WINOGRAD STRUCTURED PRUNING FOR FAST WINOGRAD CONVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Convolutional Neural Networks (CNNs) are computationally intensive, which limits deployment into mobile devices. To minimize operation counts in CNNs, pruning optimization techniques and Winograd's minimal filtering algorithm are widely used; however, the benefit of pruning disappears when both optimizations are simply applied together in CNN. To take full advantage of both approaches, two previous pruning methods were proposed: one is to apply pruning after kernel transformation, and the other is applying filter pruning on Winograd convolution. Unfortunately, the first method is hardware-unfriendly and the second approach suffers from a significant loss of accuracy. Thus, we propose structured pruning method specialized for Winograd convolution, that maximizes the hardware utilization by considering the conversion algorithm of parallel processors. We analyze the conversion algorithm of Winograd convolution on parallel processing units; then, we prune the weights in the Winograd-domain in a structured form with optimized pruning unit size, which maximizes the parallelism of the hardware while minimizing the loss of accuracy. For VGG-16 on the ImageNet dataset, the inference time of our method is 1.84 and 2.89 times better than previous two pruning methods with less than 1% accuracy loss.

1 INTRODUCTION

Deep Convolutional Neural Networks (CNNs) achieve high accuracy, significantly affecting computer vision (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015). Unfortunately, CNN inference is known to require a large number of computing resources to meet the target computing time constraints, as CNNs are composed of a huge number of layers to ensure high representation capabilities. Therefore, many prior works proposed methods to minimize the computations, leading to fewer computing resources to meet the target time (Han et al., 2016b). Especially pruning (Han et al., 2015) and Winograd Convolution (WC) (Lavin & Gray, 2016) have the most powerful results even in NVIDIA GPU and Intel CPU, which are the most frequently used general-purpose computing devices (Chetlur et al., 2014; Choquette & Gandhi, 2020; oneDNN, 2021). First, the pruning method speeds up CNN by eliminating redundant parameters (zero value), mitigating high memory usage and a large number of computations (Han et al., 2015). Second, to accelerate inference time, WC reduces the number of multiplications of Standard Convolution (SC) by applying Winograd's minimal algorithm (Winograd, 1980). The inference time of WC is 1.4 times faster on average compared to SC in ResNet-18 with negligible accuracy drop (Yan et al., 2020).

Even though the pruning and WC have significant performance gains respectively, some fine-grained pruning methods are challenging to combine with WC. To be specific, if the pruning unit size is smaller than the size of filter transformation (FTrans), FTrans removes the weight sparsity gained from pruning (Liu et al., 2018). To overcome the above problem, Winograd Weight Pruning (WWP) (Liu et al., 2018; 2017) suggests applying element-wise pruning after FTrans, leading to low accuracy drops even in high pruning ratio. Nevertheless, the previous research imposes two limitations: irregular data access patterns and no customized hardware kernels. First, WWP produces irregular data access patterns, which is inappropriate in parallel processors. Irregular data access patterns bring significant index computation overhead to parallel processors (Yu et al., 2017). Therefore, pruned models with irregular data access patterns are slower than dense models in GPUs (Greathouse & Daga, 2014). Second, WWP is hard to accelerate inference in GPUs since there is no customized hardware kernel for WWP acceleration.

Whereas, Filter Pruning (FP) (Li et al., 2016; He et al., 2018), which prunes filter unit in SC, has no side effect of eliminating sparsity when being applied after FTrans since the pruning unit size of FP is bigger than FTrans process unit. Therefore, FP can be directly combined with WC without modifying the algorithm, unlike WWP. FP brings regular weight data patterns in both SC and WC. This structured form can be executed using existing well-optimized hardware kernels without additional index computation in parallel processors, thus FP can speed up inference time in proportion to the pruning ratio. However, the pruning unit size of FP is too large, depending on the layer, causing a loss of representation ability on a high pruning ratio (Mao et al., 2017). As a result, FP drops accuracy since FP simultaneously eliminates a bunch of parameters, which indicates FP is not the appropriate approach for high pruning ratio.

The two previous pruning methods limit to accelerate WC with minimizing accuracy drop. To accelerate WC with tolerable accuracy drop by pruning, two conditions need to be satisfied: maintaining regular data access pattern and minimizing pruning unit size. First, the pruning needs to result in a regular data pattern that is suitable for parallel hardware execution. Second, to minimize the accuracy drop on high pruning ratio, the pruning unit size should be as small as possible while maintaining regular form of the pruned model (Mao et al., 2017). However, FP and WWP are biased to satisfy either side of the conditions. We analyze the computational properties (i.e. conversion algorithm (Yan et al., 2020; Jia et al., 2020)) of WC on the parallel processor to propose novel pruning that satisfies both conditions. The conversion algorithm is a process of converting the hardware-unfriendly computation part of WC, which has low data reuse, to Batched General Matrix Multiplication (BGEMM). We propose a Winograd Structured Pruning (WSP), which applies structured pruning to BGEMM matrices individually. The WSP pruned model has a structural form for each matrix of BGEMM. Also, the height and width sizes of the BGEMM matrix are the number of channels or filters on the convolutional layer, respectively. Therefore, the pruning unit size of the WSP is significantly smaller than FP's, resulting in high accuracy. Moreover, our WSP is scalable and can be directly adopted to current neural network eco-system, without any hardware modification.

Our proposed WSP can be divided into WSP-R and WSP-C according to row-wise or column-wise pruning. We conduct various experiments on WSP-R and WSP-C in the experiments section. The pruning unit size of WSP-R is twice as small or equal to WSP-C, so WSP-R has less accuracy drop than WSP-C at the same pruning ratio. Unlike WSP-C, WSP-R does not require index computation. Therefore, we conclude that WSP-R is better than WSP-C in accelerating CNN inference on parallel processors. Additionally, we experiment with NVIDIA GPU for further precise comparisons with WSP. While maintaining representation ability ($\sim 1\%$) in VGG-16 on ImageNet, WSP-R is $2.89\times$ and $1.84\times$ faster than WWP and FP, respectively, on GPU.

2 Preliminary: Winograd Convolution

Prerequisites Given n filters of l^{th} convolution layer as $\mathcal{F}^{(l)} \in \mathbb{R}^{n\times c\times k_h\times k_w}$, where n, c, k_h, and k_w are the number of filters, the number of channels, height, and width of the filters, respectively. The H and W denotes the height and width of the input feature map.

2.1 Process of Winograd Convolution

As shown in Figure 1 and equation (1), there are five steps in WC: Input Transformation (ITrans), Filter Transformation (FTrans), Element-Wise Matrix Multiplcation (EWMM), Channel-wise Summation (CS), and Output Transformation (OTrans). The $p \times p$ input patch (d), which is the basic block of WC, is extracted with stride of $(p - 2) \times (p - 2)$ from input feature map. The input feature map has ip number of input patches. Considering of padding, the $p \times p$ input patch is convolved with a 3 filter (g) to produce an $(p - 2) \times (p - 2)$ output patch (S) (Liu et al., 2018).

ITrans and FTrans transform spatial-domain input patch (d) and filter (g) to Winograd-domain input patch ($B^T dB$) and filter (GgG^T) using matrices B and G, respectively. After ITrans and FTrans, there is EWMM ($\odot$) between $B^T dB$ and GgG^T. CS performs channel-by-channel summation to reduce the channel of EWMM output. OTrans transforms from Winograd-domain output patch to spatial-domain output patch S by using matrix A. The output patches are assembled into an

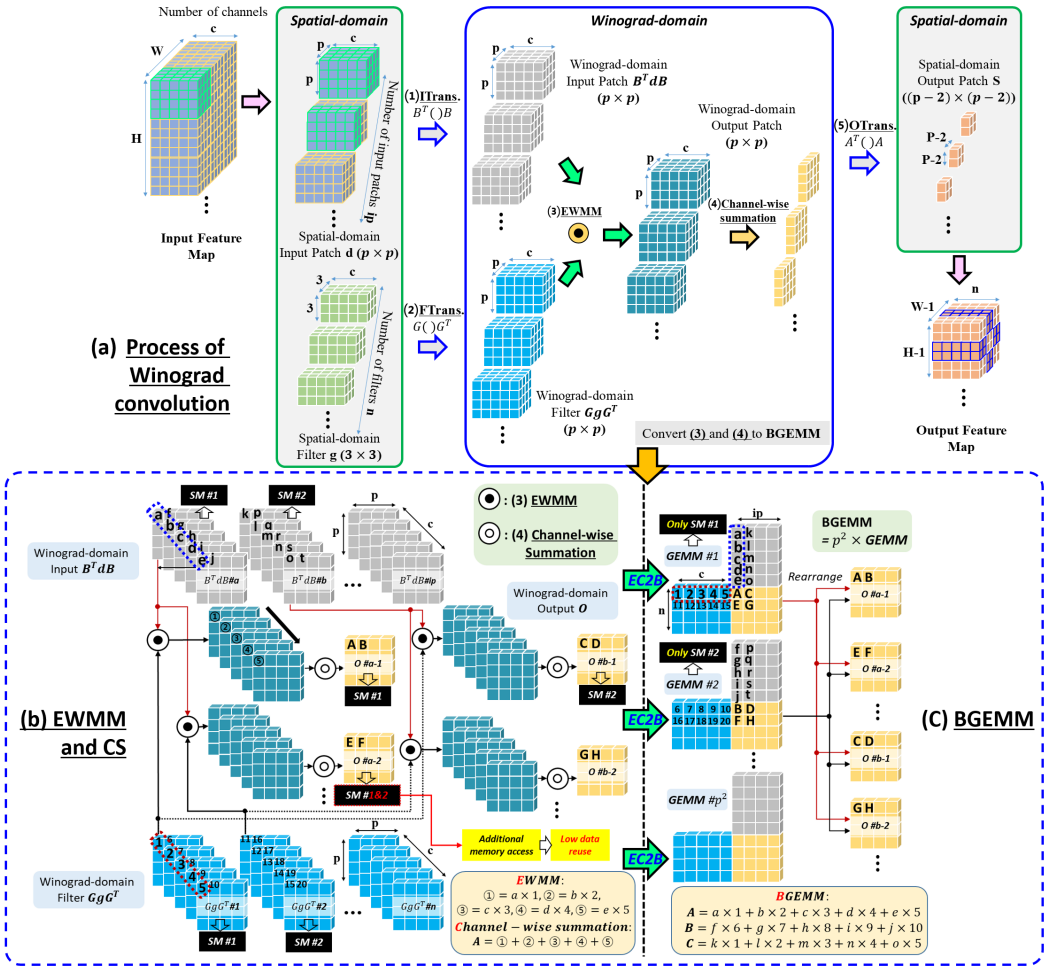


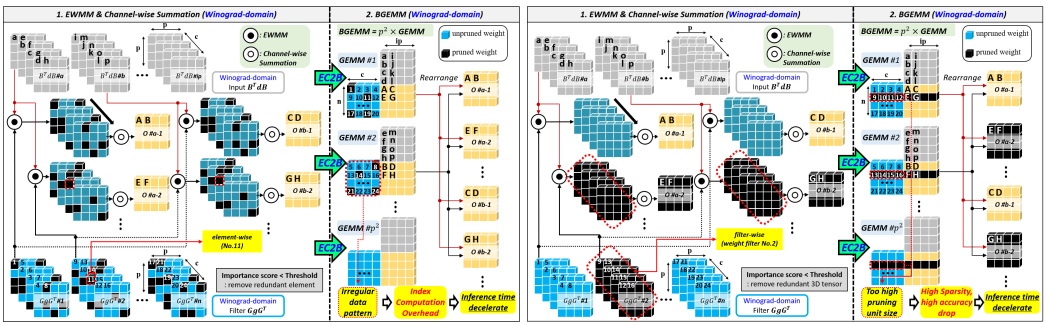
Figure 1: Overview of Winograd Convolution and Conversion from EWMM to BGEMM. (a) Process of Winograd convolution. (b) EWMM and CS. (C) BGEMM. SM denotes NVIDIA GPU SM, which is processing module.

output feature map. The matrices B, G and A used in ITrans, FTrans, and OTrans are made from Winograd's minimal algorithm.

$$S = A^T[(GgG^T) \odot (B^T dB)]A \tag{1}$$

2.2 Conversion for Winograd Convolution

Inefficient process of Winograd convolution According to ARM, EWMM accounts for most of the inference time of WC, so EWMM is a target for acceleration. However, since the EWMM results in an inefficient operation on the GPU, the parallel acceleration of the WC is not sufficient. GPU consists of dozens of independent functional modules called Streaming Multiprocessors (SMs) for parallel processing, and during EWMM on GPUs, 3D input patch ($B^T dB$) and a 3D filter (GgG^T) are loaded and processed in a single SM. As illustrated in Figure 1, to calculate filter GgG^T $\#1$ and input $B^T dB$ $\#a$, element $\#1$ of filter GgG^T $\#1$ is calculated only once with element $\#a$ of input patch $B^T dB$ $\#a$ and is never used again in the SM. Element $\#1$ of filter GgG^T $\#1$ is not reused within the SM, so the required data (element $\#k$ of input patch $B^T dB$ $\#b$) must be obtained from another SM. However, overhead occurs when the SMs exchange data because the memory of each SM is independent.



(a) Winograd Weight Pruning (b) Filter Pruning on Winograd-domain

Figure 2: Previous pruning methods for Winograd convolution.

Efficient Winograd convolution on parallel processors In order to solve the low computing intensity and frequent memory access overhead of EWMM, NVIDIA and ARM convert and process WC's EWMM and CS with BGEMM operations. The BGEMM operation is a structure that maximizes the data reuse of the GPUs. We call this covert process as EC2B (Conversion from EWMM and CS to BGEMM). As shown in Figure 1, BGEMM consists of a total of p^2 with GEMM operations with the same shape. The GEMM of BGEMM is a matrix product operation between the weight matrix of ($\mathbb{R}^{n \times c}$) shape and the input matrix of ($\mathbb{R}^{c \times k}$) shape. For parallelism, one GEMM is assigned to each SM. In $GEMM\#1$, the weight vector ($\#1 \sim \#5$) and the input vector ($\#a \sim \#e$) generate output elements ($\#A$) by performing MAC operations at once. The BGEMM rearranges the weight vectors ($\#11 \sim \#15$) just below the weight vectors ($\#1 \sim \#5$) in the same $GEMM\#1$ to increase the reuse of the input vectors ($\#a \sim \#e$). Thus, as shown in Figure 1, the input vector and weight vector are converted from EWMM to BGEMM so that they can be reused as much as possible within one SM.

3 Related Work

Winograd Convolution and Pruning are challenging to be compatible, but the following two previous pruning methods are compatible: Winograd weight pruning and filter pruning.

Winograd Weight Pruning (WWP) Winograd Weight Pruning (WWP) removes redundant Winograd-domain weight by element to reduce the parameters of Winograd Convolutional Neural Networks (Figure 2(a)). Liu et al. (2017) proposed a pruning and re-training approach in the Winograd-domain for the first time. Li et al. (2017) presents the 90.0% sparsity for AlexNet's Winograd parameters with less than 0.1% accuracy loss in the large dataset. Winograd-ReLU CNN Pruning (Liu et al., 2018) also exploits dynamic sparsity due to activation function as well as sparsity due to weight pruning in Winograd-domain. Unfortunately, according to the Yu et al. (2019), Winograd-domain dynamic sparsity of Winograd-ReLU CNN pruning (Liu et al., 2018) requires additional changes in the network structure. All the WWP methods are element-level pruning, which creates an irregular data pattern in Winograd-domain. The irregular data pattern requires significant index computation when accessing the pruned weights. Multiple indices cause a low throughput of the GPU owing to the memory divergence (Kloosterman et al., 2015), which makes WWP require specialized hardware kernel, or a dedicated accelerator (Han et al., 2017; 2016a; Parashar et al., 2017) for practical use. Therefore, we apply hardware efficient sparsity scheme in Winograd Convolutional Neural Networks to reduce the overhead of index computation.

Filter Pruning (FP) on Winograd-domain Structured pruning typically removes rows (filter pruning) or columns (group pruning) in lowered weight matrices. According to the Yu et al. (2019), the pruning unit size of filter pruning (FP) is the entire 3D filter, so the sparsity of FP is not removed by Filter Transformation progressing in 2D units in Winograd Convolution (Figure 2(b)). Li et al. (2016) and Yu et al. (2018) also adopt $l1$-norm criterion that prunes redundant filters. Channel pruning (He et al., 2017) prunes channels through LASSO regression-based channel selection and least square reconstruction. A layer collapse cannot be evaded when FP is performed (Tanaka

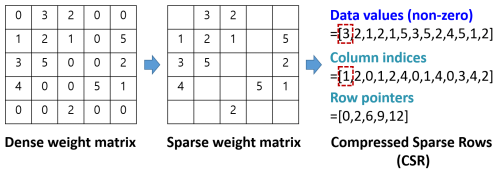


Figure 3: The CSR format use three 1-D arrays to store a matrix (Yu et al., 2017). Row pointer stores the pointer to the index of the beginning of each row.

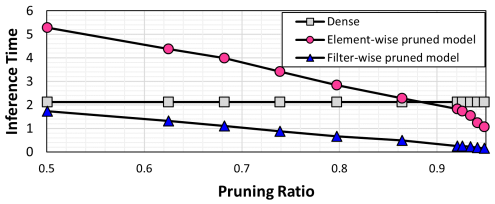


Figure 4: Comparison of inference time of matrix multiplication. The size of matix multiplication is $16384 \times 8196 \times 8$ (Yao et al., 2019).

et al., 2020). Since the pruned model of FP has a structural form in both standard convolution and Winograd convolution, FP has no index computation overhead. Thus, FP can accelerate Winograd convolution in proportion to the pruning ratio. However, FP loses a significant representation ability at a high pruning ratio because the pruning unit size that determines weight redundancy is too large. Therefore we propose WSP with less accuracy drop than FP at a high pruning ratio while maintaining structural data pattern by selecting fine-grained pruning unit size over FP in consideration of the WC computational structure on hardware.

4 Winograd Pruning

4.1 Problem of Conventional Winograd Pruning

Irregular data pattern problem of WWP As shown in Figure 2(a) and equation (2), since WWP is an element-wise pruning method, the WWP pruned model is an irregular data pattern before and after EC2B conversion. The irregular data pattern is not adequate for parallel processors such as GPUs (Greathouse & Daga, 2014). The irregular data pattern is calculated by converting weights into a sparse data format (e.g. CSR) to not calculate the zero values in hardware. In Figure 3, among the various sparse data formats, the representative CSR data format consists of two index arrays and one non-zero value array. When calculating sparse matrix multiplication using CSR, an index computation using two index arrays is required to select and obtain input data that operates with non-zero values. Then, the sparse matrix multiplication is performed with the input data and non-zero value array that are selected. Sparse matrix multiplication only computes non-zero values, so the amount of computation for values is less than dense matrix multiplication. However, in Figure 4, the overall inference time is slower than dense matrix multiplication since sparse matrix multiplication has an considerable index computation. As a result, WWP with irregular data pattern produces index computing overhead, so hardware support such as NPU, is essential for efficient use of WWP.

$$S = A^T[WeightPrune(GgG^T) \odot (B^T dB)]A \tag{2}$$

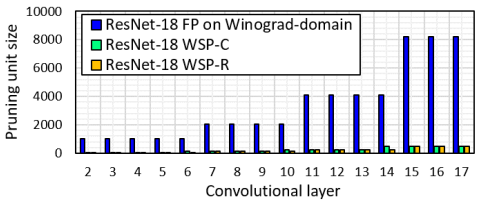


Figure 5: Comparison of pruning unit size of pruned model on ResNet-18 convolutional layer except convolutional 1 layer. The pruning unit size of FP on Winograd-domain is a $\mathbb{R}^{c \times p \times p}$.

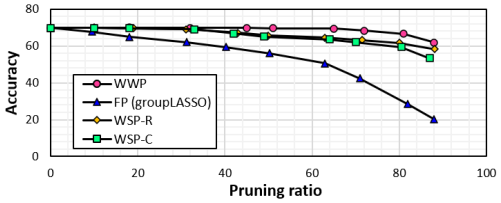


Figure 6: Comparison of accuracy and pruning ratio. Top-1 validation and pruning ratio for four model on a variation of ResNet-18 on ImageNet. The fine-tuning epoch is one.

Too high pruning unit size problem of FP In Figure 2(b) and equation (3), FP has a regular pattern in both before and after EC2B conversion. The regular structure rarely requires index computation. Therefore, the FP pruned model is computable using the existing hardware kernel as it is, and there is no increase in inference time due to additional operations. However, the effect of FP is not appreciable, because it is limited to pruning ratio. As shown in Figure 2(b), the pruning unit size of FP on Winograd-domain is $\mathbb{R}^{c\times p\times p}$, which size is too large to easily destroy the representative of weights, that makes hard to maintain accuracy in intensive pruning. For example, on ResNet-18, the pruning unit size of FP is 1024, 2048, 4096, and 8192 parameters depending on the layer (Figure 5). In Figure 6, the accuracy drop of VGG-16 remains within 1%, only up to 10% pruning ratio. Therefore, although FP is hardware-friendly structured, and even re-training is applied, FP has a significant accuracy drop in the high pruning ratio. Therefore, structured pruning with high pruning ratio needs to be applied to WC.

$$S = A^T[(G(FilterPrune(g))G^T) \odot (B^T dB)]A \tag{3}$$

4.2 Winograd Structured Pruning (WSP)

We introduce Winograd structured pruning (WSP), that satisfies hardware-friendly regularity and high pruning ratio. We apply structured pruning in Winograd-domain as shown in equation (4) and Figure 7. Structured pruning with optimized pruning unit and format solves the problem of irregular computation overhead of weight pruning and low pruning ratio of filter pruning. Structured pruning only requires a small amount of index computation, which is a negligible portion of hardware computation, which does not require any hardware change.

$$S = A^T[StructuredPrune(GgG^T) \odot (B^T dB)]A \tag{4}$$

We first analyze the operation flow of Winograd convolution, to present new pruning method that is specialized for Winograd convolution. As explained in subsection 2.2, the EWMM operation which accounts for the largest computation time among the Winograd convolution, has less data reuse during parallel processing. Hardware utilization decreases for low data reuse, and thus EWMM increases the total computation time. In NVIDIA GPUs and ARM CPUs, the EWMM operation of the Winograd convolution is converted to a BGEMM operation that has relatively high data reuse, as described in subsection 2.2. In Figure 1, BGEMM operation consists of p-square of sub matrix multiplication (sub-MM) operations.

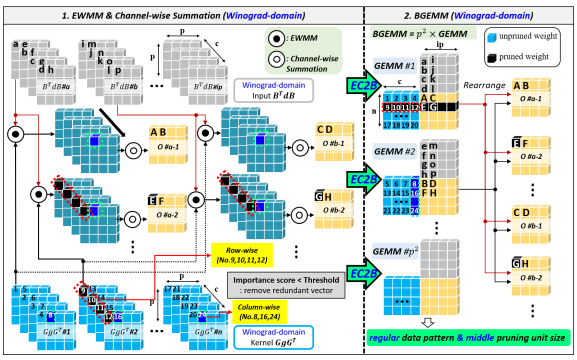


Figure 7: Overview of WSP

As explained in subsection 4.1, pruned model of WWP has irregular data pattern, since weights are pruned randomly in element units in all sub-MM weights in E2CB conversion. Therefore, WWP results in significant index computation. Unlike WWP, our WSP prunes vector-wise in every sub-MM weight, to achieve regular weight structure in Winograd-domain. The WSP uses the groupLASSO method to obtain the representative value of the vector, and if the absolute value of the representative value is less than the threshold, weight vector is removed. Figure 2(b) and 7 show that FP simultaneously removes vectors of the same position for all sub-MMs, but our WSP removes only one vector of sub-MMs. Therefore, WSP maintains regularity while pruning in unit size that is less than FP, so WSP results in high compression ratio with tolerable accuracy drop. Each structured sub-MM is processed in parallel with very little additional operation. We propose two WSP techniques: WSP-R pruning on a row vector basis and WSP-C pruning on a column vector basis.

WSP-R and WSP-C are defined as below.

- WSP-R: WSP-R prunes elements in a group unit within the same input channel before the conversion process called EC2B. In WSP-R, one row vector ($\mathbb{R}^{1\times c}$) among several sub-MM weights of BGEMM after EC2B is a pruning unit.
- *W*SP-C: WSP-C prunes elements in a group unit within the same output channel before the conversion process called EC2B. In WSP-C, one column vector ($\mathbb{R}^{n\times 1}$) among several sub-MM weights of BGEMM after EC2B is a pruning unit.

Training In WSP, we use pre-trained model provided by PyTorch. WSP is conducted with pre-trained model transformed by FTrans. Our WSP also applies pruning after FTrans like the existing WWP, so the sparsity does not disappear by FTrans (Liu et al., 2017; 2018).

Pruning ratio WSP calculates the threshold t by setting the desired pruning ratio. All weight groups whose absolute value of groupLASSO is less than t are set to 0 to prune the filter (Wen et al., 2016). WSP uses the same pruning ratio for all layers, but, in the case of the first layer, the pruning sensitivity was high, so it is excluded from the application of WSP.

Acceleration We measure the inference time of our proposed WSP using the BGEMM kernel provided by the NVIDIA CUTLASS library (Kerr et al., 2017). Since CUTLASS is an open-source CUDA C++ templates library providing customizable GEMM templates, we modify the BGEMM kernel of CUTLASS for WSP-C. The customized code processes index computation first, producing a dense BGEMM. WSP-C prunes by vector size, so the index computation of WSP is n or c times less than WSP, which is an element-wise method. On the other hand, the GPU kernel for WSP-R does not need customizing.

5 EXPERIMENTS

Datasets and Models We evaluate the performance of Winograd Structured Pruning using ImageNet dataset (Deng et al., 2009). ImageNet contains $1.28M$ training images and $50K$ validation images for 1000 object classes. For the validation of image classification, we assess our method with CNN models: VGG-16 (Simonyan & Zisserman, 2015) and ResNet-18 (He et al., 2016).

Baseline Settings We evaluate WSP using NVIDIA RTX 2080 TI GPUs. In our experiments, we use the pre-trained CNN models from Pytorch framework (Paszke et al., 2019). We perform fine-tuning with only 90 epochs after conducting pruning methods on ImageNet. For settings of the fine-tuning, we use SGD optimizer with the weight decay, 1×10^{-4} and the momentum as 0.9. We set a batch size of 256. The initial learning rate is set to 0.1 and divide it by 10 every 30 epochs.

5.1 COMPARISON OF WWP AND FP

We compare our proposed WSP with existing Winograd prunings in terms of accuracy drop, pruning ratio, and speedup. First, looking at Figure 8, FP is faster than the dense model in most pruning ratios because FP is a structured form after Winograd transformation, as described in section 3. However, the pruning unit size is too large to avoid indiscreet pruning because FP creates a structured form without considering the Winograd convolution. Due to FP's large pruning unit size, the accuracy drop exceeds 1% in FP from 30% pruning ratio. Therefore, FP is limited in Winograd applications requiring high accuracy. Second, WWP is a suitable method for accuracy only because even if the pruning ratio is increased to 90%, the accuracy drop is within 1%. However, despite pruning up to 90% pruning ratio, it is slower than the dense model due to the significant index computation of WWP, and there is a limitation that there should be additional hardware support. Finally, as shown in Figure 8, the WSP has an accuracy drop of almost 1% at 70% pruning ratio and is about 2.11 times faster than the dense model. Therefore, the WSP is suitable for the pruning method that satisfies all three aspects at the same time.

5.2 COMPARISON OF SOTA PRUNING

The pruning unit size of WSP-C is a $\mathbb{R}^{n\times 1}$. However, the pruning unit size of FP on Winograd-domain is $\mathbb{R}^{c\times p\times p}$. Since n and c are the same sizes in most layers in ResNet-18, the pruning unit

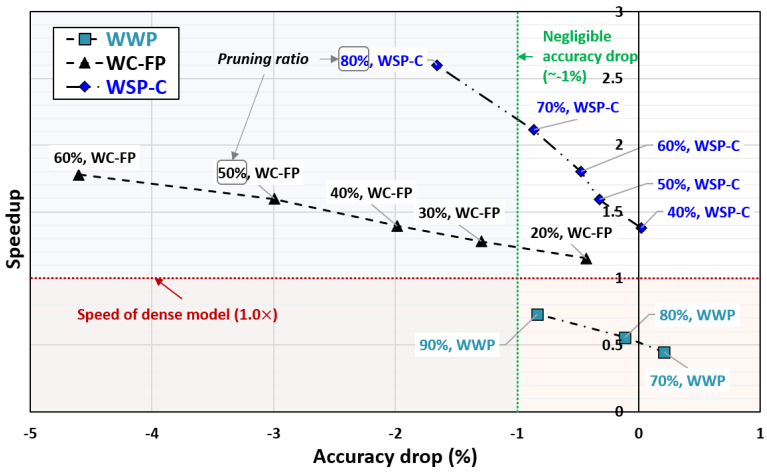


Figure 8: Comparison of speedup and accuracy with WWP and FP (groupLASSO) of VGG-16 on ImageNet (Deng et al., 2009). WC-FP denote FP on Winograd convolution. We simply accelerate WSP using CSRMM on cuSPARSE library (Naumov et al., 2010). Additionally, WSP can be slightly faster if customized GPU kernel for WSP is provided in future research.

Model	Pruning ratio	Top-1↓	Pruning unit size	Inference time	Speedup
ResNet-18	Dense	-	-	7.270ms	1.00×
Winograd Convolution	Dense	-	-	5.229ms	1.39×
TAS (Dong & Yang, 2019)	33.3%	-1.50%	$\mathbb{R}^{c\times p\times p}$	3.532ms	2.06×
LCCL (Dong et al., 2017)	34.6%	-3.65%	$\mathbb{R}^{c\times p\times p}$	3.439ms	2.11×
WSP-C (Ours)	**40.1%**	**-0.32%**	$\mathbb{R}^{n\times 1}$	3.389ms	**2.14**×
SFP (He et al., 2018)	41.7%	-3.18%	$\mathbb{R}^{c\times p\times p}$	3.094ms	2.35×
FPGM (He et al., 2019)	41.7%	-1.87%	$\mathbb{R}^{c\times p\times p}$	3.094ms	2.35×
WSP-C (Ours)	**60.7%**	**-1.04%**	$\mathbb{R}^{n\times 1}$	2.357ms	**3.08**×
WSP-C (Ours)	**70.1%**	**-1.49%**	$\mathbb{R}^{n\times 1}$	1.864ms	**3.90**×

Table 1: Comparison of inference time (ms) with filter pruning (FP) on Winograd-domain of ResNet-18 on ImageNet (Deng et al., 2009). All pruned models work with Winograd convolution (Lavin & Gray, 2016). Since WSP can prune only 3 convolutional layers, WSP-C performs filter pruning (groupLASSO) on the FC layer. Top-1↓ denotes accuracy drop.

size of WSP-C is about $p \times p$ times smaller than that of FP's. Therefore, WSP-C can prune more delicately than FP. In table 1, When the pruning ratio of LCCL (Dong et al., 2017) and SFP (He et al., 2018) is 34.6% and 41.7%, respectively, the top-1 accuracy drop is more than 3%. On the other hand, WSP-C's accuracy drop is only less than 2% in 70% pruning ratio. Since both WSP-C and FP have a pruned model as structured form, both prunings improve performance in proportion to the pruning ratio. In 60% pruning ratio, WSP-C has an accuracy drop of about 1% and 3.08 times faster inference time than the ResNet-18, simultaneously.

5.3 Per-layer Speedup Comparison

In GPUs, the speedup increases as the pruning ratio increases in each convolutional layer (Conv). However, since the number and structure of parameters are different for each layer, the speedup ratio according to the pruning ratio is different. As shown in Figure 9, WSP-C has a faster inference time than WSP-R from Conv 2 to Conv 5 layers in VGG-16 on GPU. Conv2 to Conv5 have a smaller

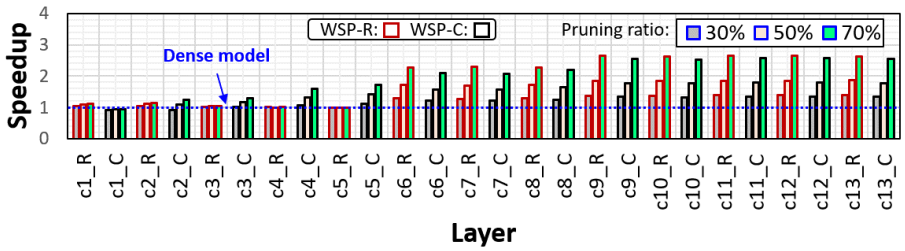


Figure 9: Comparison of speedup than dense VGG-16 model on ImageNet with WSP on GPU. c denotes convolutional layer. C and R denotes WSP-C and WSP-R, respectively.

parameter size of the layer than the other layers. Thus, it can be executed by assigning most Conv layer operation threads to the GPU cores at once, so pruning, which reduces the number of threads, such as WSP-R, is hardly effective in Conv layers with a small number of parameters of weight. On the other hand, from Conv 6 to Conv 13, the number of parameters is more than $600k$, which means that the number of GPU threads required is also high, so the WSP-R speedup ratio is higher than that of WSP-C. Additionally, unlike WSP-R in GPUs, WSP-C requires an index computing process that requires additional index loading, but WSP-R does not have an index computing overhead, so the speedup ratio is higher from Conv 6 to Conv 13 than WSP-C.

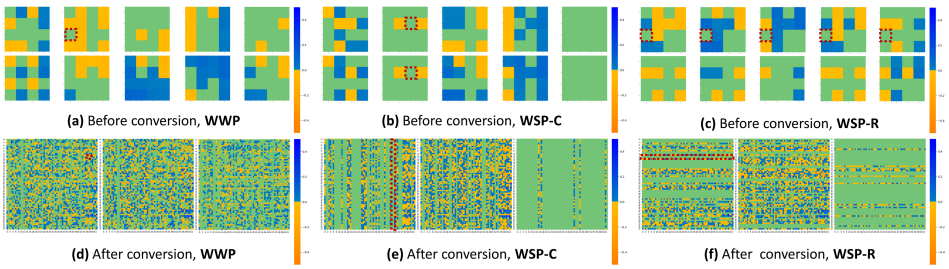


Figure 10: Filter visualizations in res2a_2a layer of ResNet-18 on ImageNet. Positive, negative and pruned weights are in blue, yellow and green respectively. Dotted redline denotes a pruning unit.

5.4 Filter Visualization

We use a visualization method to understand that WSP has a regular data pattern and middle pruning unit size at Winograd convolution. In Figure 10, we sequentially visualized WWP (Liu et al., 2018), WSP-C, and WSP-R. The filter visualization is done with the pruning ratio of 50% within the 1% variance. For visualization, we convert the EWMM 4D weight matrix of the WC to BGEMM weight matrix using EC2B converting method. The weight map of WWP shows irregular data patterns in both non-converted and converted weight matrice as shown in Figure 10(a) and 10(d). WSP-C has the same pruning mask between matrices with the same output channel as shown in Figure 10(b). In Figure 10(e), the WSP-C shows regular data patterns which have column-wise vector pruning unit at converted weight. In Figure 10(c), WSP-R has the same pruning mask between matrices with the same input channel. WSP-R has row-wise vector pruned data pattern as shown in Figure 10(f).

6 Conclusion

We propose Winograd structured pruning that accelerates Winograd convolution in consideration of Winograd convolution computational characteristics on a parallel processor. The EWMM, the core operation of Winograd convolution, is converted to BGEMM for execution in CPU and GPU. Since our WSP performs vector-wise pruning in BGEMM operation, it is possible to create a more delicate structured data pattern than WWP and FP. This efficient approach does not change the computational structure of BGEMM, so there is no additional computation overhead or hardware support. Moreover, even though WSP is a type of structured pruning, due to our optimized pruning unit size, our WSP brings a high pruning ratio for realistic CNN inference acceleration.

REFERENCES

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.

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.

Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. *arXiv preprint arXiv:1905.09717*, 2019.

Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5840–5848, 2017.

Joseph L Greathouse and Mayank Daga. Efficient sparse matrix-vector multiplication on gpus using the csr storage format. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 769–780. IEEE, 2014.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016a.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016b.

Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 75–84, 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, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4340–4349, 2019.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

Liancheng Jia, Yun Liang, Xiuhong Li, Liqiang Lu, and Shengen Yan. Enabling efficient fast convolution algorithms on gpus via megakernels. *IEEE Transactions on Computers*, 69(7):986–997, 2020.

JD Andrew Kerr, Duane Merrill, and J Tran. Cutlass: Fast linear algebra in cuda c++. *NVIDIA Developer Blog*, 2017.

John Kloosterman, Jonathan Beaumont, Mick Wollman, Ankit Sethia, Ron Dreslinski, Trevor Mudge, and Scott Mahlke. Warppool: Sharing requests with inter-warp coalescing for throughput processors. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 433–444. IEEE, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4013–4021, 2016.

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

Sheng Li, Jongsoo Park, and Ping Tak Peter Tang. Enabling sparse winograd convolution by native pruning. *arXiv preprint arXiv:1702.08597*, 2017.

Xingyu Liu, Song Han, Mao Huizi, and William J Dally. Efficient sparse-winograd convolutional neural networks. In *International Conference on Learning Representations (ICLR) Workshop*, 2017.

Xingyu Liu, Jeff Pool, Song Han, and William J Dally. Efficient sparse-winograd convolutional neural networks. In *International Conference on Learning Representations*, 2018.

Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the granularity of sparsity in convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 13–20, 2017.

M Naumov, LS Chien, P Vandermersch, and U Kapasi. Cusparse library. In *GPU Technology Conference*, 2010.

Intel oneDNN. https://github.com/oneapi-src/onednn. *https://github.com/oneapi-src/oneDNN*, 2021.

Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 45(2):27–40, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning\-library.pdf.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33:6377–6389, 2020.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pp. 2074–2082, 2016.

Shmuel Winograd. *Arithmetic complexity of computations*, volume 33. Siam, 1980.

Da Yan, Wei Wang, and Xiaowen Chu. Optimizing batched winograd convolution on gpus. In *Proceedings of the 25th ACM SIGPLAN symposium on principles and practice of parallel programming*, pp. 32–44, 2020.

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.

Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 548–560. IEEE, 2017.

Jiecao Yu, Jongsoo Park, and Maxim Naumov. Spatial-winograd pruning enabling sparse winograd convolution. *arXiv preprint arXiv:1901.02132*, 2019.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.