Appendix: Winograd Structured Pruning for Fast Winograd Convolution

A. Details of Background

A.1. Structure of Winograd Convolution

In equation 1, Winograd convolution consists of five steps: (1) Input Transformation (**ITrans**), (2) Filter Transformation (**FTrans**), (3) Element-wise Matrix Multiplication (**EWMM**), (4) **Channel-wise Summation**, and (5) Output Transformation (**OTrans**). As shown in Fig. 1, input patches (indicated by d) of the $p \times p$ shape are extracted from an input feature map of the $H \times W$. There are k number of input patches.

- (1) Input patches (d) are transformed using matrix B to be $B^T dB$ with $p \times p$ shape (Winograd-domain input patch).
- (2) Kernels (g) with $k_h \times k_w$ shape are transformed using matrix G to be GgG^T with $p \times p$ shape (Winograd-domain Kernel). For maximizing parallelism, k_h and k_w are generally three in Winograd convolution.
- (3) Winograd-domain output patch with $p \times p$ shape is obtained by element-wise multiplying $B^T dB$ and GgG^T.
- (4) Through summation of elements between channels, the channel of Winograd-domain output patch is reduced from c to 1.
- (5) After channel-wise summation in Winograd-domain, the output patch S with $(p-2) \times (p-2)$ shape is transformed using matrix A.

The matrices B, G and A are predefined by the Winograd algorithm [12]. The matrices have a shape of $p \times p$. When p is set to 4, B and G consist of only 1, -1, and 0. Thus, **ITrans** and **FTrans** requires only addition operations.

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

A.2. Convert EWMM and Channel-wise Summation to BGEMM

According to the ARM compute library [3], EWMM accounts for the majority of inference time in Winograd convolution, although it varies from layer to layer. However, GPUs cannot perform EWMM well, due to low computational intensity. Since one multiply and add operation requires three memory accesses (loads two operands and stores results) [4]. In Winograd-domain of Fig. 1, EWMM operation is the element-wise matrix multiplication of k input tensors ($\mathbb{R}^{c\times p\times p}$ shape) and n weight tensors ($\mathbb{R}^{c\times p\times p}$ shape). After EWMM, there is channel-wise summation. To describe more specifically, EWMM proceeds in two steps in the Winograd convolution (see Fig. 1). First, five elements $(1, 2, 3, 4, 5)$ located in the same height and width of the 3D weight tensor $GgG^T\#1$ and located in different channels perform EWMM with a, b, c, d, e of input 3D tensor $B^T dB\#a$, respectively. Second, the channel-wise summation executes, by adding ①, ②, ③, ④, and ⑤ located in different channels made from EWMM.

For efficient GPU execution, EWMM and Channel-wise Summation can convert to a BGEMM through transposition and reconstruction. We call this conversion, **EC2B** (EWMM and Channel-wise summation to BGEMM). After **EC2B**, BGEMM operates which are a combination of p^2 identical shapes of GEMM operations. The GEMM is a matrix multiplication of a weight matrix of $n \times c$ shape and an input matrix of $c \times k$ shape. The aforementioned two operation of EWMM are changed into one MAC (Multiply-ACcumulate) operation (equation 2). The results are same with EWMM operation + Channel-wise Summation, and BGEMM.

$$Output(A) = (a\times 1)+(b\times 2)+(c\times 3)+(d\times 4)+(e\times 5) \tag{2}$$

B. Details of Previous Work

We introduce two previously proposed pruning techniques, WWP and FP, which are compatible with existing Winograd convolution. However, as previously mentioned, WWP suffers from low performance, while FP results in low accuracy. In this section, we describe the limitations of WWP and FP in more detail.

B.1. Filter Pruning (FP)

Looking at Fig. 2, the FP pruned model after EC2B is still a structural matrix. Thus, since there is no additional index computation, the existing GPU kernel widely used in Winograd convolution can be applied. In Fig. 2, the pruning unit size of FP is $\mathbb{R}^{c\times p\times p}$ in Winograd convolution, which makes it hard to achieve high pruning ratio within 1% accuracy drop. FP does not support fine-grained structured pruning [9, 10], thereby we propose novel GPU-friendly and fine-grained pruning that considers the GPU computation method of Winograd convolution.

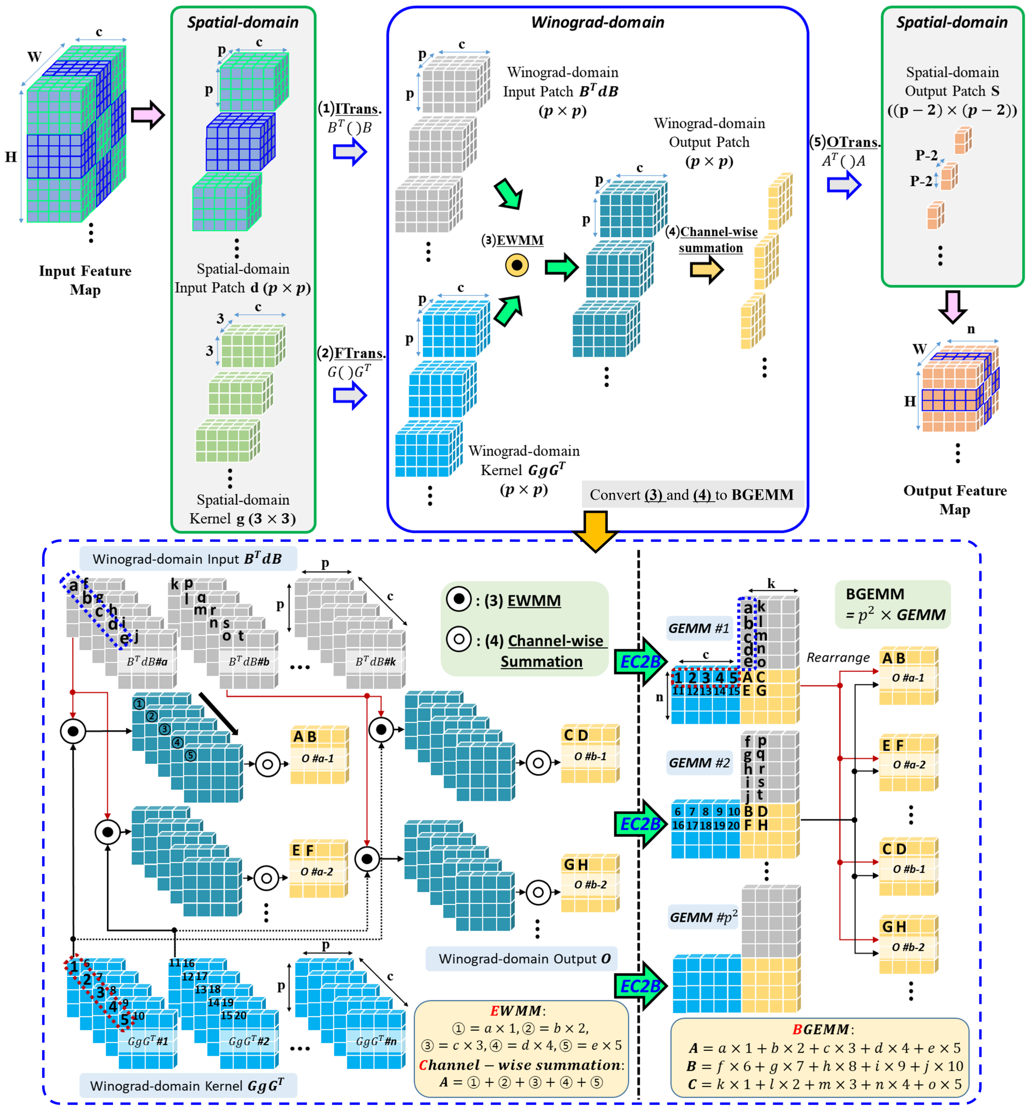
Spatial-domain
Winograd-domain
Spatial-domain
Input Feature Map
Spatial-domain Input Patch d (p × p)
Spatial-domain Kernel g (3 × 3)
(1)ITrans. B^T( )B
(2)FTrans. G( )G^T
Winograd-domain Input Patch B^T dB (p × p)
Winograd-domain Output Patch (p × p)
Winograd-domain Kernel GgG^T (p × p)
(3)EWMM
(4)Channel-wise summation
Convert (3) and (4) to BGEMM
(5)OTrans. A^T( )A
Spatial-domain Output Patch S ((p − 2) × (p − 2))
Output Feature Map
Winograd-domain Input B^T dB
Winograd-domain Kernel GgG^T
Winograd-domain Output O
: (3) EWMM
: (4) Channel-wise Summation
EC2B
GEMM #1
GEMM #2
GEMM #p^2
BGEMM = p^2 × GEMM
Rearrange
EWMM: ① = a × 1, ② = b × 2, ③ = c × 3, ④ = d × 4, ⑤ = e × 5
Channel − wise summation: A = ① + ② + ③ + ④ + ⑤
BGEMM: A = a × 1 + b × 2 + c × 3 + d × 4 + e × 5
B = f × 6 + g × 7 + h × 8 + i × 9 + j × 10
C = k × 1 + l × 2 + m × 3 + n × 4 + o × 5

Figure 1. Overview of Winograd Convolution (Top) & EC2B process (Bottom)

$$S = A^T[(GFilter\_Prune(g)G^T) \odot (B^T dB)]A \quad (3)$$

B.2. Winograd Weight Pruning (WWP)

WWP can prune elements ($\mathbb{R}$) from weights as long as they do not affect the accuracy of DNNs as presented in Equation 4. When the pruned models generated by WWP are executed in GPUs, the models are converted to BGEMM to maximize GPU throughput. However, such a conversion creates sparse matrices causing irregular data structures and making it harder for GPUs to load data from the global memory (see Fig. 3). Loading irregular data structures requires significant index computation time in GPUs

$$S = A^T[Weight\_Prune(GgG^T) \odot (B^T dB)]A \quad (4)$$

C. Details of WINS

We propose Winograd Structured Pruning (WINS). WINS performs pruning on the transformed weight (Winograd-domain weight) to address the compatibility issue between Winograd convolution and pruning. For GPU efficiency, Winograd convolution is converted into BGEMM operation. BGEMM consists of a combination of p^2 sub-matrix multiplication operations, which are identical shapes. The weight shape of sub-matrix multiplication is $n \times c$. As

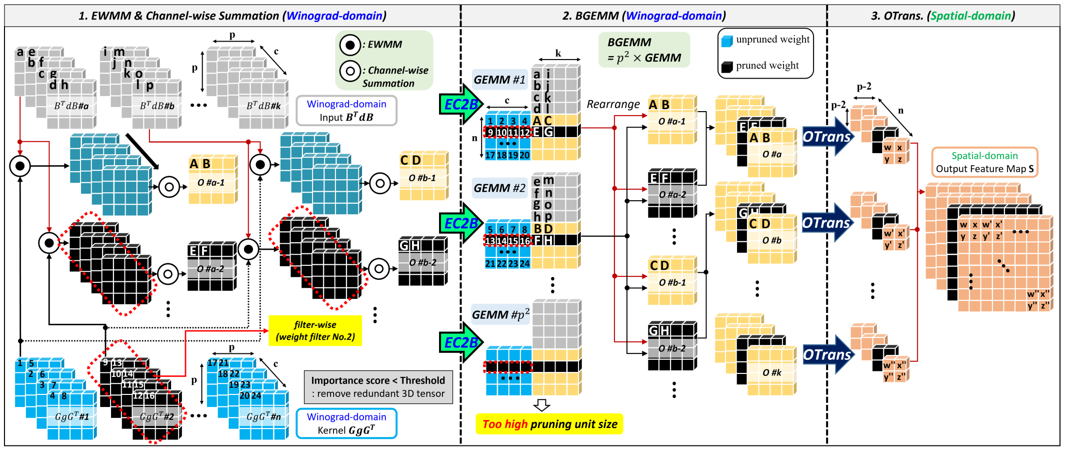
1. EWMM & Channel-wise Summation (Winograd-domain)
2. BGEMM (Winograd-domain)
3. OTrans. (Spatial-domain)
: EWMM
: Channel-wise Summation
Winograd-domain Input $B^T dB$
Winograd-domain Kernel GgG^T
BGEMM $= p^2 \times$ GEMM
unpruned weight
pruned weight
Rearrange
EC2B
OTrans
Spatial-domain Output Feature Map S
filter-wise (weight filter No.2)
Importance score < Threshold : remove redundant 3D tensor
Too high pruning unit size

Figure 2. Overview of Filter Pruning (FP) during Winograd convolution [5].

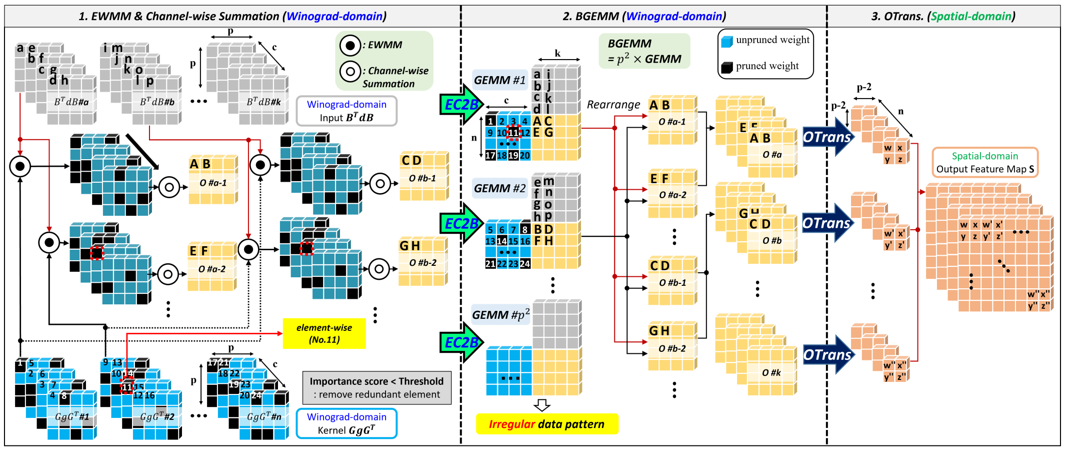
1. EWMM & Channel-wise Summation (Winograd-domain)
2. BGEMM (Winograd-domain)
3. OTrans. (Spatial-domain)
element-wise (No.11)
Importance score < Threshold : remove redundant element
Irregular data pattern

Figure 3. Overview of Winograd Weight Pruning (WWP) [8].

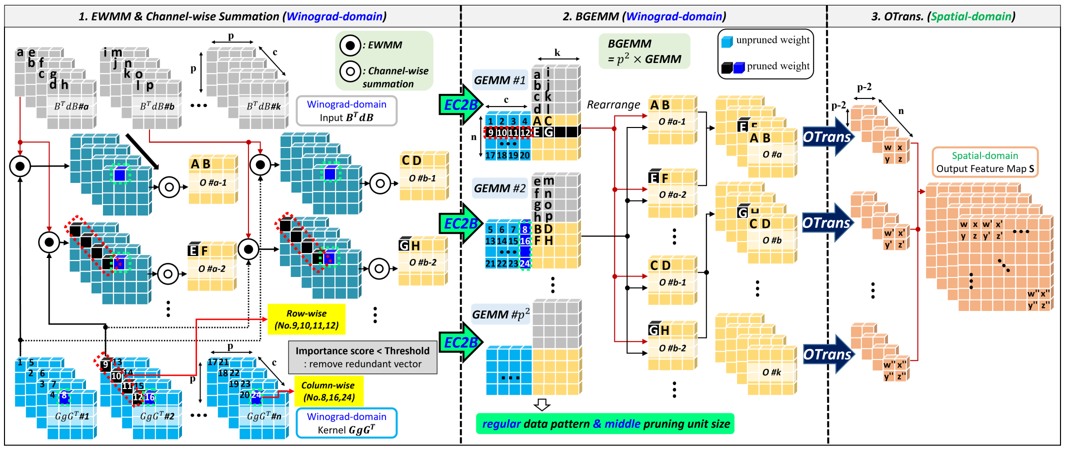
1. EWMM & Channel-wise Summation (Winograd-domain)
2. BGEMM (Winograd-domain)
3. OTrans. (Spatial-domain)
Row-wise (No.9,10,11,12)
Column-wise (No.8,16,24)
Importance score < Threshold : remove redundant vector
regular data pattern & middle pruning unit size

Figure 4. Overview of Winograd Structured Pruning (WINS).

shown in Fig. 4, WINS performs vector-wise pruning for each p^2 sub weight matrices, respectively. WINS groups each row vector and column vector in the sub weight matrix, and if the representative value of the vector is less than the threshold, it is considered redundant and is removed. WINS uses a representative value of the vector through

Model	Dataset	Method	Pruning Ratio (PR)	Baseline mAP	Pruned mAP
DETR [1] [Backbone: ResNet-50]	COCO [6]	BCBP [9]	50%	42.0%	40.9%
		WINS-AB	**50%**	**42.0%**	**41.2%**
SSD300 [7] [Backbone: VGG-16]	Pascal VOC [2] '07 + '12	BCBP [9]	30%	77.8%	76.7%
		WINS-AB	**30%**	**77.8%**	**77.3%**

Table 1. Experiments on object detection. Pruning refers solely to the backbone.

group LASSO [11]. Since the height and width of the sub weight matrix are n and c, the pruning unit size of WINS is n or c vector. The pruning unit size of FP is one 3D filter ($\mathbb{R}^{c\times p\times p}$). Therefore, WINS is fine-grained pruning version of FP.

Column-wise method (WINS-C) groups elements with different output channels (n) in the EC2B converted weight and processes them as one pruning unit size. The pruning unit of WINS-C is $\mathbb{R}^{n\times 1\times 1\times 1}$. After EC2B, the column vector of GEMM weight is pruned, as shown in Fig. 4.

Row-wise method (WINS-R) groups elements with different input channels (c) in the EC2B converted weight and processes them as one pruning unit size. The pruning unit of WINS-R is $\mathbb{R}^{c\times 1\times 1}$. After EC2B, the row vector of GEMM weight is pruned, as shown in Fig. 4.

$$S = A^T[Structured\_Prune(GgG^T) \odot (B^T dB)]A \quad (5)$$

C.1. Object Detection

As shown in Table. 1, we conducted experiments applying pruning on DETR and SSD300 on the COCO and Pascal VOC, respectively. First, for DETR, we applied pruning with retraining (300 epochs), and WINS-AB showed a mAP drop of 0.8% compared to the baseline. Second, for SSD300, we applied pruning with fine-tuning (5 epochs), and WINS-AB showed a mAP drop of 0.5% than baseline. In both models, WINS-AB has a smaller accuracy drop than BCBP [9] at the same pruning ratio.

References

[1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 4

[2] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010. 4

[3] Peter Harris. The mali gpu: An abstract machine, part 2-tilebased rendering, 2014. 1

[4] 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. 1

[5] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4013–4021, 2016. 3

[6] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. 4

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016. 4

[8] Xingyu Liu, Jeff Pool, Song Han, and William J Dally. Efficient sparse-winograd convolutional neural networks. In *International Conference on Learning Representations*, 2018. 3

[9] Cheonjun Park, Mincheol Park, Hyun Jae Oh, Minkyu Kim, Myung Kuk Yoon, Suhyun Kim, and Won Woo Ro. Balanced column-wise block pruning for maximizing gpu parallelism. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9398–9407, 2023. 1, 4

[10] Cheonjun Park, Mincheol Park, Hyunchan Moon, Myung Kuk Yoon, Seokjin Go, Suhyun Kim, and Won Woo Ro. Deprune: Depth-wise separable convolution pruning for maximizing gpu parallelism. *Advances in Neural Information Processing Systems*, 37:106906–106923, 2024. 1

[11] 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*, pages 2074–2082, 2016. 4

[12] Shmuel Winograd. *Arithmetic complexity of computations*. Siam, 1980. 1