1xN Pattern for Pruning Convolutional Neural Networks

Mingbao Lin, Yuxin Zhang, Yuchao Li, Bohong Chen, Fei Chao, Member, IEEE, Mengdi Wang, Shen Li, Yonghong Tian, Fellow, IEEE, and Rongrong Ji, Senior Member, IEEE

Abstract—Though network pruning receives popularity in reducing the complexity of convolutional neural networks (CNNs), it remains an open issue to concurrently maintain model accuracy as well as achieve significant speedups on general CPUs. In this paper, we propose a novel 1×N pruning pattern to break this limitation. In particular, consecutive N output kernels with the same input channel index are grouped into one block, which serves as a basic pruning granularity of our pruning pattern. Our 1×N pattern prunes these blocks considered unimportant. We also provide a workflow of filter rearrangement that first rearranges the weight matrix in the output channel dimension to derive more influential blocks for accuracy improvements and then applies similar rearrangement to the next-layer weights in the input channel dimension to ensure correct convolutional operations. Moreover, the output computation after our 1×N pruning can be realized via a parallelized block-wise vectorized operation, leading to significant speedups on general CPUs. The efficacy of our pruning pattern is proved with experiments on ILSVRC-2012. For example, given the pruning rate of 50% and N=4, our pattern obtains about 3.0% improvements over filter pruning in the top-1 accuracy of MobileNet-V2. Meanwhile, it obtains 56.04ms inference savings on Cortex-A7 CPU over weight pruning. Our project is made available at https://github.com/lmbxmu/1xN.

Index Terms—Network pruning, pruning pattern, CPUs acceleration, CNNs

1 INTRODUCTION

Convolutional neural networks (CNNs) have substantially advanced varieties of computer vision tasks [1], [2], [3]. Despite these tremendous success, newly developed networks tend to have more learnable parameters which also mean more floating-point operations (FLOPs). As a result, these CNNs can be rarely run on the general CPUs embedded devices with limited computation power [4]. By pruning the redundancy in CNNs, the emerging network pruning has become a broad consensus in favour of model deployment by both the academia and industries.

As illustrated in Fig. 1, according to the basic pruning granularity, existing works accomplishing network pruning are categorized into weight pruning and filter pruning. The basic granularity of weight pruning falls into individual weights at a fine-grained level and is demonstrated to achieve an extremely high compression rate and high accuracy performance [4], [5], [6]. However, weight pruning receives very limited speed gains since its irregular sparsity barely takes advantage of vector processing architectures such as Single Instruction Multiple Data (SIMD), and poorly utilizes memory buses. In contrast, this increases latency due to the dependent sequences of reads [7]. Recent studies [7], [8], [9], [10] advocate N:M weight pruning where N out of M weights are zeros for every continuous M weights. Currently, this pattern achieves acceleration only in the case of 24. Besides, the acceleration is realized on the specially designed sparse Matrix Multiply-Accumulate (MMA) instructions of NVIDIA A100 towards modern single- and multi-GPU workstations, servers, clusters, and even supercomputers [7], making it impossible to be utilized on other types of GPUs, let alone the CPUs-based platforms.

Different from weight pruning, the basic granularity of filter pruning in Fig. 1 consists of the whole filters. It reduces network complexity at a coarse-grained level by removing all weights in a filter. Consequently, the network structure does not change, thus the sparsified network can be well deployed by both the academia and industries.
fitted by regular hardware and off-the-shelf basic linear algebra subprograms (BLAS) library to obtain acceleration. Nevertheless, filter pruning only maintains accuracy under moderate sparsity rates. Otherwise, such methods suffer more significant performance degradation than weight pruning methods. For example, a recent study [11] shows that a 5.96x parameter reduction of ResNet-50 can well retain the accuracy performance of the original network by weight pruning, however, it is only a 1x reduction in filter pruning. Though the research community has developed various techniques [12], [13], [14], [15], recent works [16], [17] demonstrate that the capacity of these techniques for performance improvement is indeed limited if appropriate training settings are given to the previous works.

Above all, how to simultaneously retain the performance and achieve apparent acceleration on mobile and embedded devices becomes a challenging but valuable problem. In this paper, we propose a novel pattern of 1xN pruning with its merits in realizing both high-performing accuracy and apparent CPUs acceleration for practical model deployment. Our 1xN pattern provides an intermediate granular level for network pruning, which is coarser as compared to the fine-grained weight but finer as compared to the coarse-grained filter. An example of our pruning pattern that satisfies N=4 requirement is shown in Fig. 1: the core distinction of our pruning from existing scenarios [12], [13], [15] lies in that our basic pruning granularity consists of consecutive N output kernels with the same input channel index. In short, we aim to remove these consecutive kernels with smaller $\ell_1$ norms that are considered less important in literature [12]. Our 1xN pruning in this paper follows the typical three-step pipeline of network training, pruning consecutive kernels with smaller $\ell_1$ norms, and fine-tuning the sparsified one to recover the performance. At the point of the second step, we further propose a workflow of filter rearrangement, which rearranges the weight matrix in the output channel dimension according to the $\ell_1$ norm of each filter, based on which more influential consecutive kernels with larger $\ell_1$ norms are observed for accuracy improvements. Then, the next-layer weights are similarly rearranged in the input channel dimension to ensure the same convolutional results. In contrast to earlier developments [18], [19], [20] that also explore removing kernels, we have a stronger requirement of continuity on the N removed kernels, with its benefit in acceleration because these consecutive kernels can be stored continuously in the memory cache and the convolution with the inputs can proceed using a block-wise vectorized operation in parallel as analyzed in Section 3.4.

We display multiple compression rates using the lightweight MobileNet-V1 [21], -V2 [22], -V3 [23] and large-scale ResNet-50 [2] on the challenging ILSVRC-2012 [24], and compare our 1xN pruning with weight pruning and filter pruning. The experiments suggest obvious increasing accuracy performance compared with filter pruning, and apparent inference acceleration compared with weight pruning. For example, given a pruning rate of 50% and N=4, our 1xN pattern obtains around 3.0% improvements over filter pruning in the top-1 accuracy of MobileNet-V2 on ImageNet, meanwhile, it obtains 56.04ms inference savings on Cortex-A7 CPU compared to weight pruning which obtains no speedup.

This work addresses the problem of simultaneously maintaining accuracy and achieving general CPU speedups to enable practical model deployment on CPUs-based platforms. The key contributions of this paper include: (1) One novel pattern of 1xN for network pruning. (2) A workflow of filter rearrangement for accuracy improvements. (3) Simultaneously maintaining high-performing accuracy and achieving apparent CPUs acceleration.

The remainder of this paper is organized as follows: We briefly discuss some relevant prior works in network pruning in Section 2. Then, we present details of our 1xN pattern for network pruning in Section 3. In Section 4, a discussion on the empirical evaluation of our method in comparison with weight pruning and filter pruning is presented. Moreover, a brief discussion on the limitation of this work is given in Section 5, laying out some avenues for future research in our 1xN pruning pattern. We finally conclude in Section 6.

2 RELATED WORK

Traditional network pruning including weight pruning and filter pruning is a classical research topic. We briefly review some related works below.

Weight Pruning. Weight pruning dates back to Optimal Brain Damage [25] and Optimal Brain Surgeon [26], which prune weights based on the Hessian of the loss function. Despite their accuracy retaining, the second-order Hessian needs additional computation cost. Dong et al. [27] restricted the second-order derivatives for a specific layer to enable tractable computation. Han et al. [4] proposed to recursively remove small-weight connectivity and retrain the $\ell_2$-regularized subnetwork to derive smaller weight values. Dynamic network surgery [28] performs pruning and splicing on-the-fly. The former compresses the network and the latter recovers the incorrect pruning. The lottery ticket hypothesis [29] randomly initializes a dense network and trains it from scratch. The subnets with high-weight values are extracted, and retrained with the initial weight values of the original dense model. Lin et al. [30] proposed a dynamic allocation of sparsity pattern and incorporated feedback signal to reactivate prematurely pruned weights.
Filter Pruning. The norm of filter weight such as $\ell_1$-norm [12] is often considered as an indicator of filter importance. Filters with smaller norms are considered unimportant and removed. He et al. [31] pruned the filter with $\ell_1$-norm criterion, but the pruned filters are changeable and endowed with the chance to be recovered during network training. Ding et al. [32] computed the changes in the next layer’s outputs to evaluate the impact of pruning the filters. Lin et al. [33] used the artificial-bee-colony-based evolutionary algorithm to automatically search for the best pruning structure for each layer. He et al. [34] leveraged reinforcement learning to sample many subnetworks from the original CNN for evaluation, and ultimately find the best compressed network. Liu et al. [13] adopted meta-learning to prune redundant filters. It trains a weight-generating meta-network in advance for subnetworks evaluation, and then searches for the best subnetwork.

Discussion. While a variety of approaches for network pruning have been proposed, existing methods fail to either maintain accuracy or achieve apparent speedups on the general CPUs-based platforms. Thus, it is natural for researchers to go further on pruning neural networks. This motivates our search for designing one new pruning pattern that enables general CPUs acceleration as well as maintains accuracy performance.

3 METHODOLOGY

In this section, we introduce the intuition of our method and present its implementation details. In order to simplify the explanations, we only talk about the convolutional layers as illustrations. However, our $1\times N$ pruning can also be applied to the fully-connected layers since their weights can be regarded as $1 \times 1$ convolutions.

3.1 Preliminaries

We start with notation definitions. Considering a pre-trained $L$-layer CNN model $F$, we denote its filter set as $W = \{W_l\}_{l=1}^L$ with $W_l = \{W^{(n')}_{l,i}\}_{i=1}^{n'} \in \mathbb{R}^{m' \times h' \times w'}$, where $n'$, $m'$, $h'$ and $w'$ respectively indicate the number of output channel, input channel, kernel height and kernel width in the $i$th layer; $W$ is the filter set for the $i$th layer and $W^{(n')}_{l,i}$ is the $j$th filter in the $i$th layer. For the fully-connected layer, its weight matrix is indeed an exception of $h' = 1$ and $w' = 1$. In order to simplify the explanations, in the following contents, we only talk about the convolutional layers as illustrations.

Network pruning can be implemented by imposing a mask $T$ upon $W$. Here $T$ is a binary tensor (0 or 1) with its entries indicating the states of network connections, i.e., whether the corresponding weights are pruned or not. Thus, given an expected pruning rate $p$, network pruning is formally expressed as

$$\arg\max_T \mathcal{L}(W^{(n')} \oplus T), \quad s.t. \frac{\|T\|_0}{K} = 1 - p,$$

where $\oplus$ represents the masking operation, $\mathcal{L}(\cdot)$ measures the importance of its input, and $K$ denotes the size of $T$ that varies according to the basic pruning granularity. We measure the input importance using the $\ell_1$ norm of the basic pruning granularity. We find this criterion sufficient, however, other metrics, such as weight gradients [6], activation sparsity [35], can be adopted as well.

Weight Pruning. The studies on weight pruning remove individual weights at any location of $W$. Therefore, each mask $T$ in weight pruning has the same shape with $W^{(n')} \in \mathbb{R}^{m' \times h' \times w'}$ and its size $K = n' \cdot m' \cdot h' \cdot w'$. The specific objective of weight pruning is

$$\arg\max_T \sum_{j=1}^{n'} \mathcal{L}(W^{(n')}_{j,i} \cdot T^{(n')}_{j,i}), \quad s.t. \frac{\|T^{(n')}\|_0}{K} = 1 - p. \quad (2)$$

Filter Pruning. The studies on filter pruning remove the entire filter $W^{(n')}$. Thus, each mask $T$ in filter pruning has the shape of $\mathbb{R}^{n'}$ and $K = n'$. The specific objective of filter pruning is

$$\arg\max_T \sum_{i=1}^{n'} \mathcal{L}(W^{(n')}_{j,i} \cdot T^{(n')}_{j,i}), \quad s.t. \frac{\|T^{(n')}\|_0}{K} = 1 - p. \quad (3)$$

In the following, we introduce a novel pattern of $1\times N$ pruning, whose basic pruning granularity falls into consecutive $N$ output kernels with the same input channel index. We show that our $1\times N$ pruning can be an efficient and effective alternative to simultaneously accelerate the model inference on modern CPUs-based platforms and retain the accuracy performance.

3.2 $1\times N$ Pruning Pattern

We define the problem of pruning CNNs using our $1\times N$ pattern. In order to simplify the explanations, we reformat the representation of $W^{(n')} \in \mathbb{R}^{m' \times h' \times w'}$ as $\Omega^{(n')} \in \mathbb{R}^{m' \times n'}$. Note that each element in $\Omega^{(n')}_{k,j}$ is a kernel with the shape of $h' \times w'$, i.e., $\Omega^{(n')}_{k,j} = W^{(n')}_{j,k,:}$. Each column $\Omega^{(n')}_{j}$ stands for a filter and each row $\Omega^{(n')}_{k,j}$ consists of these kernels that have the same input channel index of $k$.

As shown in Fig. 1, we further partition the whole $\Omega^{(n')}_{k,j}$ into a collection of smaller blocks. Our partition can be made more precise for an $m' \times n'$ matrix $\Omega^{(n')}_{k,j}$ by partitioning $m'$ into a collection of $m'$ row-groups, and then further partitioning $n'$ into a collection of $n'$ col-groups. Consequently, each block $(k,j)$ is a $1\times N$ matrix including consecutive $N$ output kernels with the same input channel index $k$, namely $\Omega^{(n')}_{k,(j-1):N+1:j,N}$. Based on this partitioned matrix, the basic pruning granularity of our $1\times N$ sparsity falls into these blocks. Thus, the mask, $T^{(n')}$, in our $1\times N$ pruning has the shape of $\mathbb{R}^{m' \times \frac{N}{n'}}$ and its size $K = m' \cdot \frac{N}{n'}$. Finally, the specific objective of our $1\times N$ pruning is

$$\arg\max_T \sum_{k=1}^{n'} \sum_{j=1}^{\frac{N}{n'}} \mathcal{L}(\Omega^{(n')}_{k,(j-1):N+1:j,N} \cdot T^{(n')}_{k,j}), \quad s.t. \frac{\|T^{(n')}\|_0}{K} = 1 - p. \quad (4)$$

Furthermore, we realize that weight pruning and filter pruning are two special cases of our proposed $1\times N$ pruning.
pattern. Specifically, our $1 \times N$ pruning degenerates to weight pruning subject to $N = 1$, $h' = 1$ and $w' = 1$. Besides, it further degenerates to filter pruning if $N = n'$. When $1 < N < n'$, our method provides an intermediate granular level for network pruning, since it is coarser as compared to the fine-grained weight pruning but finer as compared to the coarse-grained filter pruning. Many previous researches [18], [19], [20] also explore removing kernels; however, our pruning pattern has a stronger requirement of continuity on the N removed kernels, with its merits in acceleration since these consecutive kernels can be stored continuously in the memory cache and the convolution with the inputs can proceed using a parallelized block-wise vectorized operation as analyzed in Section 3.4. Therefore, it is expected that our $1 \times N$ pruning can offer a better performance than the filter pruning, and also an apparent inference speedup compared to the weight pruning.

As a distinguished difference from weight pruning and filter pruning, pruning kernels provides an intermediate granular level for network sparsity, since it is coarser as compared to the fine-grained weight pruning but finer as compared to the coarse-grained filter pruning.

### 3.3 Filter Rearrangement

Our $1 \times N$ pruning in this paper follows the typical three-step pipeline of network training, then pruning consecutive kernels with smaller $\ell_1$ norms, and fine-tuning the sparse one to recover the performance in the end. At the second pruning step, inspired by [10] which conducts channel permutations to preserve more high-magnitude weights in the N:M weight pruning, we realize that the layout of $\Omega^\prime$ can be altered to further relieve the pruning impact.

To implement the above process, we propose a workflow of filter rearrangement, whose working principle is shown in Fig. 2. Given the original weight matrix $\Omega^\prime$ in the top-left of (a), simply applying $1 \times N$ pruning upon $\Omega^\prime$ leads to the loss of some kernels with relatively large values of $\ell_1$ norms in the top-right of (a). We calculate the $\ell_1$ norm of each filter, i.e., one column in $\Omega^\prime$, then rearrange $\Omega^\prime$ in the output channel dimension according to the calculated filter norm in a decreasing order as shown in the lower-left of (a). The rearranged weight matrix is denoted as $\Omega^\prime_1$, to which our $1 \times N$ pruning is further applied. As a consequence, more kernels with larger $\ell_1$ norms are preserved after pruning in the lower-right of (a), leading to an overall increasing weight magnitude as verified on MobileNet-V2 of (b). The filter rearrangement requires the outputs to be similarly rearranged as well, so as to maintain the same convolutional results with the next-layer weight matrix. However, frequently rearranging outputs incurs more run-time cost in the inference. Alternatively, we choose to apply a similar rearrangement to the input channel dimension of the next-layer weight matrix as illustrated in the middle of (a), which is accomplished once for all before pruning and thus brings no run-time cost. The effectiveness of filter rearrangement for accuracy improvements is presented in Section 4.

Herein, we want to stress the difference of our filter rearrangement against the channel permutation [10]. First, rearranging the columns of $\Omega^\prime$ is indeed to change the position of each filter in our setting. Second, our goal is to preserve more high-magnitude kernels, while [10] is to preserve more individual high-magnitude weights. Third, we simply accomplish our rearrangement according to the $\ell_1$ norm of each filter. The implementation details of channel permutations are discussed in another paper [36] where a bounded regressions-based permutation search is proposed to find a high-quality permutation.

Then, Eq. (4) after filter rearrangement is rewritten as:

$$
\arg \max_{T^\prime} \sum_{k=1}^{m'} \sum_{j=1}^{w'} \mathcal{L}(\Omega^\prime_{k,(j-1)} N_{N+1:j} N', T^\prime_{k,j}),
$$

subject to $\|T^\prime\|_0 = 1 - p$. (5)

Note that the maximization of the above objective can be achieved by setting to 1s the entries of $T^\prime$ corresponding to
these blocks in $\hat{\Omega}^i$ with their $\ell_1$ norms within the largest top-$(1-p)$, and 0s otherwise. As a consequence, the pruned weights after applying our $1 \times N$ pruning pattern is then derived as

$$\tilde{\Omega}^i = \hat{\Omega}^i + T^i.$$

### 3.4 Encoding and Decoding Efficiency

Given the input activation tensor $X$ as illustrated in Fig. 3, the output tensor $Y^i$ calculated by the standard dense-matrix structure is obtained as

$$Y^i_{k,j} = \sum_{z=1}^{m^i} X^i_{k,z} \hat{\Omega}^i_{z,j}.$$  

(7)

Considering a sparse matrix, operations using the standard dense-matrix structure bear inefficiency as processing and memory are wasted on a large number of zero-valued elements. Thus, it is of great necessity to take advantage of specialized data structures in order to store and manipulate the sparse matrix. However, the irregular sparse matrix resulting from weight pruning requires a great many of indices to record the positions of the reserved weights. Though Compressed Sparse Row Format (CSR) can be used to save index storage, irregular sparsity barely takes advantage of vector processing architectures and memory buses, resulting in little acceleration and even speed deterioration.

Weight pruning leads to an irregular sparse matrix, therefore, a large number of indices are needed to record the positions of the reserved weights. To save the index storage, the relative Compressed Sparse Row (CSR) format is usually adopted [4], [37], which encodes each index by the relative distance (i.e., the number of zeros) between two adjacent non-zero weights. Besides, a decoding process is needed to select the corresponding activations for the reserved weights. The main drawback of irregular pruning is that decoding one index requires a search over the whole activation vector, thus it brings little acceleration, even speed degradation.

In contrast, due to the requirement of continuity on the $N$ removed kernels, our pruning pattern results in a sparse matrix $\Omega^i$ with constant-sized blocks. This good property brings two merits: First, the constant-sized blocks are by nature more easily encoded by Block Compressed Sparse Row Format (BSR) [38] to save non-zero elements in $\Omega^i$ with significantly less storage for the indices. Second, the output tensor $Y^i$ is derived using a block-wise vectorized operation in parallel to achieve an apparent speedup. Specifically, as illustrated in Fig. 3, the pruned weight matrix $\tilde{\Omega}^i$ in our $1 \times N$ pruning is encoded by BSR into three components: 1) $D^i \in \mathbb{R}^{(1+N) \times N}$, 2) $I^i \in \mathbb{R}^t$ and 3) $P^i \in \mathbb{R}^{t+1}$, where $t$ is the number of non-zero blocks in $\tilde{\Omega}^i$. The matrix, $D^i$, and vector, $I^i$, contain non-zero blocks and their row indices in $\tilde{\Omega}^i$, respectively. We form $D^i$ by first stacking up non-zero blocks within the same col-group of $\tilde{\Omega}^i$, and concatenating the stacked ones across different col-groups. The vector $I^i$ records the row index of each block item in $\tilde{\Omega}^i$. The $k$th element of the vector $P^i$ encodes the start row index of the $k$th col-group in $D^i$. The last element of $P^i$ is a fictitious index, which is always equal to $t+1$. Attributes to the block storage format of $D^i$, we can calculate $Y^i$ in a block-wise manner during decoding as follows:

$$Y^i_{k,(j-1)N+1:jN} = \sum_{z=P^i_{j-1}}^{P^i_j-1} X^i_{k,z} \cdot D^i_{z,j}.$$  

(8)

With proper index vectors $P^i$ and $I^i$, we can directly fetch these activations corresponding to non-zero weights for the
4 EXPERIMENTS

4.1 Implementation Settings

For fair comparison, similar to our $1 \times N$ pruning, we re-implement the compared baselines of weight pruning and filter pruning using $\ell_1$ norm as the importance evaluation. Besides, given the expected pruning rate $p$, we simply remove per-layer weights/filters/blocks with their corresponding $\ell_1$ norms within the smallest top-$p$. All experiments are performed using the pre-trained light-weight MobileNet-V1 [21], -V2 [22], -V3 [23] and large-scale ResNet-50 [2] on ILSVRC-2012 [24] that contains over 1.2 million images for training and 50,000 validation images from 1,000 classes. After pruning, we fine-tune the sparse models for 90 epochs on two NVIDIA V100 GPUs with the settings: Stochastic Gradient Descent (SGD) optimizer with a momentum of 0.9, weight decay of 4e-5 for MobileNets and 1e-4 for ResNet-50, and initial learning rate of 0.1 with a cosine annealing. The data augmentation includes random cropping and horizontal flipping.

4.2 Ablation Study

We first analyze the influence of filter rearrangement in Section 3.3. Table 1 compares the performance of our $1 \times N$ pruning for pruning MobileNet-V2 with and without filter rearrangement. The pruning rate $p$ is set to 50%. As can be seen, rearranging filters consistently enhances the accuracy performance in both the top-1 and top-5, even with various block size (N). For example, the top-1 classification accuracy of pruned MobileNet-V2 with $1 \times 16$ pruning is increased by 0.331% (69.352% with and 68.971% without filter rearrangement). To dive into a deeper analysis, by rearranging the weight matrix in the output channel dimension, more blocks with larger $\ell_1$ norms are preserved after applying our $1 \times N$ as validated in Fig. 2b. These results well validate the effectiveness of filter rearrangement in boosting the performance of pruned models.

We continue the study on our consecutive kernel removal. Recall that consecutive $N$ output kernels with the same input channel index are grouped into one block and our $1 \times N$ removes these blocks with a smaller $\ell_1$ norm. In Table 2, we compare our consecutive kernel removal with two variants including (1) removing kernels randomly and (2) removing kernels with smaller $\ell_1$ magnitudes. From these results in Table 2, we can see that removing random kernels performs worse than removing smaller magnitude kernels, indicating the importance of preserving larger magnitude weights. Our stronger constraint of block-level magnitude leads to performance increase compared with the kernel-level magnitude. Also, our $1 \times N$ merits in its acceleration since the consecutive kernels can be stored continuously in the memory cache. Moreover, the convolution with the inputs can proceed using a parallelized block-wise vectorized operation as analyzed in Section 3.4 and verified in the following Section 4.3.

4.3 Performance Comparison

In this section, we compare the performance of our proposed $1 \times N$ pruning with traditional weight pruning and filter pruning. We show that the advantages of pruning pattern are reflected from two perspectives: 1) maintaining better accuracy than filter pruning, and 2) achieving apparent CPUs acceleration compared to weight pruning.

Accuracy Performance. We first study the performance of $1 \times N$ sparsity across different networks. Table 3 displays the pruning results of our $1 \times N$ pruning and existing weight pruning and filter pruning using MobileNet-V1/V2/V3 with the pruning rate $p$ set to 50%. Table 3 shows that filter pruning suffers the most performance degradation of 5.06%, 5.07%, 8.17%, and 5.14% when pruning MobileNet-V1, V2, V3-small and V3-large, respectively. Such severe performance losses are attributed to its coarse-grained pruning granularity. Consequently, the poor

| TABLE 1 | Performance Studies of Our $1 \times N$ Pruning With and Without Filter Rearrangement |
|----------|-----------------------------------------------|
| $N=2$ (%) | $N=4$ (%) | $N=8$ (%) | $N=16$ (%) | $N=32$ (%) |
| Top-1 | Top-5 | Top-1 | Top-5 | Top-1 | Top-5 | Top-1 | Top-5 | Top-1 | Top-5 |
| w/o Rearrange | 69.900 | 89.296 | 69.521 | 88.920 | 69.206 | 88.608 | 68.971 | 88.399 | 68.431 | 88.315 |
| Rearrange | 70.233 | 89.417 | 69.579 | 88.944 | 69.372 | 88.862 | 69.352 | 88.708 | 68.762 | 88.425 |

The experiment is conducted using MobileNet-V2 with the pruning rate $p = 50\%$. | TABLE 2 | Performance Studies of Our $1 \times N$ Pruning With Kernel-Wise Pruning |
|----------|-----------------------------------------------|
| ResNet-50 (%) | MobileNet-V2 (%) |
| Top-1 | Top-5 | Top-1 | Top-5 |
| $1 \times N$ (N=4) | 76.506 | 93.238 | 69.706 | 89.165 |
| kernel (random) | 74.834 | 92.178 | 68.615 | 88.434 |
| kernel ($\ell_1$) | 75.370 | 92.582 | 69.514 | 89.012 |

The experiment is conducted using MobileNet-V2 and ResNet-50 with the pruning rate $p = 50\%$. |
TVM compiler for weight pruning and filter pruning. Besides, additional experiments by TFLite compiler [41] are also presented for weight pruning and filter pruning. After model compiling, we respectively deploy the sparse models to obtain network latencies on the mobile platform of Pixel2 equipped with a Snapdragon 835 CPU and the embedded platform of Zeropi equipped with a Cortex-a7 CPU.

From Fig. 5, we observe no speedup from weight pruning, despite its ability to preserve good performance. As analyzed in Section 1, weight pruning leads to irregular sparsity that hardly utilizes the vector processing architectures and memory buses. Thus, weight pruning often results in little acceleration and even speed deterioration. Filter pruning leads to the most significant speedups since it does not modify the network structure, so that the pruned network can be well fitted by regular hardware to achieve acceleration. Nevertheless, the severe performance degradation disables the using of filter pruning in the model deployment. In contrast, our \(1\times N\) pruning degenerates to weight pruning with a small \(N\) and filter pruning with a large \(N\). The rationale behind this is that a larger \(N\) indicates coarser pruning. As analyzed in Section 3.2, our \(1\times N\) pruning degenerates to weight pruning with a small \(N\) and filter pruning with a large \(N\).

Table 3 also provides the performance comparison when using ResNet-50 as the network backbone with the pruning rate \(p = 50\%\). We can observe similar phenomena to MobileNets that filter pruning suffers the most performance drops and weight pruning presents best performance while our \(1\times N\) provides a trade-off. Besides, our performance decreases as the block size \(N\) increases.

Fig. 4 further shows the performance comparison when applying different pruning rates to sparsifying MobileNet-V2 and ResNet-50. We can see that the increasing pruning rate results in decreasing accuracy performance for all methods. However, filter pruning degrades drastically if \(p > 50\%\). In contrary, our pruning pattern maintains a similar decreasing tendency and close performance to weight pruning even if the pruning rate \(p\) is very high\(^1\).

**CPLUs Acceleration.** Fig. 5 presents the experimental results, which are conducted to further explore the acceleration capacity of different methods on CPUs-based platforms. In Section 3.4, we adopt TVM [39] to compile the pruned model of our pruning pattern. For fair comparison, we also consider the experiment conducted using MobileNet-V1\(-\)V2\(-\)V3 and ResNet-50 with the pruning rate \(p = 50\%\).

### Table 3

|                  | MobileNet (\(p = 50\%\)) | ResNet-50 (\(p = 50\%\)) |
|------------------|--------------------------|--------------------------|
|                  | V1 (%)                   | V2 (%)                   | V3-small (%)               | V3-large (%)               | Top-1 | Top-5 |
|                  | Top-1 | Top-5 | Top-1 | Top-5 | Top-1 | Top-5 | Top-1 | Top-5 |
| Origin           | 71.154 | 89.834 | 71.737 | 90.452 | 67.225 | 87.351 | 74.280 | 91.928 | 77.008 | 93.654 |
| Weight Pruning   | 70.764 | 89.592 | 71.146 | 89.872 | 66.376 | 86.868 | 72.897 | 91.093 | 77.088 | 93.614 |
| Filter Pruning   | 65.348 | 86.264 | 66.730 | 87.190 | 59.054 | 81.743 | 69.137 | 89.097 | 75.382 | 92.518 |
| \(1\times2\) Pattern (Ours) | 70.281 | 89.370 | 70.233 | 89.417 | 65.380 | 86.060 | 72.120 | 90.677 | 76.654 | 93.466 |
| \(1\times4\) Pattern (Ours) | 70.052 | 89.056 | 69.706 | 89.165 | 64.645 | 85.495 | 71.935 | 90.458 | 76.506 | 93.238 |
| \(1\times8\) Pattern (Ours) | 69.908 | 89.027 | 69.372 | 88.862 | 64.101 | 85.274 | 71.478 | 90.163 | 76.146 | 93.134 |
| \(1\times16\) Pattern (Ours) | 69.559 | 88.933 | 69.352 | 88.708 | 63.126 | 84.203 | 71.112 | 90.129 | 76.254 | 93.084 |
| \(1\times32\) Pattern (Ours) | 69.541 | 88.801 | 68.762 | 88.425 | 62.881 | 83.982 | 70.769 | 89.696 | 75.960 | 92.950 |

\(1\). The raw data for plotting Fig. 4 can be found from our project at [https://github.com/limbsma/1xN](https://github.com/linmsma/1xN)

### Limitations

First, our filter rearrangement using \(\ell_1\) norms of filters does not always guarantee maximizing magnitude. Though it does not seem to have happened in the experiments, opportunity exists that lower-magnitude kernels are retained. The approach to cluster large values as in [42] has no this issue. We observe a similar performance of the cluster against our rearrangement. We adopt the \(\ell_1\) norm based rearrangement since it is much easier to implement.

Second, though our \(1\times N\) is originally proposed for CNNs, we observe rare speedups on recurrent neural networks (RNNs) compared to these RNNs acceleration [43], [44]. Nevertheless, we are making efforts to break this limitation and expecting that our method can be well generalized to a wider variety of networks in the future.

Third, this paper misses comparisons to many studies that also explore an intermediate pruning granularity [18], [42], performance barricades the using of filter pruning in practical model deployment. In contrast, due to its fine-grained pruning granularity, weight pruning presents the best performance with top-1 accuracy losses of 0.390\%, 0.591\%, 0.849\%, and 1.383\% when pruning MobileNet-V1, V2, V3-small and V3-large, respectively. Despite its ability to maintain high accuracy, weight pruning achieves rare acceleration as analyzed in the following. The poor speedup also disables the using of filter pruning. With respect to our proposed \(1\times N\) pruning, we have two observations: 1) our method well boosts the performance of filter pruning regardless of the block size \(N\). Taking MobileNet-V2 as an example, our \(1\times4\) pruning achieves 69.706\% top-1 accuracy, significantly better than 66.730\% of filter pruning. Though it is slightly poorer than 71.164\% of weight pruning, our \(1\times4\) pruning obtains an apparent speedup as detailed in the following context. 2) The performance of \(1\times N\) pruning degrades as the block size \(N\) increases. The rationale behind this is that a larger \(N\) indicates coarser pruning. As analyzed in Section 3.2, our \(1\times N\) pruning degenerates to weight pruning with a small \(N\) and filter pruning with a large \(N\).
most of which only report the theoretical acceleration in their papers. In our deployment, we find most of them fail to obtain practical acceleration, or only reach few CPU speedups in a pruning rate of over 90% [45]. Currently, we are not sure if something is wrong in our implementation of these methods. Our future work will focus on this topic to show that our method is worth building on.

Fourth, we do not introduce a new pruning criterion but use the \( \ell_1 \) norm as our measure to reflect the importance of these consecutive kernels. This is because we find that existing pruning criteria show similar performance if fair training settings are given, which is also discussed in [16], [17]. Thus, we focus on designing a new pruning pattern. However, it is unclear if a specialized pruning criterion exists in our \( 1 \times N \) pattern. We will continue excavating this issue.

6 CONCLUSION
We introduce a novel \( 1 \times N \) pruning pattern to simultaneously maintain model accuracy and achieve significant speedups on general CPUs. Unlike previous approaches that prune the individual weights or the whole filters, we design a pruning pattern that supports network pruning by removing consecutive \( N \) output kernels with the same input channel index. To preserve more influential kernels, we propose a workflow of filter rearrangement that rearranges the weight matrix in the output channel dimension and applies similar rearrangement to the next-layer weight matrix in the input channel dimension. Our pruning pattern leads to a sparse matrix with constant-sized blocks enabling the computation outputs by using a parallelized block-wise vectorized operation. The experiments on the MobileNets/ResNet-50 and

Fig. 4. Performance comparison of our \( 1 \times N \) pruning against weight pruning and filter pruning under different pruning rates. The experiment is conducted using MobileNet-V2 (left) and ResNet-50 (right). Best viewed in colors.

Fig. 5. Performance and latency comparison between our \( 1 \times N \) (\( N=4 \)) pruning against weight pruning and filter pruning. The experiment is conducted using MobileNet-V2 on the mobile platform of Pixel2 equipped with a Snapdragon 835 CPU (left), and the embedded platform of Zeropi equipped with a Cortex-a7 CPU (right). Best viewed in colors.
CPUs embedded hardware platforms demonstrate the efficacy of our approach.

References

[1] Y. Sun, Y. Chen, X. Wang, and X. Tang, “Deep learning face representation by joint identification-verification,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2014, pp. 1988–1996.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE/CVF Comput. Vis. Pattern Recognit., 2016, pp. 770–778.

[3] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, “Character-aware neural language models,” in Proc. AAAI Conf. Artif. Intell., 2016, pp. 2741–2749.

[4] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Hierarchical block structured sparse networks from scratch,” in Proc. Int. Conf. Learn. Representations, 2021.

[5] H. Liu, B. Chmiel, M. Island, R. Banner, S. Naor, and D. Sooraj, “Accelerating sparse neural training: A provable and efficient method to find N:M transposable masks,” 2021, arXiv:2102.08124.

[6] A. Mishra et al., “Accelerating sparse deep neural networks,” 2019, arXiv:1906.04655.

[7] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” in Proc. Int. Conf. Learn. Representations, 2019.

[8] D. H. Le and B.-S. Hua, “Network pruning that matters: A case study on retraining variants,” in Proc. Int. Conf. Learn. Representations, 2021.

[9] W. J. Dally, “Fast localized network pruning,” in Proc. IEEE/CVF Comput. Vis. Pattern Recognit. Workshop, 2017, pp. 1927–1934.

[10] M. Rastegari, T. Ordonez, V. Jain, and T. Pentzer, “Faster and more efficient convolutional neural networks,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2019, pp. 2684–2694.

[11] S. Xie, A. Kirillov, R. Girshick, and K. He, “Exploring randomly wired neural networks for image recognition,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2017, pp. 8494–8503.

[12] S. Howard, A. Rastegari, T. Ordonez, and T. Pentzer, “Networks that learn to prune themselves,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2017, pp. 2852–2860.

[13] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” in Proc. Int. Conf. Learn. Representations, 2019.

[14] D. H. Le and B.-S. Hua, “Network pruning that matters: A case study on retraining variants,” in Proc. Int. Conf. Learn. Representations, 2021.

[15] W. J. Dally, “Fast localized network pruning,” in Proc. IEEE/CVF Comput. Vis. Pattern Recognit. Workshop, 2017, pp. 1927–1934.

[16] M. Rastegari, T. Ordonez, V. Jain, and T. Pentzer, “Faster and more efficient convolutional neural networks,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2019, pp. 2684–2694.

[17] S. Xie, A. Kirillov, R. Girshick, and K. He, “Exploring randomly wired neural networks for image recognition,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2017, pp. 8494–8503.

[18] S. Howard, A. Rastegari, T. Ordonez, and T. Pentzer, “Networks that learn to prune themselves,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2017, pp. 2852–2860.

[19] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” in Proc. Int. Conf. Learn. Representations, 2019.

[20] D. H. Le and B.-S. Hua, “Network pruning that matters: A case study on retraining variants,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2014, pp. 1988–1996.

[21] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE/CVF Comput. Vis. Pattern Recognit., 2016, pp. 770–778.

[22] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, “Character-aware neural language models,” in Proc. AAAI Conf. Artif. Intell., 2016, pp. 2741–2749.

[23] H. Liu, B. Chmiel, M. Island, R. Banner, S. Naor, and D. Sooraj, “Accelerating sparse neural training: A provable and efficient method to find N:M transposable masks,” 2021, arXiv:2102.08124.

[24] A. Mishra et al., “Accelerating sparse deep neural networks,” 2019, arXiv:1906.04655.

[25] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” in Proc. Int. Conf. Learn. Representations, 2019.

[26] D. H. Le and B.-S. Hua, “Network pruning that matters: A case study on retraining variants,” in Proc. Int. Conf. Learn. Representations, 2021.

[27] W. J. Dally, “Fast localized network pruning,” in Proc. IEEE/CVF Comput. Vis. Pattern Recognit. Workshop, 2017, pp. 1927–1934.

[28] M. Rastegari, T. Ordonez, V. Jain, and T. Pentzer, “Faster and more efficient convolutional neural networks,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2019, pp. 2684–2694.

[29] S. Xie, A. Kirillov, R. Girshick, and K. He, “Exploring randomly wired neural networks for image recognition,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2017, pp. 8494–8503.

[30] S. Howard, A. Rastegari, T. Ordonez, and T. Pentzer, “Networks that learn to prune themselves,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2017, pp. 2852–2860.

[31] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” in Proc. Int. Conf. Learn. Representations, 2019.

Mingbao Lin received the BS degree from Fuzhou University, Fuzhou, China, in 2016, and the MS degree and PhD degree in intelligence science and technology from Xiamen University, Xiamen, China, in 2022. He is currently a senior researcher with the Tencent Youtu Lab, Shanghai, China. His publications on top-tier conferences/journals include the IEEE Transactions on Pattern Analysis and Machine Intelligence, International Journal of Computer Vision, IEEE Transactions on Image Processing, IEEE Transactions on Neural Networks and Learning Systems, CVPR, NeurIPS, AAAI, IJCAI, ACM MM and so on. His current research interest is to develop efficient vision model, as well as information retrieval.
Yuxin Zhang received the BE degree in computer science from the School of Informatics, Xiamen University, Xiamen, China, in 2020. He is currently working toward the BS degree with Xiamen University, China. His research interests include computer vision and neural network compression & acceleration.

Yuchao Li received the MS degree in computer science from the School of Information Science and Engineering, Xiamen University, Xiamen, China, in 2020. He is currently working toward the algorithm engineer in Alibaba. His research interests include computer vision and neural network compression & acceleration.

Bohong Chen received the BE degree in computer science from the School of Informatics, Xiamen University, Xiamen, China, in 2020. He is currently working toward the master's degree with Xiamen University, China. His research interests include computer vision, and neural network compression & acceleration.

Fei Chao (Member, IEEE) received the BSc degree in mechanical engineering from the Fuzhou University, Fuzhou, China, in 2004, the MSc degree with distinction in computer science from the University of Wales, Aberystwyth, U.K., in 2005, and the PhD degree in robotics from the Aberystwyth University, Wales, U.K., in 2009. He is currently an associate professor with the School of Informatics, Xiamen University, Xiamen, China. He has authored/co-authored more than 50 peer-reviewed journal and conference papers. His current research interests include developmental robotics, machine learning, and optimization algorithms.

Mengdi Wang received the PhD degree in electronic engineering from Tsinghua University, Beijing, China, in 2017. She is currently working with Alibaba Group. Her research interests include efficient deep learning computing, model compression, and neural architecture search.

Shen Li received the MS degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2013. He is currently working as algorithm expert at Platform of AI, Alibaba Cloud. His research interests include deep learning model compression and acceleration.

Yonghong Tian (Fellow, IEEE) is currently a Boya distinguished professor with the Department of Computer Science and Technology, Peking University, China. His research interests include neuromorphic vision, brain-inspired computation, and multimedia Big Data. He is the author or coauthor of more than 200 technical articles in refereed journals such as the IEEE Transactions on Pattern Analysis and Machine Intelligence/IEEE Transactions on Neural Networks and Learning Systems/IEEE Transactions on Multimedia/IEEE Transactions on Circuits and Systems for Video Technology/IEEE Transactions on Knowledge and Data Engineering/IEEE Transactions on Parallel and Distributed Systems, ACM Computing Surveys/ACM Transactions on Information Systems/ACM Transactions on Multimedia Computing, Communications, and Applications and conferences such as NeurIPS/CVPR/ICCV/AAAI/ACMMM/WWW. He was/is an associate editor of the IEEE Transactions on Circuits and Systems for Video Technology (2018.1-), IEEE Transactions on Multimedia (2014.8-2018.8), IEEE Multimedia Magazine (2018.1-), and IEEE Access (2017.1-). He co-initiated IEEE Int’l Conf. on Multimedia Big Data (BigMM) and served as the TPC co-chair of BigMM 2015, and as co-chair of the Technical Program co-chair of IEEE ICME 2015, IEEE ISM 2015, and IEEE MIPR 2018/2019, and general co-chair of IEEE MIPR 2020 and ICME2021. He is the steering member of IEEE ICME (2018-2019) and IEEE BigMM (2019-2020), and is a TPC member of more than ten conferences such as CVPR/ICCV, IEEE KDD, AAAI, ACM MM, and ECCV. He was the recipient of the Chinese National Science Foundation for Distinguished Young Scholars in 2018, two National Science and Technology awards and three ministerial-level awards in China, and obtained the 2015 EURASIP Best Paper Award for Journal on Image and Video Processing, and the Best Paper Award of IEEE BigMM 2018. He is a senior member of the CIE and CCF, a member of the ACM.

Rongrong Ji (Senior Member, IEEE) is currently a professor and the director of the Intelligent Multimedia Technology Laboratory, and the dean assistant with the School of Information Science and Engineering, Xiamen University, Xiamen, China. His research work mainly focuses on multimedia technologies for multimedia signal processing, computer vision, and pattern recognition, with more than 100 papers published in international journals and conferences. He is a member of the ACM. He was a recipient of the ACM Multimedia Best Paper Award and the Best Thesis Award of Harbin Institute of Technology. He serves as an associate/guest editor for international journals and magazines such as the Neurocomputing, Signal Processing, Multimedia Tools and Applications, IEEE Multimedia Magazine, and Multimedia Systems. He also serves as program committee member for several Tier-1 international conferences.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdli.