Complementary Sparsity: Accelerating Sparse CNNs with High Accuracy on General-Purpose Computing Platforms

Anonymous authors
Paper under double-blind review

Abstract

Model sparsity is a promising approach to reduce parameters or FLOPs of convolutional neural networks (CNNs). Compared to unstructured or coarse-grained structured sparsity, fine-grained structured sparsity, e.g., N:M sparse pattern, can achieve better balance between accuracy and efficiency on general computing platforms like CPUs and GPUs. In particular, the 2:4 sparsity can accelerate CNN inference by 2× speed and with negligible accuracy drop. However, N:M sparsity needs to be supported by GPU within specific hardware circuits and hardly achieve significant speedups on common GPUs. To accelerate CNNs with general-purposed computing resources and simultaneously retain the model accuracy as much as possible, this paper proposes complementary sparsity (CS). CS denotes that only one weight can be retained for weights spaced at the same distance. On the one hand, CS features high mask flexibility, which is naturally favorable to high model accuracy. Moreover, we propose a CS-specific sparse training method to improve CS-based CNNs' accuracy under high parameter sparsities (>75%). On the other hand, CS itself is memory-access balanced and robust to pattern hyperparameters, which can be utilized to speedup CS-based convolution computation on CPUs and common GPUs. We thus propose a CS convolution parallel computing algorithm that adapts to common GPUs without sparse tensor cores. Experimental results show that compared to other sparsity patterns, the proposed CS can achieve the optimal trade-off in terms of accuracy and latency for CPUs and common GPUs, respectively.

1 Introduction

Weight sparsification is a crucial method to compress CNNs. The rationality behind weight sparsification is that there are redundant weights in regular CNNs which tend to generate overlapped features (Hoefler et al., 2021; Ayinde et al., 2019). Thus, removing a certain amount of weights in CNNs has little or manageable impact on CNN's accuracy, while can significantly lower CNN's number of floating-point operations (FLOPs) during inference.

According to the extent of pruning freedom and acceleration affinity, the existent weight sparsification technologies for CNNs can be divided into three categories: unstructured sparsity (US), coarse-grained structured sparsity (CSS), and fine-grained structured sparsity (FSS). US, depicted in Fig. 1(a), also called random sparsity in some studies (Huang et al., 2022), permit to prune weights anywhere inside a weight tensor (Zhu & Gupta, 2017; Gale et al., 2019; Mostafa & Wang, 2019; Evci et al., 2020; Kusupati et al., 2020; Liu et al., 2021; Ma et al., 2021; Peste et al., 2021; Tai et al., 2022; Jaiswal et al., 2022; Park & No, 2022; Chen et al., 2021; Li et al., 2022a). Due to US's highest degree of pruning freedom, the most important weights affecting network quality can always be retained under any sparsity. Hence, unstructurally sparsed networks can preserve a decent accuracy even sparsity is very high ($\geq$ 90%). However, the nonuniformity of weight distribution makes it nearly impossible to accelerate US-based convolution on general-purpose computing platforms. In contrast, for CSS, e.g., Fig. 1(b)-(d), the pruning granularity is block, channel or filter -wise (Wen et al., 2016; Li et al., 2016; Gray et al., 2017; Ji et al., 2018; Liu et al., 2017; Tang et al., 2020; Ding et al., 2021; Hou et al., 2022; Liu et al., 2022; Chen et al., 2022; Zhang et al., 2022; He & Xiao, 2023). CSS's patterns are generally with high regularity, which can significantly speedup the network

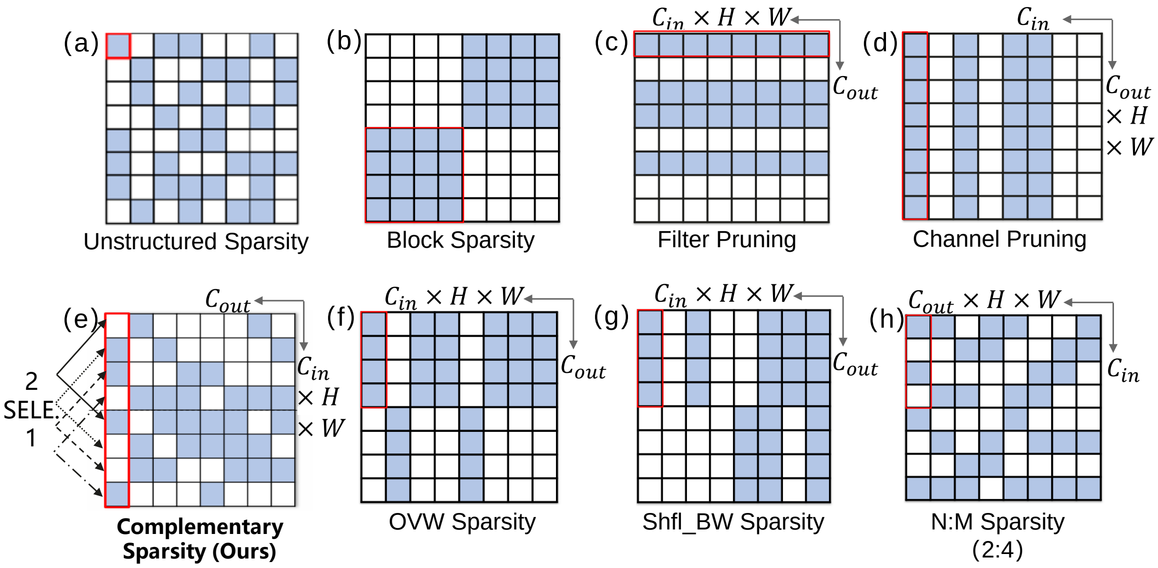


Figure 1: Visualization of different sparse patterns at the 50% sparsity. (a) Unstructured structured sparsity that allows to discard weights of arbitratry positions. (b)-(d) Coarse-grained structured sparsity. (e)-(h) Fine-grained structured sparsity. In particular, (e) is the proposed complementary sparsity and '2 SELE. 1' represents retaining one from two weights in all the complementary positions.

inference. For some of CSS patterns such as filter pruning and channel pruning, i.e., Fig. 1(c) and 1(d), the pruned CNNs can directly operate on the original platforms without any new acceleration algorithm design. Nonetheless, the relatively bigger pruning granularity invetiably leads that many important weights are removed together with unimportant weights. Under the same accuracy, CNNs within CSS own lower compression ratio compared to that within US patterns.

In this study, we focus on FSS since it generally results in better tradeoffs between accuracy and efficiency. We classify a sparsity pattern as FSS if its pruning granularity is vector-wise (Yao et al., 2019; Mishra et al., 2021; Huang et al., 2022; Tan et al., 2022; Meng et al., 2020), e.g., Fig. 1(e)-(h). Compared with US and CSS, FSS possesses both high prunability and decent acceleration affinity. However, the existing FSS pattern more or less have some shortcomings, as shown in Table 1. N:M sparsity, which has been the most popular FSS patterns lately, mainly facilitates inference efficiency on specific hardware, e.g., Amphere arhitecture within sparse tensor cores (Choquette & Gandhi, 2020). Some work tries to accelerate N:M-base convolution on common GPUs without sparse tensor cores (Yao et al., 2019), but the practical speedup benefits compared to the dense convolution is probably limited, since the dense convolution has been adequately optimized and perfectly supported by current common GPUs. Shfl_BW Huang et al. (2022) and OVW Tan et al. (2022) sparse patterns achieve the practical speedups on common GPUs, but CNNs using these two patterns have not reach a satisficatory accuracy so far.

In this paper, we propose a complementarily sparsed pattern to better balance the accuracy and inference efficiency of sparse CNNs. The design principles behind complementary sparsity (CS) is to absorb the non-conflict advantages of the prior sparse patterns as much as possible. Firstly, like N:M sparsity, CS prunes weights inside a vector. Secondly, the positions of the pruned weights and retained weights are complementary. For example, Fig. 1(e) shows a 50% CS. In this subfigure, a vector's shape is 8 × 1, as marked by the red frame. The 'complementary' means that inside the vector, if the 1st weight is pruned, then the 5th weight has to be retained. This logic is the same for the 2nd and 6th weights, and so forth. Lastly, the size of a minimum vector to form CS is variable, which property is called hyperparameter robustness. The hyperparameter robustness of CS makes the pattern very adaptive to different computing platforms, such as CPUs and GPUs.

The major contributions of this paper are as follows:

Pattern	Acceleration w/o ASICs	Accu.
US	Almost impossible	High
Block sparsity	Yes	Low
Filter pruning	Yes	Low
Channel pruning	Yes	Low
N:MMishra et al. (2021)	Hard	High
Shfl_BWHuang et al. (2022)	Yes	Medium
OVWTan et al. (2022)	Yes	Medium
CS (Ours)	**Yes**	**High**

Table 1: Comparison among different sparse patterns.

- We propose a new sparse pattern—CS, which features both the high mask flexibility and high acceleration affinity. CS allows to prune CNNs with less accuracy reduction and to accelerate sparse CNNs on both CPUs and common GPUs.
- Used in the training phase, a CS-specific sparse training method is proposed to boost CS-based CNNs' accuracy under high sparsities. With the propose method, CS-based CNNs performs on par with or better than that with N:M sparsity in terms of accuracy. At the 50% sparsity on ImageNet, CS-based ResNet50 achieves 76.37% accuracy with negligible accuracy loss. At the 93.75% sparsity, CS-based ResNet50 achieves 71.07% accuracy with 5.32% drop, which is better than N:M sparsified ResNet50 that drops 5.8%.
- Used in the inference phase, a parallel acceleration algorithm is proposed to speedup the CS-based convolutions on common GPUs. With the acceleration algorithm, CNNs within CS achieves 2.59×~3.07× speedups at the 93.75% sparsity over the dense counterparts supported by cuDNN.

Through the algorithm-software co-optimization, the proposed CS reaches the better tradeoffs between sparse CNNs' model quality and inference efficiency compared with other fine-grained structured sparse patterns. To be clear, the advatanges of our CS over similar works are shown in Table 1.

2 Related Work

Unstructured sparsity (US) Neural networks within US has been researched for a long time. The wining ticket hypothesis denotes that there always exists a sparse neural network inside a dense network and the subnetwork can be trained to reach the comparable accuracy as the dense one (Frankle & Carbin, 2018). Since the hypothesis is proposed, a surge of studies have focused on developing good pruning methods to form US. Gale et al. (2019); Zhu & Gupta (2017) improve the magnitude-based pruning methods simply by gradually sparsifying and prolonging the training time, respectively. Rather than fully training a dense network before pruning, Mostafa & Wang (2019); Evci et al. (2020); Ma et al. (2021) adopt the sparse training to directly generate the unstructured sparse neural networks. These sparse training methods basically contain a common mechanism that periodically prunes and regrows some weights according to certain criterion. Furthermore, Peste et al. (2021) alternatively conducts sparse and dense training. In this way, both a dense and a unstructured sparse neural networks are generated after training. Instead of pruning weights by carefully designed criterions, Kusupati et al. (2020); Tai et al. (2022) learn the sparse masks by differentiation. In particular, the proposed method in Tai et al. (2022) reports the state-of-the-art accuracy of US-based CNNs on the ImageNet dataset. Generally, CNNs within US can not obtain significant speedup gains on CPUs and common GPUs due to the severe memory-access conflict.

Coarse-grained structured Sparsity (CSS) Normally, CSS can be enforced along either input or output axes of a weight tensor. Separately shrinking the output and input axis is called filter and channel pruning, respectively. Pruning a square block in both input and ouput axes are called block sparsity (Gray et al., 2017). To our knowledge, Wen et al. (2016) is the first to propose filter pruning or channel pruning for CNN compression. In their study, the group LASSO method is directly enforced into weights to induce sparsities among filters or channels. Some studies like Liu et al. (2017); Ding et al. (2021) employ extra indicators to evaluate the filters, e.g., scaling factors in batch normalization layers, or propoerly placed 1×1 convolutions that can be absorbed during inference. Since pruning filters also result in pruning the related channels in the next layers, the proposed method in Li et al. (2016) jointly considers the impact of filter and channel pruning on network accuracy. Rather than developing various hypotheses to measure the important of filters, Tang et al. (2020) assesses filters by observing the network responses to real data and adversial samples. Besides, some principles originally used for US have lately been introduced to realize CSS, e.g., Hou et al. (2022); Tai et al. (2022). Despite these efforts, CSS-based CNNs' accuracy is still relatively lower and drops drastically especially when the required sparsity$> 70\%$.

Fine-grained structured sparsity (FSS) N:M sparsity is a well-known fine-grained structured sparse pattern where at most N non-zero weights are retained for every continuous M weights (Yao et al., 2019; Mishra et al., 2021; Lu et al., 2023; Zhang et al., 2023). However, N:M sparse pattern needs to be supported by GPUs embeded with sparse tensor cores. On common GPUs, the pattern hardly outperforms dense convolutions supported by cuDNN. To tackle the problem, Shfl_BW and OVW sparsity regard a $M \times 1$ vector as an entirety which is pruned or retained together (Huang et al., 2022; Tan et al., 2022). By this design, the retained weights and the related features during convolution computaion can be easily indexed. Thus, Shfl_BW and OVW sparsity can accelerate convolutions on common GPUs to a great extent. However, the relatively large pruning unit of $M \times 1$ still decreases the flexility (Hubara et al., 2021), which results in the reduced model accuracy. In contrast, our CS can help maintain the similar or better model accuracy relative to N:M sparsity, as well as achieve the practical speedups of sparse convolutions on common GPUs and CPUs.

GPU acceleration for convolutions So far, there are bascially four sorts of parallelable algorithms to implement convolutions: direct convolution, Winograd (Chikin & Kryzhanovskiy, 2022), FFT (Wang et al., 2020), explicit general matrix multiplication (GEMM) (Jiang et al., 2022) and implicit GEMM (Zhou et al., 2021b). Among these, GEMM-base convolution algorithms are more performant on GPUs since the modern parallel computing platforms have highly optimized the GEMM operations (Chetlur et al., 2014; Jorda et al., 2019; Li et al., 2022b). However, explicit GEMM-based convolutions need to firstly inovke img2col to change tensors to matrices, which is memory access-intensive and time-consuming. By contrast, implicit GEMM-based convolutions remove the memory access overheads, which is top-performed in most cases. Moreover, Tan et al. (2022) employs the implicit GEMM to accelerate sparse convolutions, which implies the potential of implicit GEMM to speedup other sparse patterns. In this work, the implicit GEMM is also utilized to develop the parallel acceleration algorithm of CS-based convolutions on common GPUs.

3 Method

3.1 Complementary Sparsity Formulation

For a sparse weight tensor W with the shape of $C_{out} \times C_{in} \times F_h \times F_w$, each filter W_i has the shape $C_{in} \times F_h \times F_w$ and is flattened. We use $W_i[j]$ to denote the jth weight in the flatttened W_i. S is the sparsity of the weight tensor. Two key parameters are introduced to conviently describe CS: 1) K. K denotes the amount of the complementary positions from which only one single weight should be selected. For instance, at the 50% sparsity, there should be $K = 2$ complementary positions for selecting a weight. for the 75% sparsity, there are $K = 4$ complementary positions from which a weight is selceted. 2) M. M represents the address offset with which weights in the complementary positions can mutually be located. Specifically,

$$K = \frac{1}{1 - S} \tag{1}$$

$$M = \frac{L}{K}, L \in \{C_{in}/c, C_{in} * F_h * F_w\} \tag{2}$$

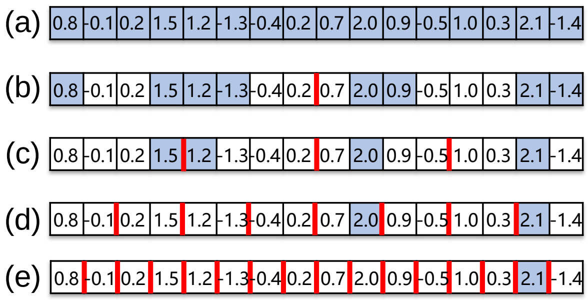
In Equation 2, if $L = C_{in}/c$, the typical values of M include $2, 4, 8, 16$. Here c only means that C_{in} should be divisible by M. Then, a sparse weight tensor is regarded as complementarily sparsed as long as for any $W_i[j], i \in [1, C_{out}], j \in [1, M]$,

$$\|W_i[j], W_i[j + 1 * M], ...W_i[j + (K - 1) * M]\|_0 < K \tag{3}$$

Furthermore, a sparse weight tensor is regarded as strictly conforming to the pattern of CS if and only if

$$\|W_i[j], W_i[j+1*M], ...W_i[j+(K-1)*M]\|_0 = 1 \tag{4}$$

To intuitively understand the definition of CS, Fig. 2 gives some examples strictly obeying the pattern of CS across multiple sparsities. Accordingly, the parameter values, i.e., K and M of each example, are listed in Table 2. Note that K is only related to the specified sparsity S and is independent of the specific weight tensor shapes. Unless otherwise specified, the rest of the paper only discusses the strict CS.



(a) 0.8 -0.1 0.2 1.5 1.2 -1.3 -0.4 0.2 0.7 2.0 0.9 -0.5 1.0 0.3 2.1 -1.4

(b) 0.8 -0.1 0.2 1.5 1.2 -1.3 -0.4 0.2 0.7 2.0 0.9 -0.5 1.0 0.3 2.1 -1.4

(c) 0.8 -0.1 0.2 1.5 1.2 -1.3 -0.4 0.2 0.7 2.0 0.9 -0.5 1.0 0.3 2.1 -1.4

(d) 0.8 -0.1 0.2 1.5 1.2 -1.3 -0.4 0.2 0.7 2.0 0.9 -0.5 1.0 0.3 2.1 -1.4

(e) 0.8 -0.1 0.2 1.5 1.2 -1.3 -0.4 0.2 0.7 2.0 0.9 -0.5 1.0 0.3 2.1 -1.4

Figure 2: Examples of CS at different sparsities. The blue shading of (b)-(e) indicates the selected weights. (a) a dense weight tensor. For convience, the tensor is 2D and with the shape of 1 × 16, i.e., $C_{out} = 1$ and $C_{in} * F_h * F_w = 16$ (b) The 50% CS of the weight tensor. (c) The 75% CS of the weight tensor. (d) The 87.5% CS of the weight tensor. (e) The 93.75% CS of the weight tensor.

Table 2: Parameter values of the examples shown in Fig. 2.

Sparsity (%)	K	M	Encoding bit number	Encoding results
50	2	8	1	0,1,1,0,0,0,1,1
75	4	4	2	1,2,3,0
87.5	8	2	3	7,4
93.75	16	1	4	14

3.2 CS-specific Gradual Sparse Training Method

The aim of designing a CS-specific sparse training method is to attain universality. That is, the desired training method can improve the accuracy of various CS-based CNNs among different sparsities. With the training method, users do not need to customize training recipes for different CNN structures.

Conventionally, training a sparse neural networks follows the process of dense training, single-shot pruning and finetuning. We argue the conventional training process is unfavorable to CS-based CNNs under high sparsities (> 75%) in which case the derived sparse masks from the dense weight tensors tend to be suboptimal. In contrast, we propose a two-phase training scheme: gradual sparse training followed with retraining. As demonstrated in Algorithm 1, assuming that the allowed training iteration amount in each phase is I, the first phase starts training with a dense network and the sparsity of the network discretely steps up every P iterations. The completeion of the first phase offers the sparse masks conforming to the pattern of CS and the weights as the initialization of the second phase. The second phase simply uses the same training setups as the first phase to retrain the retained weights in newtorks selected by the sparse masks. Obviously, the novelty of our training method mainly lies in the first phase. The first phase comprises two features as detailed below.

Variable number of steps for gradual sparsification. By emprical observation, we find the higher the required sparsity is, the more steps for gradual sparsification are needed to obtain the better model quality. Hence, to obtain a CS-based CNN at the sparsity S, we set K steps for gradually sparsifying the network. For instances, in case that the target sparsity is 75%, the change of sparsities during training in the first phase is 0% → 25% → 50% → 75%, while for the target sparsity of 87.5%, the change is 0% → 12.5% → 25% → 37.5% → 50% → 62.5% → 75% → 87.5%. To be formulated, for the ith training iteration, the desired sparsity S_i is:

$$S_i = \frac{ceiling\{\frac{i}{I} * K\} - 1}{K} \tag{5}$$

Equation 5 intuitively means that S_i performs the piecewise linear growth as training iterations. The position of Equation 5 in the workflow of our training method is shown in Algorithm 1. Note that the gradual sparsification only means the amounts of weights paticipating in the forward passes are gradually and discretely reduced. During backward passes, the gradients of all the weights are computed and every weight is updated no matter which sparsity step a network is at.

Algorithm 1 Workflow of Our Training Method

```
Initialization: Dense weights $W$
Input: Required sparsity $S$, training iterations $I$, data $D$
Output: Sparse weights $W^S$
Key params: $K$, sparse mask $M$, mask updating freq. $F$
——————-Gradual Sparse Training Phase——————-
for $i = 0; i < I; i++$ do
    if $i\%F$ then
        $S_i$←Equation 5$(i, I, K)$
        $M$ ← Weights_Reselect( $S_i, W$ )
    end if
    Forward$(W, M, D)$
    Backward$(W, D)$
    Weights_Update$(W)$ #Update all weights
end for
————————Retraining Phase————————
for $i = 0; i < ite; i++$ do
    Forward$(W, M, D)$
    Backward$(W, M, D)$
    Weights_Update$(W, M)$ #Update selected weights
end for
————————————————————————————
$W^S \leftarrow W * M$
```

CS-specific weight reselection. Since all the weights are kept updating in the first training phase, it is beneficial to reselect the weights, i.e., update the sparse masks once in a while. Supposing every F iterations, the sparse masks should be updated. The update process for CS is: 1) Reshape. A weight tensor with the shape $C_{out} \times C_{in} \times F_h \times F_w$ is reshaped into $G \times K \times L$, where K and L is defined in Equation 1 and 2, respectively. G can be infered given K and L. 2) Reselection. Along K axis to reselect k_i weights with the highest absolute value. k is related to the sparsity step that a network is at, i.e.,

$$k_i = (1 - S_i) * K \tag{6}$$

The positions of the reslected weights in masks are set ones while the other positions are set zeros. 3) Restoration, which means inversely reshape the weight tensor from $G \times K \times L$ back to $C_{out} \times C_{in} \times F_h \times F_w$. Fig. 3 shows an example of the CS-specific weight reselection procedure, which mainly visualizes the step 2). Notably, the gradual sparsification is exactly achieved by our weight reselection procdeure by properly setting F to make P divisble by F.

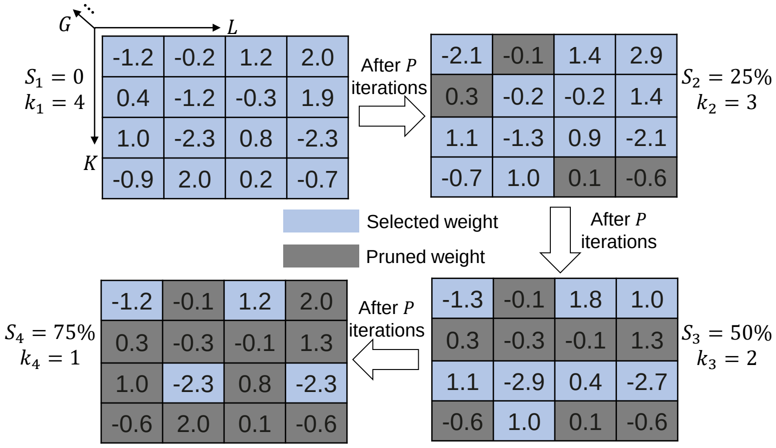


Figure 3: An example of CS-specific weight reselection.

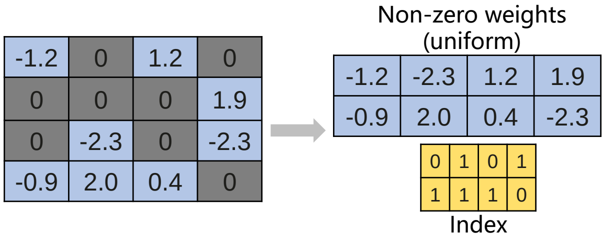


Figure 4: A diagram of CS compression at the 50% sparsity.

3.3 Acceleration of CS-based Convolution

The obtained sparse weight tensors after training is stored in the compressed format. That is, only the non-zero weights and their indices are stored and used for inference. This procedure is formulated by, converting W^S to W^s and Idx, where W^s is the non-zero weight tensor with the smaller shapes and Idx is the index tensor. The index of an non-zero weight denotes the weight's position number among K complementary positions, thus the value of an index is constantly less than K. Fig. 4 exemplifies the compression of CS at the 50% sparsity. In this case, the W^s has half of the shape than W^S. Although Idx has the same shape as W^s, each index in Idx can be encoded with very low numbers of bits. As shown in Table 2, only at most 4 bits are needed to encode a non-zero weight's index.

After acquiring W^s and Idx, given an input activation X, the output featuremap Y can be computed in parallel. Contrary to the conventional method of feature reuse to speedup sparse convolutions Tan et al. (2022) on common GPUs, we propose a weight resue-based algorithm. In implicit GEMM-based dense convolutions, each thread is generally in charge of a subblock in Y, e.g., of size $4 * 4$. Then,

$$Y_{i:i+4,j:j+4} = \sum_{n=0}^{L} W_{i:i+4,n} \otimes X_{n,j:j+4} \tag{7}$$

In Equation 7, L is only equal to $C_{in} * F_h * F_w$ for GPUs, and $\otimes$ represents the outer product operation of two vectors. When CS-based convolution is conducted, Equation 7 can be modified into:

$$Y_{i:i+4,j:j+4} = \sum_{n=0}^{L*(1-S)} W_{i:i+4,n} \circ X_{f(n,i),j:j+4}, \tag{8}$$

where

$$f(n,i) = n + M * Idx[i : i+4, n] \tag{9}$$

and $\circ$ in Equation 8 denotes the operation:

$$\circ = \begin{bmatrix} W_i * X_{n+M*Idx[i,n],j:j+4} \\ W_{i+1} * X_{n+M*Idx[i+1,n],j:j+4} \\ W_{i+2} * X_{n+M*Idx[i+2,n],j:j+4} \\ W_{i+3} * X_{n+M*Idx[i+3,n],j:j+4} \end{bmatrix} \tag{10}$$

Equation 8 makes it possible to compute CS-based convolutions by chunk. For a non-zero weight, there are K related sub-blocks in an input activation that may be indexed during convolution. On common GPUs, by loading all the related K sub-blocks to GPUs' share memory in advance, Equation 8 can be conducted efficiently. Algorithm 2 shows this parallel acceleration process, where 50% CS-based convolution is taken for example.

On CPUs, the direct method is employed to accelerate CS. Due to the instruction limit, CPUs can hardly fetch multiple values far apart from each other. Accordingly, our CS allows different values of M without reducing network accuracy, which is very friendly to CPU operation. During convolutions, CPUs conduct sparse vector products along the C_{in} axis. In this case, $L = C_{in}/c$.

Algorithm 2 Parallelization for 50% CS-based convolution

```
Input: $Idx$, $W$, $X$
Output: $Y$
Key params: $M, L$
___Shared___ float4 $local\_w, local\_idx$
___Shared___ float4 $local\_x[2], local\_y$
————————————Parallelism————————————-
for all $N * OC * OH * OW/16$ threads do
    Subblock(Y) ← SubConv( thread[i], $Idx$,$W$, $X$ )
end for
————————————Details————————————
function SubConv(tid, $Idx$,$W$, $X$)
    for $i = 0; i < L; i+ = BM$ do
        $local\_filter$ ← Subblock($filter$)
        $local\_idx$ ← Subblock($Idx$)
        $local\_x[0]$ ← Subblock1($X$)
        $local\_x[1]$ ← Subblock2($X$)
        Syncthreads();
        $local\_y$ = Eq. 8($local\_w$, $local\_x$, $local\_idx$)
    end for
    return $local\_y$
end function
```

4 Experiments

4.1 Datasets, Models and Settings

CIFAR100 Krizhevsky et al. (2009) and ImageNet-1k Deng et al. (2009) are two datasets used to test the accuracy of CS-based CNNs. Specifically, on CIFAR100, we evaluate three classical CNNs including VGG-19 Simonyan & Zisserman (2014), ResNet-18 and ResNet-50 He et al. (2016), and two parameter-efficient CNNs including MobileNetv2 and SqueezeNet Sandler et al. (2018); Iandola et al. (2016). Since on CIFAR100, low sparsities ($\leq$ 75%) may result in insignificant accuracy differences between CNNs with our CS and with other sparse patterns, we test the three classical CNNs under high sparsities: 87.5% and 93.75%, while for MobileNetv2 and SqueezeNet, accuracy results under the 50% and 75% sparsities are adequately distinguishable for comparison. At each sparsity, each CNN is sparsified by US, filter pruning, N:M, OVW and CS, respectively. All the sparse CNNs are firstly trained with the same training paradigm: dense training, pruning, and finetuning. This paradigm has been widely used to form various sparse CNNs. For simplicity, the finetuning phase uses the same setting as the dense training phase, which has been verified as resonable in Mishra et al. (2021). After that, CS-based CNNs are trained with the proposed CS-specific gradual training method for comparison. All the trainings use the common and the same settings. The total training epoch is 400. For the conventional training paradigm, the dense training phase uses 200 epochs and the finetuning phase uses the rest. Similarly, for our CS-specific gradual training method, each training phase equally uses 200 epochs as well. Each experiment is repeated three times and all the experimental results on CIFAR100 are listed in the format of "mean± standard deviation" to reduce the influence of random factors.

On ImageNet-1k, CS-based ResNet-50 at different sparsities are trained for comparing with other related works. We use the officially recommended hypermeter settings for our sparse training method zlm (2022). Besides, the speedups of CS-based CNNs over the dense counterparts are measured on a Intel(R) Xeon(R) Gold 6154 CPU and a Tesla V100 GPU without sparse tensor cores, respectively.

4.2 Results on CIFAR100 and ImageNet

Sparsity (%)	Origin	VGG19	Resnet18	Resnet50	Sparsity (%)	Origin	Mobilenetv2	SqueezeNet
		70.8	74.7	72.2			65.3	65.1
87.5	Unstructured	71.3±0.1	73.3±0.1	72.3±0.3	50	Unstructured	66.9±0.1	67.7±0.2
	Filter pruning	47.8±0.6	59.0±0.2	48.4±1.0		Filter pruning	65.5±0.5	36.4±0.2
	N:M(1:8)	70.6±0.1	72.9±0.1	72.1±0.1		N:M(2:4)	66.4±0.3	67.2±0.1
	OVW	63.6±0.3	64.1±0.7	59.2±1.6		OVW	61.8±0.3	64.0±0.4
	CS-C	70.7±0.2	73.0±0.1	72.5±0.3		CS-C	66.4±0.1	67.2±0.1
	CS (Ours)	**71.7±0.3**	**73.5±0.1**	**73.1±0.0**		**CS (Ours)**	**66.7±0.2**	**67.0±0.2**
	△	+1.4	+0.5	+0.6		△	+0.2	-0.2
93.75	Unstructured	69.8±0.1	71.5±0.0	71.3±0.0	75	Unstructured	61.7±0.1	64.5±0.2
	Filter pruning	33.2±0.4	47.9±0.4	40.0±0.6		Filter pruning	53.4±1.1	15.9±0.2
	N:M(1:16)	68.1±0.3	70.4±0.2	70.6±0.3		N:M(1:8)	58.9±0.4	63.4±0.3
	OVW	59.6±0.2	60.6±0.3	41.0±13.8		OVW	Failed	54.6±0.9
	CS-C	68.5±0.2	70.5±0.1	70.5±0.2		CS-C	59.8±0.1	63.9±0.3
	CS (Ours)	**69.9±0.1**	**72.2±0.3**	**73.0±0.3**		**CS (Ours)**	**62.6±0.2**	**64.4±0.1**
	△	+1.4	+1.7	+2.5		△	+2.8	+0.5

Table 3: Accuracy of sparse CNNs on CIFAR100. 'CS-C'represents CS-based CNNs formed by the **C**onventional training paradigm, while 'CS (Ours)' means that formed by the proposed training method. '△' means the difference between 'CS-C' and 'CS (Ours)'. For spatial brevity, all the data are rounded to one significant digit.

Table 3 shows the accuracy of different networks within different sparse patterns on CIFAR100. Firstly, for the three classical CNNs, our CS achieves the best accuracy under high sparsities among all the sparse patterns. Secondly, for MobileNetv2 and SqueezeNet, our CS also outperforms other fine-grained structured sparse pattern at the 75% sparsity. In particular, the accuracy of MobileNetv2 within CS at the 75% sparsity is even higher than that within the unstructured sparsity, i.e., 62.63%>61.7%. Thirdly, the proposed training method significantly improves accuracy of a series of CS-based CNNs, with the average accuracy increasing from 0.48% to 2.83%. Notably, at the 50% sparsity, all types of sparse patterns lead to lossless accuracy. In this case, we argue the accuracy difference among the patterns and training methods are immaterial.

Table 4 shows the experimental results on ImageNet. Compared with the OVW and Shfl_BW pattern, our CS leads to the better accuracy under high sparsities, e.g., 93.75%. For other sparsities, our CS achieves comaprable accuracy with the state-of-the-art N:M sparsity. However, the different settings of M in N:M sparsity significantly affect the network accuracy, e.g., Sun et al. (2021). On the contrary, our CS is robust to the pattern hypermeter setting which will be shown in ablation study.

Pattern	Sparsity	Error(%)			Params	Flops
		Ori.	Pruned	Gap	(M)	(G)
N:M	2:4	77.3	77.0	0.3	13.8	2.15
OVW	50%	76.12	75.76	0.36	13.8	2.15
CS(Ours)	**50%**	**76.39**	**76.37**	**0.02**	**13.8**	**2.15**
N:M	1:4	77.3	75.3	2	7.93	1.17
OVW	70%	76.12	73.35	2.77	9.14	1.37
CS(Ours)	**75%**	**76.39**	**75.18**	**1.21**	**7.93**	**1.17**
Shfl_BW	80%	N/A	75.94	N/A	6.78	0.98
CS(Ours)	**87.5%**	**76.39**	**72.44**	**3.95**	**5.02**	**0.69**
N:M	1:16	77.3	71.5	5.8	3.52	0.44
Shfl_BW	90%	N/A	73.09	N/A	4.43	0.59
CS(Ours)	**93.75%**	**76.39**	**71.07**	**5.32**	**3.52**	**0.44**

Table 4: ResNet50 accuracy comparison among different fine-grained structured sparse patterns on ImageNet. The results of N:M, OVW, and Shfl_BW are from Zhou et al. (2021a), Tan et al. (2022) and Huang et al. (2022), respectively. 'N/A' means the related work does not report the result.

4.3 Ablation Study

Firstly, we investigate the effect of different mask updating frequencies in our training method, i.e., F mentioned in Algorithm 1, on network accuracy. The results are shown in Table 5. In the table, $F = 0$ means updating the mask once for every iteration, $F = 0.5$ means that udpating frequency is 0.5 epoch, and so on. We find that the higher the required sparsity is, the higher the mask updating frequency should be. For example, at the 50% sparsity, $F = 8$ is the best, while at 93.75%, $F = 0$ outperforms others. The F settings in Table 5 is exactly used in training CS-based ResNet50 on ImageNet. Secondly, we investigate a pattern hypermeter of CS: M. Specifically, under the same sparsity decided by K, CS-based CNNs with different settings of M are trained and we conduct pairwise t-test on these CNNs' accuracy. As shown in Table 6, all the p values are larger than 0.05, which indicates that our CS is robust to pattern hyperprmeters. The robustness is quite beneficial for acceleration as CPUs and GPUs can employ different values of M to meet the respective constraints in memory access and instruction set.

F	50%	75%	87.5%	93.75%
0	74.7±0.08	74.14±0.63	73.44±0.19	**72.06±0.38**
0.5	74.78±0.09	**74.55±0.06**	**73.57±0.27**	71.9±0.52
1	74.68±0.06	74.35±0.37	73.47±0.6	71.62±0.2
2	74.71±0.24	74.12±0.23	73.16±0.37	71.19±0.59
4	74.85±0.08	74.21±0.08	72.85±0.23	70.85±0.48
8	**74.93±0.37**	74.28±0.21	72.54±0.61	70.47±0.17
10	74.66±0.19	74.5±0.19	73.06±0.17	70.14±0.09

Table 5: ResNet18 on CIFAR100: Impact of F across sparsities.

4.4 Results on Speedups

On CPU, M is set 2,4,8,16 on demand. On GPUs, we set M as large as possible, i.e., $M = C_{in} * F_h * F_w / K$. We find that the larger M makes indexing more easier on GPU. Fig. 5 shows the normalized speedups on CS. Firstly, three typical convolutions in ResNet50 are used for test speedup. As shown in Fig. 5(a), with batch size equal to 1, CPU achieves 4.27×, 5.46× and 7.7× speedups for the 3rd, 11th, 41th convolutions at the 93.75% complementary sparsity, respectively. Similarly, as shown in Fig. 5(b), with batch size equal to 64, GPU respectively achieves 4.02×, 3.33× and 2.52× speedups at 93.75% over dense counterparts supported by cuDNN. Secondly, we also estimate the speedups on network-level by averaging all the convolutions' runtime. GPU achieves 2.75×, 3.07× and 2.59× speedups for VGG19, ResNet50 and SqueezeNet, respectively. These

Sparsity	(M, K)	VGG19	ResNet18	ResNet50	Sparsity	(M, K)	MobileNetv2	SqueezeNet
87.5%	(2,8)	71.28±0.15	73.68±0.32	73.46±0.85	50%	(2,2)	66.81±0.14	67.19±0.29
	(4,8)	71.46±0.49	73.2±0.39	73.23±0.65		(4,2)	66.65±0.15	67.07±0.04
	(8,8)	71.55±0.39	73.47±0.16	73.38±0.39		(8,2)	N/A	67.27±0.09
	t-test	**0.77**	**0.29**	**0.82**		**t-test**	**0.18**	**0.57**
93.75%	(2,16)	69.62±0.09	71.89±0.19	72.23±1.4	75%	(2,4)	62.72±0.14	64.26±0.07
	(4,16)	69.76±0.19	72.22±0.39	72.56±0.11		(4,4)	N/A	64.3±0.22
	t-test	**0.31**	**0.39**	**0.71**		**t-test**	N/A	**0.7**

Table 6: Accuracy of CS-based CNNs with different settings of M on CIFAR100

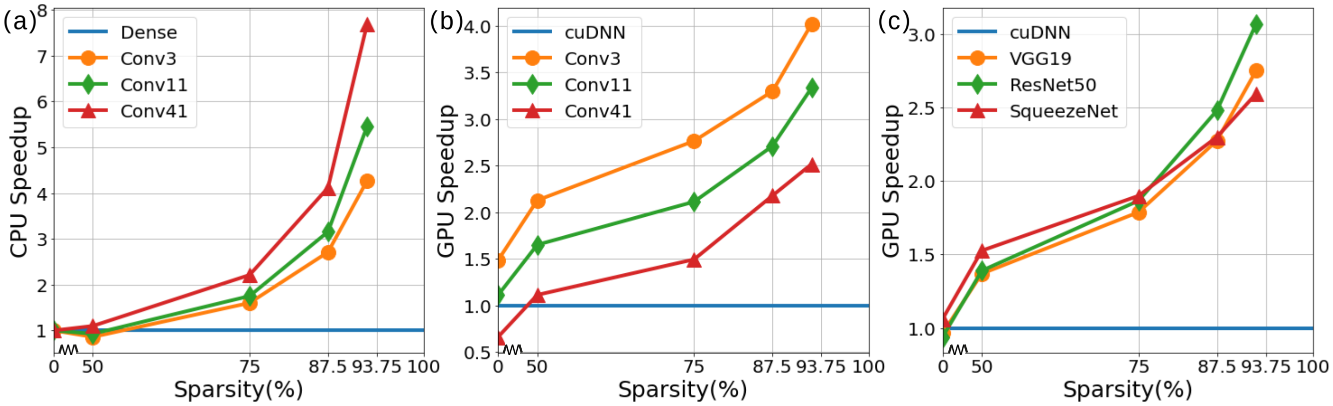


Figure 5: Speedups for CS. (a) CPU speedups for three CS-based convolutions in ResNet50. (b) GPU speedups for three CS-based convolutions in ResNet50. (c) GPU speedups for three CS-based CNNs.

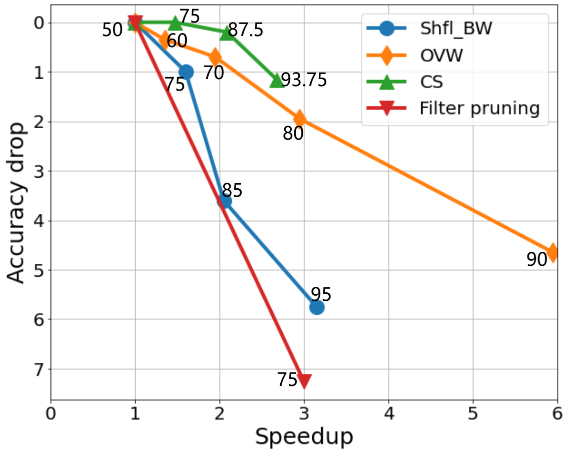


Figure 6: ResNet50 on CIFAR100: Normalized accuracy-speedup curves of three fine-grained structured sparse patterns. Note N:M sparsity does not have acceleration gains on common GPUs.

speedup performance proves the efficiency of the proposed parallel acceleration algorithm. In addition, our CS generally reaches the better accuracy-speedup tradeoffs compared with the OVW and Shfl_BW pattern, as shown in Fig. 6.

5 Conclusion

We propose a novel CS to accelerate sparse CNNs on CPUs and common GPUs and retain the network accuracy as much as possible. To our knowledge, we are the first to report the practical speedups on both CPUs and common GPUs for a sparse pattern. Not only does the proposed CS feature high mask flexibility that contributes a lot to sparse CNNs' accuracy, but also the network accuracy is robust to pattern hyperparameters. The robustness enhances CS's adaptability to different computing platforms. In the future, we will extend our CS to more network architectures, e.g., ViTs, and more computer vision tasks, e.g., detection and segmentation.

References

Image classification reference training scripts, 2022. `https://github.com/pytorch/vision/tree/main/references/classification`.

Babajide O Ayinde, Tamer Inanc, and Jacek M Zurada. Regularizing deep neural networks by enhancing diversity in feature extraction. *IEEE transactions on neural networks and learning systems*, 30(9):2650–2661, 2019.

Tianlong Chen, Xuxi Chen, Xiaolong Ma, Yanzhi Wang, and Zhangyang Wang. Coarsening the granularity: Towards structurally sparse lottery tickets. In *International Conference on Machine Learning*, pp. 3025–3039. PMLR, 2022.

Tianyi Chen, Bo Ji, Tianyu Ding, Biyi Fang, Guanyi Wang, Zhihui Zhu, Luming Liang, Yixin Shi, Sheng Yi, and Xiao Tu. Only train once: A one-shot neural network training and pruning framework. *Advances in Neural Information Processing Systems*, 34:19637–19651, 2021.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

Vladimir Chikin and Vladimir Kryzhanovskiy. Channel balancing for accurate quantization of winograd convolutions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12507–12516, 2022.

Jack Choquette and Wish Gandhi. Nvidia a100 gpu: Performance & innovation for gpu computing. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pp. 1–43. IEEE Computer Society, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4510–4520, 2021.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pp. 2943–2952. PMLR, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Scott Gray, Alec Radford, and Diederik P Kingma. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*, 3:2, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Yang He and Lingao Xiao. Structured pruning for deep convolutional neural networks: A survey. *arXiv preprint arXiv:2303.00566*, 2023.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021.

Zejiang Hou, Minghai Qin, Fei Sun, Xiaolong Ma, Kun Yuan, Yi Xu, Yen-Kuang Chen, Rong Jin, Yuan Xie, and Sun-Yuan Kung. Chex: channel exploration for cnn model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12287–12298, 2022.

Guyue Huang, Haoran Li, Minghai Qin, Fei Sun, Yufei Ding, and Yuan Xie. Shfl-bw: accelerating deep neural network inference with tensor-core aware weight pruning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1153–1158, 2022.

Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Joseph Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks. *Advances in Neural Information Processing Systems*, 34:21099–21111, 2021.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Ajay Kumar Jaiswal, Haoyu Ma, Tianlong Chen, Ying Ding, and Zhangyang Wang. Training your sparse neural network better with any mask. In *International Conference on Machine Learning*, pp. 9833–9844. PMLR, 2022.

Yu Ji, Ling Liang, Lei Deng, Youyang Zhang, Youhui Zhang, and Yuan Xie. Tetris: Tile-matching the tremendous irregular sparsity. *Advances in Neural Information Processing Systems*, 31, 2018.

Jiazhi Jiang, Dan Huang, Jiangsu Du, Yutong Lu, and Xiangke Liao. Optimizing small channel 3d convolution on gpu with tensor core. *Parallel Computing*, 113:102954, 2022.

Marc Jorda, Pedro Valero-Lara, and Antonio J Pena. Performance evaluation of cudnn convolution algorithms on nvidia volta gpus. *IEEE Access*, 7:70461–70473, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pp. 5544–5555. PMLR, 2020.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Qingyuan Li, Bo Zhang, and Xiangxiang Chu. Eapruning: Evolutionary pruning for vision transformers and cnns. *arXiv preprint arXiv:2210.00181*, 2022a.

Shigang Li, Kazuki Osawa, and Torsten Hoefler. Efficient quantized sparse matrix operations on tensor cores. *arXiv preprint arXiv:2209.06979*, 2022b.

Shiwei Liu, Tianlong Chen, Xiaohan Chen, Zahra Atashgahi, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. *Advances in Neural Information Processing Systems*, 34:9908–9922, 2021.

Yufan Liu, Jiajiong Cao, Bing Li, Weiming Hu, and Stephen Maybank. Learning to explore distillability and sparsability: a joint framework for model compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pp. 2736–2744, 2017.

Yucheng Lu, Shivani Agrawal, Suvinay Subramanian, Oleg Rybakov, Christopher De Sa, and Amir Yazdanbakhsh. Step: Learning n: M structured sparsity masks from scratch with precondition. *arXiv preprint arXiv:2302.01172*, 2023.

Xiaolong Ma, Minghai Qin, Fei Sun, Zejiang Hou, Kun Yuan, Yi Xu, Yanzhi Wang, Yen-Kuang Chen, Rong Jin, and Yuan Xie. Effective model sparsification by scheduled grow-and-prune methods. *arXiv preprint arXiv:2106.09857*, 2021.

Fanxu Meng, Hao Cheng, Ke Li, Huixiang Luo, Xiaowei Guo, Guangming Lu, and Xing Sun. Pruning filter in filter. *Advances in Neural Information Processing Systems*, 33:17629–17640, 2020.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pp. 4646–4655. PMLR, 2019.

Jinhyuk Park and Albert No. Prune your model before distill it. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI*, pp. 120–136. Springer, 2022.

Alexandra Peste, Eugenia Iofinova, Adrian Vladu, and Dan Alistarh. Ac/dc: Alternating compressed/decompressed training of deep neural networks. *Advances in Neural Information Processing Systems*, 34:8557–8570, 2021.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *IEEE*, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Wei Sun, Aojun Zhou, Sander Stuijk, Rob Wijnhoven, Andrew O Nelson, Henk Corporaal, et al. Dominosearch: Find layer-wise fine-grained n: M sparse schemes from dense neural networks. *Advances in neural information processing systems*, 34:20721–20732, 2021.

Kai Sheng Tai, Taipeng Tian, and Ser-Nam Lim. Spartan: Differentiable sparsity via regularized transportation. *arXiv preprint arXiv:2205.14107*, 2022.

Yijun Tan, Kai Han, Kang Zhao, Xianzhi Yu, Zidong Du, Yunji Chen, Yunhe Wang, and Jun Yao. Accelerating sparse convolution with column vector-wise sparsity. In *Advances in Neural Information Processing Systems*, 2022.

Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chunjing Xu, Chao Xu, and Chang Xu. Scop: Scientific control for reliable neural network pruning. *Advances in Neural Information Processing Systems*, 33:10936–10947, 2020.

Qinglin Wang, Dongsheng Li, Xiandong Huang, Siqi Shen, Songzhu Mei, and Jie Liu. Optimizing fft-based convolution on armv8 multi-core cpus. In *Euro-Par 2020: Parallel Processing: 26th International Conference on Parallel and Distributed Computing, Warsaw, Poland, August 24–28, 2020, Proceedings*, pp. 248–262. Springer, 2020.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang, and Lanshun Nie. Balanced sparsity for efficient dnn inference on gpu. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5676–5683, 2019.

Yuxin Zhang, Mingbao Lin, Chia-Wen Lin, Jie Chen, Yongjian Wu, Yonghong Tian, and Rongrong Ji. Carrying out cnn channel pruning in a white box. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Yuxin Zhang, Yiting Luo, Mingbao Lin, Yunshan Zhong, Jingjing Xie, Fei Chao, and Rongrong Ji. Bidirectional masks for efficient n: M sparse training. *arXiv preprint arXiv:2302.06058*, 2023.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n: M fine-grained structured sparse neural networks from scratch. *arXiv preprint arXiv:2102.04010*, 2021a.

Yangjie Zhou, Mengtian Yang, Cong Guo, Jingwen Leng, Yun Liang, Quan Chen, Minyi Guo, and Yuhao Zhu. Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 214–225. IEEE, 2021b.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.