matrix) simultaneously.

To determine the proper block size for each layer without the requirement of a time-consuming training process, we consider decoupling the two optimization targets: accuracy and hardware performance. To minimize the impact of pruning on hardware performance, our rule-based method will first derive the inference latency of each block size from the *offline-generated* latency models and normalize the latency (i.e., divide by the MACs of that layer). We introduce a latency threshold β, indicating the acceptable latency degradation range of the proposed general pruning regularity compared with coarse-grained structured pruning. The value of β can be adjusted according to the design requirement, and it can either be the same for the entire model or different for each layer. For example, $\beta = 20\%$ means that the inference speed of block-based/block-punched pruning can be at most 20% slower than structured pruning under the same compression rate. After the hardware performance-driven design is satisfied, we only need to consider the influence of block size on accuracy. As a smaller block size can provide a finer granularity in pruning and the consequent higher accuracy, the smallest valid block size that satisfies the β-degradation requirement is selected as the desired block size. This process depends on our latency model and is free of training.

5.2.3 3×3 CONV Layer: Pattern or Block. For 3×3 CONV layers, both pattern-based pruning and block-punched pruning can be applied. To map the best-suited pruning scheme, the problem is to compare the accuracy and inference latency of block-punched pruning and pattern-based pruning.

Accuracy perspective. To investigate the accuracy of pattern-based pruning and block-punched pruning, we conduct comprehensive experiments on ResNet-18 and VGG-16 with CIFAR-10 and ImageNet datasets. Figure 7(a) and (b) show an example of the comparison results on the CIFAR-10 dataset, and the block size is set to 4×16. Note that only 3×3 CONV layers are pruned and non-3×3 layers remain unpruned to provide a fair comparison. Here, the compression rate indicates the

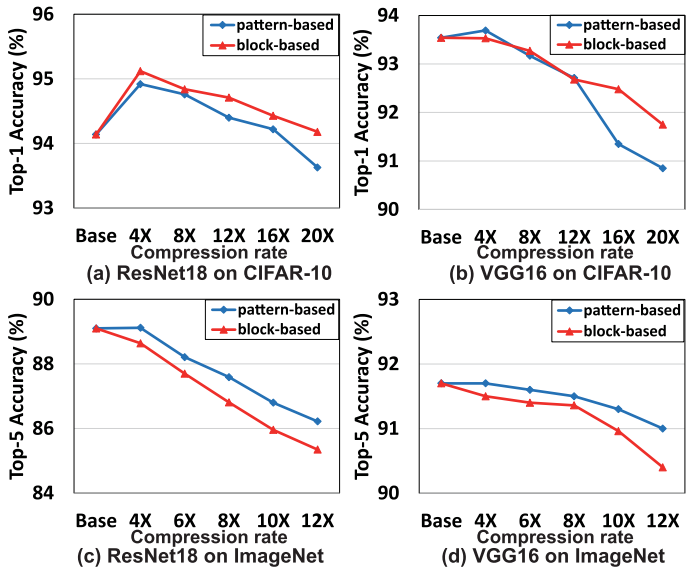

Fig. 7. The top-1/top-5 accuracy comparisons of pattern-based pruning and block-punched pruning (block size of 4 × 16) under the same compression rates for ResNet-18 and VGG-16 on CIFAR-10 and ImageNet datasets.

parameter reduction rate for each 3 × 3 CONV layer. From the figure, we can make the following observations: (i) block-punched pruning consistently shows comparable or higher accuracy for the pruned model under different compression rates on the CIFAR-10 dataset; and (ii) both block-punched and pattern-based pruning achieve accuracy enhancement when the compression rate is relatively low, especially on ResNet-18. The reason is that pruning with a small compression rate can help mitigate the overfitting problem.

The comparison results of pattern-based pruning and block-punched pruning on the ImageNet dataset with different compression rates are shown in Figure 7(c) and (d). Different from the observations on the CIFAR-10 dataset, pattern-based pruning shows better accuracy performance under various compression rate settings for both ResNet-18 and VGG-16.

We attribute the different performance on the two datasets to the following. First, for tasks on easy datasets such as CIFAR-10 that can easily achieve higher than 90% accuracy, the networks are generally overparameterized, and both block-punched and pattern-based pruning schemes can achieve a high compression rate (e.g., >10×) and significant acceleration without hurting the model generalization ability. Thus, the acceleration performance of the two pruning schemes becomes a more essential factor that contributes to the pruning scheme selection. Compared to pattern-based pruning, the block-based/block-punched pruning has a more strict constraint on the weight structure, benefiting hardware parallelism and hence a higher acceleration performance under the same compression rate. Therefore, block-based/block-punched pruning is more favorable for easier datasets. Second, for tasks on harder datasets, the pattern-based pruning scheme is more desirable than block-based/block-punched pruning on 3 × 3 CONV layers. An example of the harder dataset is ImageNet, of which even the unpruned network can only achieve less than 80% top-1 accuracy. Because the patterns used by pattern-based pruning form the shape of a Gaussian filter or Laplacian of Gaussian filter that can enhance the ability for feature extraction (as mentioned in Section 2.1). Therefore, it plays an important role in preserving accuracy under an accelerable compression rate.

Based on the preceding results, we make the following remark.

Remark 1. For 3 × 3 CONV layers, block-punched pruning is more suitable for tasks with easier datasets, whereas pattern-based pruning suits tasks with harder datasets better.

We will provide more discussions and verification of the remark in Section 6.3.

Latency perspective. Latency is the other important aspect in performance evaluation of a pruning scheme. From comprehensive comparative experiments conducted offline, we have observed that under the same compression rate, the latency performance of block-punched pruning is better than pattern-punched pruning when the block size is large but worse when the block size is small. The latency of these two pruning regularities mainly depends on which one can achieve a larger compression rate under the same accuracy. Thus, latency is considered as a secondary criterion for the best-suited pruning scheme mapping in the rule-based method. More discussion will be provided in Section 6.3.

5.2.4 3× 3-DW CONV Layer. The 3 × 3-DW CONV layer is widely used in current DNN designs such as the MobileNet family [63]. It is a special case of the 3 × 3 CONV layer, which applies a 2D depth filter at each depth level of the input tensor. Thus, both pattern-based pruning and block-punched pruning can be applied to 3 × 3-DW layers theoretically. In our rule-based selection policy, we prefer to not prune 3 × 3-DW layers mainly for two reasons: (i) 3 × 3-DW layers are computation- and memory efficient; (ii) 3 × 3-DW layers are sensitive to pruning.

We use MobileNet-V2 on ImageNet as an example; 33% of layers are 3 × 3-DW layers, but they only contribute 6.9% MACs and 1.7% parameters in total. Pruning 3 × 3-DW layers will not achieve a considerable gain even if all of them are pruned. However, the 3 × 3-DW layers contribute 33% of activations, making each weight in the 3 × 3-DW layer more significant. Moreover, in a regular 3 × 3 CONV layer, one input (activation) channel will be filtered by multiple CONV kernels that come from different CONV filters and have different pruned locations, mitigating the damage of pruning on feature extraction. On the contrary, in a 3 × 3-DW layer, one input channel will only be filtered by one CONV kernel, which makes 3 × 3-DW layers more sensitive to the pruning.

We conducted an ablation study about the impact of pruning 3 × 3-DW on accuracy and overall pruning ratio. The results showed that pruning 3 × 3-DW layers will only slightly increase the pruning ratio while leading to a noticeable accuracy loss. Our experiment results shown in Section 6.2 indicate that both pattern-based pruning and block-punched pruning lead to a non-negligible accuracy drop when applied to 3 × 3-DW layers. Therefore, our rule-based method does not map any pruning scheme to the 3 × 3-DW CONV layers.

We summarize the workflow of the training-free rule-based method in Figure 8. For each layer of a given DNN, we first examine the layer type. If the layer is a 3 × 3-DW CONV layer, no pruning scheme is mapped. For 3 × 3 CONV layers, the pruning regularity depends on the size of the dataset. Pattern-based pruning is mapped to 3 × 3 CONV layers if the task has a large dataset; otherwise, block-punched pruning is selected. The proposed general block-based/block-punched pruning is mapped to all other types of layers. When block-based/block-punched pruning is selected, the block size is determined according to an offline-generated latency model with a latency threshold. We note that the entire mapping process, including the pruning regularity mapping and block size selection, is training-free without incurring any additional cost.

6 EVALUATION

6.1 Methodology

Evaluation objective. The first part of our evaluation objective is to show the effectiveness of the general fine-grained structured pruning scheme and the corresponding compiler optimizations.

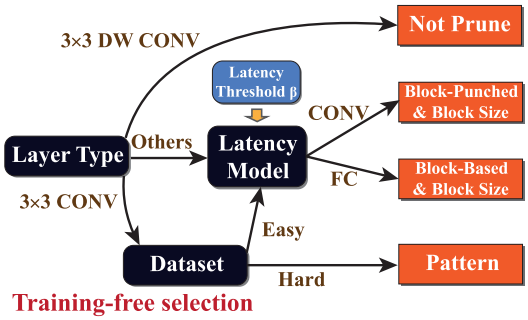

Fig. 8. Rule-based pruning scheme selection.

The second part of our evaluation objective is to compare the overall pruning scheme mapping framework with the state-of-the-art DNN inference acceleration framework PatDNN [59] in terms of accuracy and latency. Note that PatDNN already outperforms other DNN inference frameworks including TVM [11], MNN [2], and TFLite [1], thus the comparison with PatDNN is sufficient to show the effectiveness of our methods.

Our achieved speedup mainly comes from the following. First, our general fine-grained structured pruning is applicable to all types of layers, which better compresses the model size and reduces the computation workload. Second, our compressed sparse matrix storage and associated compiler optimizations improve the computation regularity/parallelism, thus transforming the computation reduction to real performance gains. Third, our automatic pruning scheme mapping methods successfully map the best-suited pruning configurations to each layer, maximizing the compression rate while maintaining accuracy.

DNN models. We evaluate on three mainstream DNNs: VGG-16 [64], ResNet-50 [25], and MobileNet-V2 [63]. They are trained on two representative datasets: CIFAR-10 and ImageNet [17]. We also conduct experiments on YOLOv4 [6] with the Microsoft COCO dateset [42].

Evaluation platforms and running configurations. All evaluated models are trained on a server with eight NVIDIA RTX 2080Ti GPUs. The training codes are implemented with the PyTorch API. The latency is measured on the mobile GPU of an off-the-shelf Samsung Galaxy S10 smartphone, which has the Qualcomm Snapdragon 855 mobile platform with a Qualcomm Kryo 485 Octa-core CPU and a Qualcomm Adreno 640 GPU. Each test takes 50 runs on different inputs with eight threads on the CPU and all pipelines on the GPU. As different runs do not vary greatly, only the average time is reported for readability. All runs are tuned to the best configurations. We empirically choose the latency threshold $\beta = 20\%$.

6.2 Evaluations of the Proposed Pruning Scheme

We first evaluate the inference latency of block-punched pruning using different block sizes on 1×1 and 3×3 CONV layers with different layer sizes, as shown in Figure 9. The input feature map size of the testing CONV layers is set to 56×56, 28×28, 14×14, and 7×7, whereas the input/output channel size is set to 64, 128, 256, and 512. These configurations are commonly used in real DNN networks such as ResNet-50 and VGG-16 on ImageNet. In addition, these configurations keep the MACs the same for all $1 \times 1/3 \times 3$ CONV layers, which can help us observe the impact of different input feature map size and number of channels on latency better.

From Figure 9(a), we can see that the latency is reduced with a larger block size. However, the speedup gradually saturates. The reason is that the remaining weights in each block are more likely to be sufficient to exploit high hardware parallelism with larger block size. Another observation

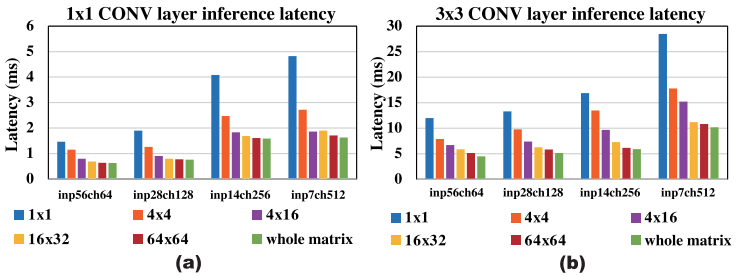

Fig. 9. Latency of 1 × 1 and 3 × 3 CONV layers under different feature sizes and input/output channels.

is that the layer inference latency increases for all block sizes as the size of the input feature map decreases and the number of input/output channels increases. The reason is that a smaller input feature map size lowers the reuse rate of each weight, causing hardware parallelism degradation. Similar observations can also be found in Figure 9(b), which shows the latency results for different 3 × 3 CONV layers.

Similar results can also be observed on FC layers with block-based pruning. Figure 10(a) shows the latency comparisons on two FC layers. The size of the FC layer on the left-hand side is used as the first FC layer in VGG-16, whereas the right-hand side is the representative FC layer in BERT. The latency of each FC layer is normalized to its own 1 × 1 block size result. We can observe that for large FC layers, increasing the block size can reduce latency effectively, whereas the latency reduction achieved by increasing the block size gets saturated gradually in relatively small FC layers.

6.3 Automatic Pruning Scheme Mapping Methods Evaluations

6.3.1 Accuracy Analysis on Pattern-Based Pruning and Block-Punched Pruning. From the results on ResNet-18 and VGG-16 with CIFAR-10 and ImageNet datasets, we make Remark 1. We further examine the remark on YOLOv4 with the Microsoft COCO dataset, which can be reasonably regarded as difficult task, as shown in Table 2. The compression rate refers to the compression rate of the entire model, and the block size is 4 × 16. When only 3 × 3 CONV layers are pruned, pattern-based pruning achieves a higher **mean average precision (mAP)**, which matches the remark that pattern-based pruning suits tasks with larger datasets better on 3 × 3 CONV layers. However, current pattern-based pruning is only applicable to 3 × 3 layers, limiting the compression performance. With the proposed general pruning scheme applicable to different layers, we achieve an 8.1× compression rate with 51.3 mAP and 11.5 **frames per second (FPS)**. A hybrid pruning scheme by mapping pattern-based pruning to 3 × 3 CONV layers and block-based/block-punched pruning to all of the other layers can further achieve an 8.5× compression rate with 51.7 mAP and 12.3 FPS. We also show the results of unstructured pruning and structured pruning, which achieve 52.5 mAP and 39.4 mAP, and 7.6 FPS and 11.8 FPS, respectively. It can be observed that our hybrid scheme method is 1.62× faster than unstructured pruning while maintaining comparable accuracy. When compared to structured pruning, our hybrid scheme method achieves much higher accuracy and is also slightly faster than structured pruning at the same time. This further strengthens the advantage of our proposed method.

6.3.2 Latency Analysis on Pattern-Based Pruning and Block-Punched Pruning. We conduct comprehensive comparative experiments offline to analyze the latency performance of pattern-based pruning and block-punched pruning to determine the best-suited pruning scheme for 3 × 3 CONV layers. Figure 10(b) shows an example of the latency comparisons for a 3 × 3 CONV layer with

Table 2. Comparison on YOLOv4

Pruning Scheme	# Weights	Compression Rate	mAP	FPS
Not prune	64.36M	1×	57.3	3.5
Structured	8.82M	7.3×	39.4	11.8
Unstructured	5.75M	11.2×	52.5	7.6
Pattern	10.22M	*6.3×	52.8	9.7
Block	10.38M	*6.2×	52.4	9.1
Block	7.94M	8.1×	51.3	11.5
Hybrid	7.57M	8.5×	51.7	12.3

*Overall compression rate, but only 3 × 3 CONV layers are pruned.

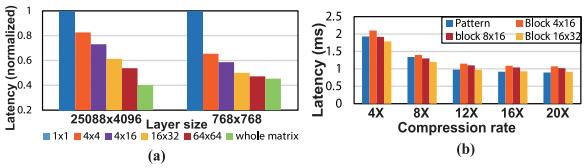

Fig. 10. (a) Latency of two example FC layers. (b) Example of latency comparisons of the 3 × 3 CONV layer using pattern-based pruning and block-punched pruning.

Table 3. Accuracy Comparison (Δ acc.) of Applying Pattern-Based Pruning and Block-Punched Pruning to the Depth-Wise 3 × 3 CONV Layers in MobileNetV2

	CIFAR-10	CIFAR-100
Compression rate	7.19 × ->8.12×	2.78 × ->2.91×
Pattern Based	−0.4	−0.9
Block Based	−1.01	−1.51

28 × 28 input feature map size and 128 input/output channels under different compression rates. Under 4× and 8× compression, pattern-based pruning has similar latency performance to block-punched pruning with a block size of 8 × 16. When the compression rate is higher than 12×, pattern-based pruning has speed that is similar to block-punched pruning with a block size of 16 × 32. However, the latency difference between pattern-based pruning and block-punched pruning is minor, as we discussed in Section 5.2.3, thus we consider latency performance as a secondary criterion in the rule-based pruning scheme mapping method.

6.3.3 Ablation Study on the 3 × 3-DW CONV Layer. As mentioned in Section 5.2.4, 3 × 3-DW CONV layers usually only account for a small portion of weights and computations, and they play an important role in capturing spatial correlations in DNNs [13], thus we propose not to prune 3 × 3-DW CONV layers. Table 3 shows the accuracy results of applying pattern-based pruning and block-punched pruning to 3 × 3-DW CONV layers in MobileNetV2. Here we use the baseline models in which all 1 × 1 CONV layers are pruned by block-punched pruning with compression rates of 7.19× and 2.78× for CIFAR-10 and CIFAR-100, respectively. Then, on top of the pruned model, we apply an extra 2.22× pattern-based/block-punched pruning only for the 3 × 3-DW CONV layers and compare the final accuracy. The results show that the overall compression rate only increases slightly, but there is a non-negligible accuracy drop for pattern-based pruning and block-based pruning. Thus, our rule-based pruning scheme mapping method will not map any pruning scheme for 3 × 3-DW CONV layers.

6.3.4 Evaluations of Automatic Pruning Scheme Mapping Methods. We compare the search-based and rule-based methods with the state-of-the-art end-to-end inference framework PatDNN [59], which uses pattern-based pruning with the ADMM pruning algorithm. The comparison results are shown in Table 4. Here, the compression rate refers to the parameter reduction rate of the CONV layers. The accuracy for the ImageNet dataset indicates the top-5 accuracy.

Table 4. Comparison with PatDNN on the CIFAR-10 Dataset (Top-1 Accuracy) and the ImageNet Dataset (Top-1/Top-5 Accuracy)

Network	Method	Pruning Scheme	Pruned Layers	Original Acc. (%)	Compression Rate	Acc. Drop (%)	Latency (ms)	MACs
CIFAR-10								
ResNet-50	PatDNN	Pattern	3 × 3 CONV	95.6	1.57×	−1.0	10.44	1.9G
	Rule based	Block	3 × 3 CONV, 1 × 1 CONV	95.6	11.51 ×	0.1	4.25	0.6G
	Search based	Hybrid	3 × 3 CONV, 1 × 1 CONV	95.6	11.88×	0.1	4.20	0.6G
VGG-16	PatDNN	Pattern	3 × 3 CONV	93.9	8.0×	−0.4	2.59	73M
	Rule based	Block	3 × 3 CONV	93.9	12.38 ×	−0.3	2.02	59M
	Search based	Hybrid	3 × 3 CONV	93.9	12.50×	−0.3	2.00	58M
MobileNetV2	PatDNN	Pattern	3 × 3 DW CONV	94.6	1.01×	−0.1	3.63	296M
	Rule based	Block	1 × 1 CONV	94.6	7.53 ×	0.2	1.86	89M
	Search based	Block	1 × 1 CONV	94.6	7.54×	0.1	1.86	89M
ImageNet								
ResNet-50	PatDNN	Pattern	3 × 3 CONV	76.1/92.8	1.56×	−/−0.2	29.89	3.0G
	Rule based	Hybrid	3 × 3 CONV, 1 × 1 CONV	76.1/92.8	4.37 ×	0.3/0.1	17.26	1.6G
	Search based	Hybrid	3 × 3 CONV, 1 × 1 CONV	76.1/92.8	4.41×	0.1/0	17.22	1.6G
VGG-16	PatDNN	Pattern	3 × 3 CONV	74.5/91.7	8.0×	−/0.1	18.91	3.8G
	Rule based	Pattern	3 × 3 CONV	74.5/91.7	8.22 ×	0.2/0.1	18.17	3.5G
	Search based	Pattern	3 × 3 CONV	74.5/91.7	8.22×	0.2/0.1	18.17	3.5G
MobileNetV2	PatDNN	Pattern	3 × 3 DW CONV	71.0/90.3	1.01×	−/0	4.90	300M
	Rule based	Block	1 × 1 CONV	71.0/90.3	1.76 ×	0.5/0.3	3.98	177M
	Search based	Block	1 × 1 CONV	71.0/90.3	1.82×	0.5/0.3	3.90	165M

The configurations of the search-based method are obtained using five GPU servers, and take 3 and 9 days for CIFAR-10 and ImageNet models, respectively, which is acceptable for RL-based search methods [67, 88]. We use a search-based method to provide a close-to-optimal result, which indicates the performance upper bound. Accelerating the search process is not the main concern of our work, and our search process can be accelerated by adopting fast evaluation techniques such as Bayesian optimization [12, 32].

For ResNet-50 on CIFAR-10, the rule-based method can achieve an 11.51× compression rate with only 0.1% accuracy drop, which is significantly higher than the results obtained by PatDNN. The reason for the limited performance of PatDNN is that only 44.3% of the parameters of ResNet-50 are in the 3 × 3 CONV layers that can be pruned with pattern-based pruning, as shown in Figure 3. Our rule-based method, however, maps the flexible block-punched pruning that can be applied to CONV layers with different kernel sizes, thus achieving a much higher compression rate. The search-based method reaches a slightly higher compression rate and minor latency reduction compared with the rule-based method.

With the automatic mapping of block-punched pruning and block size provided by the rule-based method and compression rate derived by the reweighted pruning algorithm, we reach a 12.38× compression rate with 0.3% accuracy improvement on VGG-16 for the CIFAR-10 dataset. Still, the search-based method renders slightly better performance than the rule-based method.

For MobileNet-V2, mapping block-based pruning with an optimized block size on 1 × 1 CONV layers by the rule-based method achieves a 7.53× compression rate with only 0.2% accuracy drop. The compression rate is much higher than PatDNN, as pattern-based pruning cannot be applied to 1 × 1 CONV, and 3 × 3-DW CONV layers only account for 1.9% of the parameters in the model. The performance difference between the rule-based method and the search-based method is negligible.

Different from CIFAR-10, pattern-based pruning has better accuracy performance on tasks with large datasets like ImageNet, as discussed in Remark 1. Hence, the rule-based method maps pattern-based pruning to 3 × 3 CONV layers and block-punched pruning with optimized block size to the remaining layers. For ResNet-50 on ImageNet, the rule-based method can reach a 4.37× compression rate with only 0.1% accuracy loss, and 1.73× speedup on mobile GPU over PatDNN.

Table 5. Comparisons with Models Obtained by Various Model Compression Techniques on ImageNet

Group	Model	MACs	Top-1 Acc.
300M MACs	MobileNetV2 1.0×	300M	71.0%
	NetAdapt-MobileNetV1 [72]	284.3M	69.1%
	ChamNet-B [14]	323M	73.8%
200M MACs	MobileNetV2 0.75×	209M	69.8%
	AMC-MobileNetV2 [26]	211M	70.8%
	AutoSlim-MobileNetV2 [44]	207M	73%
	MetaPruning-MobileNetV2 [47]	217M	71.2%
	Ours (rule based)	203M	70.8%
150M MACs	MobileNetV1 0.5×	150M	63.3%
	AutoSlim-MobileNetV1 [44]	150M	67.9%
	Ours (rule based)	177M	70.5%
	Ours (rule based)	151M	69.8%

For VGG-16 on ImageNet, both the rule-based method and the search-based method map pattern-based pruning to all 3×3 layers with the reweighted dynamic regularization algorithm, and achieves a 8.22× compression rate with only 0.1% accuracy loss, which outperforms PatDNN. As all methods adopt pattern-based pruning, the performance difference between our methods and PatDNN is attributed to the pruning algorithm. With the reweighted pruning aglorithm, our method has the advantage of determining the compression rate for each layer automatically, whereas PatDNN is based on ADMM and requires manual setting of the compression rate for each layer. For MobileNet-V2 on the ImageNet dataset, both the rule-based method and the search-based method map block-punched pruning to 1×1 CONV layers, and reach a 1.76× compression rate and a 1.82× compression rate, respectively.

We also compare our method with other representative model compression techniques including NetAdapt [72], ChamNet [14], AMC [26], AutoSlim [44], and MetaPruning [47] on the ImageNet dataset, and the results are shown in Table 5. At the 200M MAC level, our rule-based method achieves the same accuracy as AMC with fewer MACs. Our method also outperforms the 0.75× channel scaled MobileNetV2 in both accuracy and MACs. At the 150M MAC level, the model obtained by our rule-based model achieves the highest top-1 accuracy with similar MACs compared with AutoSlim and the 0.5× channel scaled MobileNetV1.

Combining all of the results, we can see that both the rule-based and the search-based method significantly outperform PatDNN. The rule-based method can provide pruned models with similar accuracy and latency performance as the search-based method, and avoids the policy training process, thus it is more useful in practice. Moreover, with the assist of our compiler optimization, both methods can easily achieve real-time DNN inference (less than 33 ms) on all models mentioned previously.

6.3.5 Portability Evaluation on Different Platforms. We further evaluate the portability of our proposed rule-based pruning scheme mapping method on different mobile platforms. Three tested platforms are Samsung Galaxy S10, S20, and S21, respectively. They are equipped with different types of chipsets and mobile GPUs. The detailed hardware specifications are shown in Table 6. Table 7 shows the portability evaluation results on the three platforms using our rule-based pruning scheme mapping method. We use the VGG-16 network and test on CIFAR-10 and ImageNet datasets, respectively. We build a latency model for each platform and use the same

Table 6. Hardware Specifications of Platforms for Portability Evaluation

Model	Chipset	GPU	RAM
Samsung Galaxy S10	Qualcomm Snapdragon 855	Adreno 640	8 GB
Samsung Galaxy S20	Qualcomm Snapdragon 865	Adreno 650	12 GB
Samsung Galaxy S21	Qualcomm Snapdragon 888	Adreno 666	8 GB

Table 7. Portability Evaluation on Different Platforms Using the Rule-Based Method on VGG-16

Dataset	Platform	Compression Rate	MACs	Top-1 Acc.	Latency (ms)
CIFAR-10	Galaxy S10	12.38×	59M	94.2%	2.02
	Galaxy S20	12.06×	62M	94.1%	1.85
	Galaxy S21	12.12×	61M	94.2%	1.65
ImageNet	Galaxy S10	8.22×	3.5G	74.3%	18.17
	Galaxy S20	8.12×	3.4G	74.5%	16.23
	Galaxy S21	8.15×	3.4G	74.5%	15.12

latency threshold of $\beta = 20\%$. It can be observed that our rule-based method can consistently achieve high model accuracy and leverages better hardware for a higher inference speed, which illustrates the stability of our reweighted pruning algorithm and the effectiveness and portability of our rule-based method.

7 CONCLUSION

We propose a general pruning scheme with fine-grained structured pruning regularity and a reweighted dynamic pruning algorithm. Compiler optimizations are introduced to extract the structure information and exploit hardware parallelism. We further probe into the new problem of mapping the best-suited pruning scheme for each layer of any given DNN and propose two automatic pruning scheme mapping methods. Experimental results demonstrate the effectiveness of the proposed pruning scheme and pruning scheme mapping methods.

APPENDIX

A COMPILER OPTIMIZATION DETAILS

We provide more details of our compiler optimizations in this section. Different from prior DNN inference acceleration frameworks [1–3, 11, 50, 59] that only support dense models or pattern-based pruned models, our compiler optimizations are general, and support both dense (unpruned) models and sparse (pruned) models with different pruning schemes for fast inference on various mobile platforms. Besides the BCS and the row reordering optimization mentioned in the main article (Section 4.3), our compiler-based optimization techniques also include (i) a layer fusion mechanism to fuse different layers together for the reduction of memory consumption of intermediate results and number of operators; (ii) an auto-tuning process to determine the best-suited configurations of parameters for different mobile CPUs/GPUs; and (iii) DSL-based code generation.

A.1 Layer Fusion Mechanism

To effectively reduce the model inference latency, a layer fusion technique is incorporated in our compiler optimization to fuse the computation operators in the computation graph. With layer fusion, both the memory consumption of the intermediate results and the number of operators

can be reduced. The fusion candidates in a model are identified based on two kinds of polynomial calculation properties: compression laws and data access patterns. The compression laws include associative property, communicative property, and distributive property.

However, looking for the fusion candidates in such a large space of all combinations of computation operations is too expensive. Therefore, we introduce two constraints to guide the look-up process: (i) only explore the opportunities that are specifically provided due to the preceding properties, and (ii) only consider enlarging the overall computation for CPU/GPU utilization improvement and reducing the memory access for memory performance improvement as the cost metrics in the fusion. Compared with prior works on loop fusion [4, 5, 7], our method is more aggressive without high exploration cost.

A.2 Auto-Tuning for Different Mobile CPUs/GPUs

During DNN execution, there are many tuning parameters, such as matrix tiling sizes, loop unrolling factors, and data placement on GPU memory, that influence the performance. It is hard to determine the best-suited configuration of these parameters manually. To alleviate this problem, our compiler incorporates an auto-tuning approach for both sparse (pruned) models and dense (unpruned) models. The genetic algorithm is leveraged to explore the best-suited configurations automatically. It starts parameter search after an initialization with an arbitrary number of chromosomes and explores the parallelism better. Acceleration codes for different DNN models and different mobile CPUs/GPUs can be generated efficiently and quickly through this auto-tuning process, providing the foundation for fast end-to-end inference. The auto-tuning optimizations, together with the layer fusion and sparse model optimizations, make our framework outperform other acceleration frameworks.

A.3 DSL-Based Code Generation

In DL, a computational graph of a DNN model can be represented by a directed acyclic graph (DAG). Each node in this graph corresponds to an operator. We propose a high-level DSL to specify such kind of operators. Each operator in a computational graph also with a layer-wise intermediate representation (IR) that contains BCS pruning information. The input and output are different tensors in terms of different shapes. This DSL also provides a `Tensor` function for users to create matrices (or tensors).

In this way, DSL is equivalent to a computational graph (i.e., DSL is another type of high-level functions used to simulate the dataflow of the DNN model), and they can be easily converted to each other. DSL provides users with the flexibility to use existing DNNs or create new DNNs, improving the productivity of DNN programming. If the DNN already exists, we will convert it into an optimized calculation graph and convert this graph into a DSL. Otherwise, the user writes the model code in the DSL, converts it back to a calculation graph, performs advanced optimization, and regenerates the optimized DSL code.

Finally, our compiler translates the DSL into low-level C++ code for mobile CPU and OpenCL code for mobile GPU, and optimizes the low-level code through a set of optimizations enabled by BCS pruning. The generated code can be then deployed on the mobile device.

REFERENCES

[1] TensorFlow. n.d. TensorFlow Lite. Retrieved March 2, 2022 from https://github.com/tensorflow/tflite-support.
[2] GitHub. n.d. alibaba/MNN. Retrieved March 2, 2022 from https://github.com/alibaba/MNN.
[3] PyTorch. n.d. PyTorch Mobile. Retrieved March 2, 2022 from https://pytorch.org/mobile/home.
[4] Arash Ashari, Shirish Tatikonda, Matthias Boehm, Berthold Reinwald, Keith Campbell, John Keenleyside, and P. Sadayappan. 2015. On optimizing machine learning workloads via kernel fusion. *ACM SIGPLAN Notices* 50, 8 (2015), 173–182.

[5] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM Review* 59, 1 (2017), 65–98.
[6] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934* (2020).
[7] Matthias Boehm, Berthold Reinwald, Dylan Hutchison, Alexandre V. Evfimievski, and Prithviraj Sen. 2018. On optimizing operator fusion plans for large-scale machine learning in SystemML. *arXiv preprint arXiv:1801.00829* (2018).
[8] Han Cai, Ligeng Zhu, and Song Han. 2018. ProxylessNAS: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332* (2018).
[9] Yuxuan Cai, Hongjia Li, Geng Yuan, Wei Niu, Yanyu Li, Xulong Tang, Bin Ren, and Yanzhi Wang. 2021. YOLObile: Real-time object detection on mobile devices via compression-compilation co-design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 955–963.
[10] Emmanuel J. Candes, Michael B. Wakin, and Stephen P. Boyd. 2008. Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier Analysis and Applications* 14, 5–6 (2008), 877–905.
[11] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, et al. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*.
[12] Yutian Chen, Aja Huang, Ziyu Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver, and Nando de Freitas. 2018. Bayesian optimization in Alphago. *arXiv preprint arXiv:1812.06855* (2018).
[13] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*.
[14] Xiaoliang Dai, Yangqing Jia, Peter Vajda, Matt Uyttendaele, Niraj K. Jha, Peizhao Zhang, Bichen Wu, et al. 2019. ChamNet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'19)*. https://doi.org/10.1109/cvpr.2019.01166
[15] Xiaoliang Dai, Hongxu Yin, and Niraj K. Jha. 2019. Grow and prune compact, fast, and accurate LSTMs. *IEEE Transactions on Computers* 69, 3 (2019), 441–452.
[16] Xiaoliang Dai, Hongxu Yin, and Niraj K. Jha. 2019. NeST: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers* 68, 10 (2019), 1487–1497.
[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*. IEEE, Los Alamitos, CA, 248–255.
[18] Peiyan Dong, Siyue Wang, Wei Niu, Chengming Zhang, Sheng Lin, Zhengang Li, Yifan Gong, Bin Ren, Xue Lin, and Dingwen Tao. 2020. RTMobile: Beyond real-time mobile acceleration of RNNs for speech recognition. In *Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC'20)*. IEEE, Los Alamitos, CA, 1–6.
[19] Xuanyi Dong and Yi Yang. 2019. Network pruning via transformable architecture search. In *Proceedings of the 2019 Conference on Neural Information Processing Systems (NeurIPS'19)*. 759–770.
[20] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*.
[21] Yifan Gong, Zheng Zhan, Zhengang Li, Wei Niu, Xiaolong Ma, Wenhao Wang, Bin Ren, et al. 2020. A privacy-preserving-oriented DNN pruning and mobile acceleration framework. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*. 119–124.
[22] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient DNNs. In *Proceedings of the 2016 Conference on Neural Information Processing Systems (NeurIPS'16)*.
[23] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Proceedings of the 2015 Conference on Neural Information Processing Systems (NeurIPS'15)*.
[24] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. MCDNN: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'16)*. ACM, New York, NY, 123–136.
[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*.
[26] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*.
[27] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'19)*.
[28] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'17)*.

[29] Gopalakrishna Hegde, Siddhartha, Nachiappan Ramasamy, and Nachiket Kapre. 2016. CaffePresso: An optimized library for deep learning on embedded accelerator-based platforms. In *Proceedings of the 2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES'16)*. IEEE, Los Alamitos, CA, 1–10.

[30] Loc N. Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. DeepMon: Mobile GPU-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'17)*. ACM, New York, NY, 82–95.

[31] Tong Jian, Yifan Gong, Zheng Zhan, Runbin Shi, Nasim Soltani, Zifeng Wang, Jennifer G. Dy, Kaushik Roy Chowdhury, Yanzhi Wang, and Stratis Ioannidis. 2022. Radio frequency fingerprinting on the edge. *IEEE Transactions on Mobile Computing*. Early access, March 8, 2022.

[32] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. 2017. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. 528–536.

[33] Nicholas D. Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. DeepX: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE, Los Alamitos, CA, 23.

[34] Nicholas D. Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. 2017. Squeezing deep learning into mobile and embedded devices. *IEEE Pervasive Computing* 16, 3 (2017), 82–88.

[35] Nicholas D. Lane, Petko Georgiev, and Lorena Qendro. 2015. DeepEar: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, New York, NY, 283–294.

[36] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient ConvNets. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

[37] Hongjia Li, Geng Yuan, Wei Niu, Yuxuan Cai, Mengshu Sun, Zhengang Li, Bin Ren, Xue Lin, and Yanzhi Wang. 2021. Real-time mobile acceleration of DNNs: From computer vision to medical applications. In *Proceedings of the 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC'21)*. IEEE, Los Alamitos, CA, 581–586.

[38] Ke Li and Jitendra Malik. 2016. Learning to optimize. *arXiv preprint arXiv:1606.01885* (2016).

[39] Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. 2019. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'19)*.

[40] Yuhang Li, Xin Dong, and Wei Wang. 2020. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*.

[41] Zhengang Li, Yifan Gong, Xiaolong Ma, Sijia Liu, Mengshu Sun, Zheng Zhan, Zhenglun Kong, Geng Yuan, and Yanzhi Wang. 2020. SS-Auto: A single-shot, automatic structured weight pruning framework of DNNs with ultra-high efficiency. *arXiv preprint arXiv:2001.08839* (2020).

[42] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision*. 740–755.

[43] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. 2019. AutoCompress: An automatic DNN structured pruning framework for ultra-high compression rates. *arXiv preprint arXiv:1907.03141* (2019).

[44] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. 2019. AutoSlim: An automatic DNN structured pruning framework for ultra-high compression rates. *arXiv preprint arXiv:1907.03141* (2019).

[45] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. 2020. AutoCompress: An automatic DNN structured pruning framework for ultra-high compression rates. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20)*. 4876–4883.

[46] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'17)*.

[47] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. 2019. MetaPruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV'19)*. https://doi.org/10.1109/iccv.2019.00339

[48] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*.

[49] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. ThiNet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'17)*.

[50] Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. 2020. PCONV: The missing but desirable sparsity in DNN weight pruning for real-time execution on mobile devices. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20)*.

[51] Xiaolong Ma, Zhengang Li, Yifan Gong, Tianyun Zhang, Wei Niu, Zheng Zhan, Pu Zhao, et al. 2020. BLK-REW: A unified block-based DNN pruning framework using reweighted regularization method. *arXiv preprint arXiv:2001.08357* (2020).

[52] Xiaolong Ma, Sheng Lin, Shaokai Ye, Zhezhi He, Linfeng Zhang, Geng Yuan, Sia Huat Tan, et al. 2019. Non-structured DNN weight pruning—Is it beneficial in any platform? *arXiv:1907.02124 [cs.LG]* (2019).

[53] Xiaolong Ma, Wei Niu, Tianyun Zhang, Sijia Liu, Sheng Lin, Hongjia Li, Wujie Wen, et al. 2020. An image enhancing pattern-based sparsity for real-time inference on mobile devices. In *Proceedings of the European Conference on Computer Vision*. 629–645.

[54] Xiaolong Ma, Geng Yuan, Sheng Lin, Caiwen Ding, Fuxun Yu, Tao Liu, Wujie Wen, Xiang Chen, and Yanzhi Wang. 2020. Tiny but accurate: A pruned, quantized and optimized memristor crossbar framework for ultra efficient DNN implementation. In *Proceedings of the 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC'20)*. IEEE, Los Alamitos, CA, 301–306.

[55] Xiaolong Ma, Geng Yuan, Sheng Lin, Caiwen Ding, Fuxun Yu, Tao Liu, Wujie Wen, Xiang Chen, and Yanzhi Wang. 2020. Tiny but accurate: A pruned, quantized and optimized memristor crossbar framework for ultra efficient DNN implementation. In *Proceedings of the 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC'20)*. IEEE, Los Alamitos, CA, 301–306.

[56] Xiaolong Ma, Geng Yuan, Sheng Lin, Zhengang Li, Hao Sun, and Yanzhi Wang. 2019. ResNet can be pruned 60×: Introducing network purification and unused path removal (P-RM) after weight pruning. In *Proceedings of the 2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH'19)*. IEEE, Los Alamitos, CA, 1–2.

[57] Chuhan Min, Aosen Wang, Yiran Chen, Wenyao Xu, and Xin Chen. 2018. 2PFPCE: Two-phase filter pruning based on conditional entropy. *arXiv preprint arXiv:1809.02220* (2018).

[58] Wei Niu, Zhenglun Kong, Geng Yuan, Weiwen Jiang, Jiexiong Guan, Caiwen Ding, Pu Zhao, Sijia Liu, Bin Ren, and Yanzhi Wang. 2020. Achieving real-time execution of transformer-based large-scale models on mobile with compiler-aware neural architecture optimization. *arXiv preprint arXiv:2009.06823* (2020).

[59] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*.

[60] Kaoru Ota, Minh Son Dao, Vasileios Mezaris, and Francesco G. B. De Natale. 2017. Deep learning for mobile multimedia: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications* 13, 3s (2017), 1–22.

[61] Ao Ren, Tianyun Zhang, Shaokai Ye, Jiayu Li, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. 2019. ADMM-NN: An algorithm-hardware co-design framework of DNNs using alternating direction methods of multipliers. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*.

[62] Ao Ren, Tianyun Zhang, Shaokai Ye, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. 2019. ADMM-NN: An algorithm-hardware co-design framework of DNNs using alternating direction methods of multipliers. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*.

[63] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18)*.

[64] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556* (2014).

[65] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

[66] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS'99)*. 1057–1063.

[67] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'19)*. 2820–2828.

[68] Yanzhi Wang, Shaokai Ye, Zhezhi He, Xiaolong Ma, Linfeng Zhang, Sheng Lin, Geng Yuan, et al. 2019. Non-structured DNN weight pruning considered harmful. *arXiv:1907.02124* (2019).

[69] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Proceedings of the 2016 Conference on Neural Information Processing Systems (NeurIPS'16)*.

[70] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, et al. 2019. FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'19)*. 10734–10742.

[71] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. 2018. DeepCache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, New York, NY, 129–144.

[72] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. NetAdapt: Platform-Aware neural network adaptation for mobile applications. In Computer Vision—ECCV 2018. Lecture Notes in Computer Science, Vol. 11214. Springer, 289–304. https://doi.org/10.1007/978-3-030-01249-6_18

[73] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. 2017. DeepSense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*. 351–360.

[74] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. 2018. NISP: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18)*.

[75] Geng Yuan, Payman Behnam, Zhengang Li, Ali Shafiee, Sheng Lin, Xiaolong Ma, Hang Liu, et al. 2021. FORMS: Fine-grained polarized ReRAM-based in-situ computation for mixed-signal DNN accelerator. In *Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA'21)*. https://doi.org/10.1109/isca52012.2021.00029

[76] Geng Yuan, Xiaolong Ma, Caiwen Ding, Sheng Lin, Tianyun Zhang, Zeinab S. Jalali, Yilong Zhao, Li Jiang, Sucheta Soundarajan, and Yanzhi Wang. 2019. An ultra-efficient memristor-based DNN framework with structured weight pruning and quantization using ADMM. In *Proceedings of the 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED'19)*. IEEE, Los Alamitos, CA, 1–6.

[77] Geng Yuan, Xiaolong Ma, Sheng Lin, Zhengang Li, and Caiwen Ding. 2019. A SOT-MRAM-based processing-in-memory engine for highly compressed DNN implementation. *arXiv preprint arXiv:1912.05416* (2019).

[78] Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, et al. 2021. MEST: Accurate and fast memory-economic sparse training framework on the edge. *arXiv:2110.14032* [cs.LG] (2021).

[79] Zheng Zhan, Yifan Gong, Pu Zhao, Geng Yuan, Wei Niu, Yushu Wu, Tianyun Zhang, et al. 2021. Achieving on-mobile real-time super-resolution with neural architecture and pruning search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4821–4831.

[80] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. 2019. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2224–2287.

[81] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A systematic DNN weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*.

[82] Tianyun Zhang, Kaiqi Zhang, Shaokai Ye, Jiayu Li, Jian Tang, Wujie Wen, Xue Lin, Makan Fardad, and Yanzhi Wang. 2018. Adam-ADMM: A unified, systematic framework of structured weight pruning for DNNs. *arXiv:1807.11091* (2018).

[83] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. 2019. Variational convolutional neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'19)*.

[84] Pu Zhao, Wei Niu, Geng Yuan, Yuxuan Cai, Hsin-Hsuan Sung, Wujie Wen, Sijia Liu, et al. 2020. Achieving real-time LiDAR 3D object detection on a mobile device. *arXiv preprint arXiv:2012.13801* (2020).

[85] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. 2018. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2423–2432.

[86] Xiaotian Zhu, Wengang Zhou, and Houqiang Li. 2018. Improving deep neural network sparsity through decorrelation regularization. In *Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI'18)*.

[87] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. 2018. Discrimination-aware channel pruning for deep neural networks. In *Proceedings of the 2018 Conference on Neural Information Processing Systems (NeurIPS'18)*.

[88] Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

Received June 2021; revised October 2021; accepted November 2021