POLITECNICO DI TORINO
Repository ISTITUZIONALE

Wino Vidi Vici: Conquering Numerical Instability of 8-Bit Winograd Convolution for Accurate Inference Acceleration on Edge

Original
Wino Vidi Vici: Conquering Numerical Instability of 8-Bit Winograd Convolution for Accurate Inference Acceleration on Edge / Mori, Pierpaolo; Frickenstein, Lukas; Balamuthu Sampath, Shambhavi; Thoma, Moritz; Fasfous, Nael; Rohit Vemparala, Manoj; Frickenstein, Alexander; Unger, Christian; Stechele, Walter; Mueller-Gritschneder, Daniel; Passerone, Claudio. - ELETTRONICO. - (2024), pp. 53-62. (Intervento presentato al convegno Winter Conference on Applications of Computer Vision (WACV)).

Availability:
This version is available at: 11583/2987510 since: 2024-04-02T19:02:10Z

Publisher:
IEEE

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2024 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

18 April 2024

Wino Vidi Vici: Conquering Numerical Instability of 8-bit Winograd Convolution for Accurate Inference Acceleration on Edge

Pierpaolo Mori[1,2,3], Lukas Frickenstein[2], Shambhavi Balamuthu Sampath[2], Moritz Thoma[2], Nael Fasfous[2], Manoj Rohit Vemparala[2], Alexander Frickenstein[2], Christian Unger[2], Walter Stechele[3], Daniel Mueller-Gritschneder[3], Claudio Passerone[1]

[1]Politecnico Di Torino, Turin, Italy; [2]BMW Group, Munich, Germany; [3]Technical University of Munich, Munich, Germany

{<firstname>.<lastname>}[1]@polito.it,[2]@bmw.de,[3]@tum.de

Abstract

Winograd-based convolution can reduce the total number of operations needed for convolutional neural network (CNN) inference on edge devices. Most edge hardware accelerators use low-precision, 8-bit integer arithmetic units to improve energy efficiency and latency. This makes CNN quantization a critical step before deploying the model on such an edge device. To extract the benefits of fast Winograd-based convolution and efficient integer quantization, the two approaches must be combined. Research has shown that the transform required to execute convolutions in the Winograd domain results in numerical instability and severe accuracy degradation when combined with quantization, making the two techniques incompatible on edge hardware. This paper proposes a novel training scheme to achieve efficient Winograd-accelerated, quantized CNNs. 8-bit quantization is applied to all the intermediate results of the Winograd convolution without sacrificing task-related accuracy. This is achieved by introducing clipping factors in the intermediate quantization stages as well as using the complex numerical system to improve the transform. We achieve 2.8× and 2.1× reduction in MAC operations on ResNet-20-CIFAR-10 and ResNet-18-ImageNet, respectively, with no accuracy degradation.

1. Introduction

Convolutional neural networks (CNNs) achieve remarkable results in many computer vision tasks making them the state-of-the-art (SotA) solution for a wide variety of applications. Over the last decade, CNNs have become more computationally complex, making their deployment highly challenging on edge. The main computational effort is due to large amounts of multiply-accumulate operations (MACs) required to compute the convolution operations.

To reduce their memory and computational footprint, quantization became a standard technique applied before deploying CNNs on edge devices [3]. Quantizing a network implies reducing the bit-width of its weights and activations to enable low-precision arithmetic on edge hardware. Moving from 32-bit floating-point representation to low-precision datatypes (INT8, INT4) reduces the complexity of hardware arithmetic units and the on-chip memory requirements. However, when the hardware architecture does not take advantage of lower-bit data representation (e.g. 8-bit hardware and 4-bit quantization), quantization does not bring any further performance improvement (NVIDIA Jatson Nano [1], Snapdragon 8 Gen 1 [4], Google Coral [2]). For such general-purpose hardware, where quantization below 8-bit is not supported, the latency of the CNN can be further reduced using fast algorithms such as Winograd-based convolution. The standard convolution operation strides the filters across an input activation tensor. Each stride in the standard convolution computes a single pixel in an output map. Alternatively, convolution operations can be transformed to the Winograd domain to reduce the number of MAC operations [21], resulting in faster inference on general-purpose hardware [25]. Winograd-based convolution produces tiles of output pixels in each step. For an $F(4,3)$ Winograd algorithm, a $4\times$ MAC reduction can be achieved. Winograd convolution consists of three stages: **(1)** inputs and weights transformation, **(2)** element-wise matrix multiplication (EWMM) of the transformed matrices and **(3)** inverse transformation to produce the spatial output feature maps. These three steps are visualized in Fig. 1. Theoretically, the transform is lossless and brings its benefits with a negligible degradation in full-precision arithmetic [7, 10]. This makes it popular in training and server settings [5, 6]. While floating-point Winograd algorithms have been improved to support new layer types [19], to adopt reuse schemes [18, 28] and to further reduce the computational complexity throughout pruning [23, 29], bringing the benefits of Winograd to edge de-

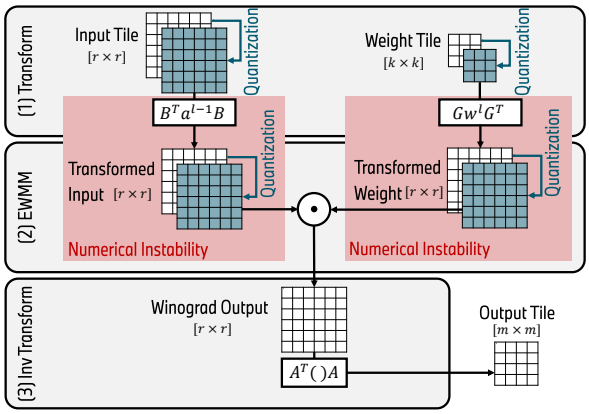


Figure 1. The three steps of the Winograd algorithm. Numerical instability due to quantization is highlighted.

vices with limited arithmetic precision reveals its *numerical instability* problem [16]. The transformation applied to execute the convolution in the Winograd domain causes numerical overflow by increasing the range of values in the resulting tiles. With the limited bit-width available on edge devices and the quantization of all the operands involved in all three aforementioned stages of the Winograd algorithm, large numerical errors are unavoidable. For example, a 16-bit standard convolution ResNet-18 achieving 92% accuracy on CIFAR-10, would degrade down to 19.25% when using 16-bit Winograd F(4,3) convolution [16]. This fundamental contradiction of numerical instability during the transform and limited bit-width on edge arithmetic units makes the Winograd algorithm incompatible with standard quantization techniques, severely degrading the accuracy of the CNNs. A lot of effort has been made to achieve accurate 8-bit inference with Winograd-based convolution. However, this goal has always been achieved by quantizing only a part of the Winograd algorithm (i.e. steps 1, 2, or 3), while allowing other parts to remain at higher bit-widths. For example, by quantizing only the EWMM, or by quantizing the transform and extending the bit-width for the EWMM, thereby incurring a computational or hardware overhead [13, 22, 24, 26].

In this work, we focus on efficient and accurate *full* 8-bit inference of CNNs adopting Winograd algorithms, without introducing any hardware overhead and minimizing accuracy degradation compared to their standard 8-bit convolution counterpart. Our core focus is to further improve the efficiency of CNNs on devices that already support 8-bit quantization [1, 2, 4]. We tackle the challenges of 8-bit quantized Winograd with the following contributions:

1. Introducing, for the first time, Winograd-based convolution using complex numbers during CNN training time, resulting in a full 8-bit $F(4,3)$ Winograd-based ResNet-20-CIFAR-10 with 2.78× MACs reduction and *no accuracy degradation*.

2. Introducing a trainable clipping factor for quantizing transformed parameters in the Winograd domain, resulting in a MACs reduction of 2.45× for ResNet-18-ImageNet with *only* ∼1 p.p. accuracy degradation.

3. After combining training with complex numerical representations (Contribution 1) with trainable clipping factors in the Winograd domain (Contribution 2) we achieve 2.11× reduction in operations on ResNet-18-ImageNet and 2.14× operation reduction for DeepLabV3+ on the CityScapes dataset, with *no prediction quality degradation*.

2. Related Work

2.1. Post-Train Winograd-based Quantized CNNs

Li et al. [22] resorted to the Winograd $F(4,3)$ algorithm to speed up inference time on CPUs. Full-precision transformations to/from the Winograd domain are used to minimize the accuracy degradation. Post-training quantization is then used to discretize the transformed weights and activations to 8-bit, performing only EWMM on 8-bit multipliers. The input and weights for all the convolutional layers are stored in 32-bit floating-point, demanding expensive computations for transformations and high memory bandwidth between two layers. Chikin et al. [13] propose to balance the data ranges of inputs and filters by scaling them channel-wise with balancing coefficients in order to equalize channel ranges to improve the quality of quantization. Furthermore, they also apply scaling factors per pixel within the tile to better map the transformed inputs/weights from floating-point to the quantized range. The channel balancing and channel-wise tile scaling factors increase the computational complexity of the transformation. Additionally, the pixel-wise scaling factors lead to the introduction of an expensive dequantization step before performing inverse transformation. Only the element-wise multiplication is quantized, similar to Li et al. [22], leading to huge hardware overhead.

Differently, our work realizes CNNs with full 8-bit Winograd-based convolutions and only needs *one scaling factor* for the transformations of one layer.

2.2. Winograd with Other Number Systems

Mattina et al. [24] leverage the residue number system to avoid accuracy degradation for 8-bit quantized CNNs. Winograd transformation matrices are converted to residue number system, turning hardware-friendly operations such as addition and shift into more costly integer multiplications. At several stages in the Winograd transformations, modulo operations are used, requiring dedicated hardware. For an 8-bit Winograd-based convolution in the residue

number system, the computation for $F(4,3)$ is triplicated, thereby eliminating the Winograd MAC operation savings. Their approach works efficiently for large tile sizes (e.g. $F(14,3)$), which is not feasible in modern CNNs demanding more hardware complexity for Winograd transformation stages. Meng et al. [26] propose to use complex interpolation points to compute the Winograd transformation matrices, leading to lower magnitude values for $F(4,3)$ algorithm which results in less severe numerical overflow. With hardware-friendly transformations, and integer-only arithmetic, numerical error is minimized, at the cost of a lower MAC savings compared to the standard $F(4,3)$ Winograd algorithm (from $4\times$ down to $3.13\times$). Although they explore the potential of complex Winograd transformations for quantized CNNs, they adopted 12-bit multipliers for the EWMM to avoid numerical errors, making the approach not suitable for common 8-bit edge accelerators.

In our approach we embed the complex-Winograd algorithm in the Winograd-aware training loop, maintaining 8-bit quantization for the Winograd-based convolution.

2.3. Winograd-Aware Training (WAT)

Marques et al. [16] first included the Winograd algorithm in the training loop, making the model aware of the numerical instability problem. The authors made the Winograd transformation matrices trainable to reduce the accuracy degradation caused by quantization overflow. They further proposed wiNAS to search for Winograd-aware quantized networks by deciding the optimal Winograd tile size for each convolutional layer. However, as transformation matrices are trainable and maintained in floating-point representation, the computational cost in edge deployment gets expensive. Different Winograd tiles in various layers demand additional hardware logic to implement required transformations. Barabasz [9] resorts to Legendre polynomials to tackle numerical error for 8-bit Winograd quantization. In this approach additional floating-point matrix multiplications are added along with standard Winograd transforms, increasing the complexity of the method. Moreover, the best accuracy values are achieved by making trainable Winograd transformation matrices, increasing the complexity during CNN inference. Andri et al. [8] propose the use of pixel-wise scaling factors per tile, similar to those presented in [13], as a means to prevent quantization overflow in the Winograd-based convolutions. These scaling factors are efficiently determined during training through the use of powers-of-two values. Furthermore, they explore the different quantization levels at various stages of transformations and realize an 8-bit WAT pipeline using knowledge distillation. However, the quantized model still requires pixel-wise scaling factors, demanding additional dequantization to perform inverse transformation.

Our work also proposes a novel WAT to realize 8-bit quantized CNNs with no overhead in terms of hardware logic/storage, using only one scaling factor for all the transformations in one layer.

3. Methodology

We breakdown the proposed methodology into quantization-aware training, Winograd-aware training, quantization clipping factors in the Winograd domain, and complex numbers based Winograd algorithm. The following subsections detail each part of the method.

3.1. Quantization-Aware Training

Consider a convolutional layer $l \in [1,...,L]$ in an L-layer deep CNN with weights $\mathcal{W}^l \in \mathbb{R}^{k_y \times k_x \times C_i \times C_o}$ receiving an input feature map $\mathcal{A}^{l-1} \in \mathbb{R}^{H_i \times W_i \times C_i}$. H_i, W_i and C_i represent the feature map's spatial and channel dimensions. k_y, k_x and C_o are the kernel window and output channel dimensions. The convolution of $\mathcal{W}^l$ and $\mathcal{A}^{l-1}$ produces $\mathcal{A}^l \in \mathbb{R}^{H_o \times W_o \times C_o}$, where H_o, W_o, and C_o are output spatial and channel dimensions. Each layer requires a vast number of MAC operations, quantified as:

$$\#\text{MAC} = H_o \times W_o \times C_o \times k_x \times k_y \times C_i \tag{1}$$

Modern CNNs are characterized by a huge number of parameters, usually represented as 32-bit floating-point values during training. However, inference on limited resource hardware of such big models limits performance. Therefore, activations and weights are quantized on fewer N bits (e.g. 8, 4, 2, 1) to increase throughput, reduce memory consumption and limit power. The quantization of a floating-point value x_f is shown in Eq. 2 and Eq. 3. For activation quantization, x_f is clipped between $[-c,+c]$, where c represents the trainable clipping threshold value for every layer and it is determined by the task specific loss function of the CNN model [14]. Based on the determined c, a scaling factor is computed as $SF_a = c/(2^{N-1}-1)$. Considering the ReLU activation function, the range is clipped to $[0,c]$, resulting in $SF_a = c/(2^N-1)$, representing the quantized value as an unsigned number.

$$x_{int} = \mathtt{Q_c}(x_f) = \mathtt{Round}(\mathtt{Clip}(x_f,-c,+c)/SF_a) \tag{2}$$

Floating-point weights are quantized following Eq. 3, where the distribution is limited between $[-1,+1]$ [30].

$$x_{int} = \mathtt{Q}(x_f) = \mathtt{Round}(x_f/SF_w) \tag{3}$$

This results in a scaling factor for weights $SF_w = 1/(2^{N-1}-1)$. In order to deal with the discreteness of Eq. 2 and Eq. 3 at training time, a straight-through estimator (STE) is used to update full-precision weights during backpropagation, while quantized values are used in the forward pass [11]. In this paper we focus on 8-bit quantization, where all operands are represented as 8-bit integers.

3.2. Winograd-Aware training

Winograd algorithms can be used to reduce the number of MACs in a convolution operation [21]. We refer to 2D Winograd algorithms as $F(m_x \times m_y, k_x \times k_y)$, where the output tile size is $m_x \times m_y$ and the kernel size is $k_x \times k_y$. To compute an $m_x \times m_y$ output tile, standard convolution requires $(m_x \times k_x) \times (m_y \times k_y)$ multiplications, whereas the 2D Winograd convolution requires only $(m_x + k_x - 1) \times (m_y + k_y - 1)$ multiplications. The $m_x \times m_y$ output tile is obtained through the algorithm flow shown in Fig. 1 and Eq. 4.

$$a^l = A^T[(Gw^lG^T) \odot (B^T a^{l-1} B)]A \tag{4}$$

Here, a^{l-1}, w^l, and a^l are 2D tiles of the input feature map $\mathcal{A}^{l-1}$, of the weight tensor $\mathcal{W}^l$, and of the convolution output $\mathcal{A}^l$, respectively. The symbol $\odot$ represents the EWMM operator. B, G, A are the *constant* Winograd matrices responsible for transforming a^{l-1}, w^l and a^l to and from the Winograd domain [21]. The transformed input tile $B^T a^{l-1} B$ and weights Gw^lG^T are of dimensions $r_x \times r_y$, where $r_x = m_x + k_x - 1$ and $r_y = m_y + k_y - 1$. For the whole convolutional layer, the number of MAC operations needed by the Winograd algorithm is expressed in Eq. 5.

$$\#\text{MAC}_{wino} = \left\lceil \frac{H_o}{m_y} \right\rceil \times r_y \times \left\lceil \frac{W_o}{m_x} \right\rceil \times r_x \times C_o \times C_i \tag{5}$$

Therefore, MAC reductions achieved with the Winograd algorithm for a convolutional layer can be computed as the ratio of Eq. 1 to Eq. 5. For the floating-point numerical representation, Winograd algorithms can easily replace standard convolution, with almost zero accuracy degradation in the CNN. However, when data-precision is reduced, Winograd transformations cause a non-negligible numerical error that heavily affects the prediction quality. The lower the data-width adopted (N) is, the higher the numerical instability. As mentioned earlier, for Winograd $F(4 \times 4, 3 \times 3)$ 16-bit integer variant, numerical instability causes a 75 p.p. accuracy degradation for ResNet-18 on CIFAR-10 [16]. This numerical error is caused by two main reasons: 1) numerical range *enlargement* after Winograd transform, and 2) *rounding error* exacerbated by undertilizing the quantized range, as visualized in Fig. 2a. Winograd transformation matrices present non-integer coefficients that cause numerical errors when the transformation is performed in the quantized domain. Moreover, the transformed input tile values ($B^T a^{l-1} B$) are a linear combination of activations. Thus, Winograd transformations enlarge the exploited numerical range by a factor that can be referred to as the *enlargement factor* γ. In this work, we consider 2D Winograd algorithms for convolution kernels of size 3×3, which are common in modern CNNs. Therefore, from now on we will refer to $F(m_x \times m_y, 3 \times 3)$ as $F(m, 3)$, with $k_x = k_y = 3$. The operation reduction ratio, S, is related to the output tile size m as shown in Eq. 6. The larger the size of m is, the more MAC reductions can be achieved with respect to standard convolution.

$$S = \frac{m^2 \times 3^2}{(m + 3 - 1)^2} \tag{6}$$

In Table 1, we compare four different Winograd algorithms named $F(2, 3)$, $F(3, 3)$, $F(4, 3)$, and $F(6, 3)$. For each Winograd variant, we report the theoretical reduction in the number of operations (S) with respect to standard convolution and the maximum enlargement factor (γ) computed considering the worst-case scenario. Moreover, according to the SotA [9, 13], we consider that the weight transformation (Gw^lG^T) is performed offline, using floating-point representation to avoid unnecessary transform latency and quantization error at run-time.

The offline, accurately pre-transformed, quantized Winograd weights are readily available for the EWMM at runtime. Although $F(6, 3)$ presents the highest theoretical MAC reduction, it also has the greatest enlargement factor (γ) and memory overhead for storing weights. Moreover, $F(6, 3)$ transformation matrices require non-integer multiplications (supplementary S1.C), making the 8-bit transformation of the activations challenging for 8-bit hardware. $F(2, 3)$ with its small enlargement factor and simple transformation matrices (supplementary S1.D) guarantees a limited numerical error, however, it presents the smallest operations reduction. A further point to consider is that the theoretical MAC reduction S cannot always be achieved across the whole CNN. For layers not supported by the Winograd algorithm, standard convolution has to be used with no operation savings. The real MAC reduction is also related to layer spatial dimensions (Eq. 5). If either H_o or W_o are not divisible by m, extra unnecessary operations are performed in the Winograd algorithm. For instance, if $H_o = W_o = 7$ (as in the fifth residual module in ResNet-18 [17]), each output tile of $F(6, 3)$ will have $m \times m = 6 \times 6$ pixels, leading to a 12×12 output feature map and 256 MACs. Instead, in the $F(4, 3)$ algorithm, each output tile will have 4×4 pixels, leading to an 8×8 output feature map and only 144 MACs, resulting in better MAC reductions. For this reason, in Table 1, we also reported the real MAC reductions for the analyzed algorithms for a case study CNN model, ResNet-18. In this work, we focus on the $F(4, 3)$ Winograd algorithm since it represents the best compromise in terms of real operation reduction, weight memory overhead, and enlargement factor among the analyzed algorithms. The Winograd matrices for the $F(4, 3)$ algorithm are shown in supplementary S1.A.

We embedded the Winograd algorithm in the training loop, replacing standard convolution with Winograd-based convolution where possible. Contrary to other works [9, 13, 22, 26], we implement **full 8-bit quantization** for the Winograd algorithm, making our models ready to be

Table 1. Comparison of different Winograd algorithms to standard convolution in terms of enlargement factor (γ), weight memory, theoretical and practical (ResNet-18) number of MAC savings (#MAC).

Algorithm	γ	Weight Memory	#MAC Reduction Theoretical (S)	ResNet-18
F(2,3)	4×	1.78×	2.25×	1.76×
F(3,3)	36×	2.78×	3.24×	2.05×
F(4,3)	**100**×	**4**×	**4**×	**2.45**×
F(6,3)	156.25×	7.1×	5.06×	2.24×

deployed on 8-bit hardware (e.g. [1–4, 28]). We quantize weights, activations, transformed weights, and transformed activations. For weights and activations, we use the standard quantization approaches in [30] and [14], respectively. Furthermore, we introduce two other quantization steps within the Winograd domain. For quantizing transformed weights and activations, we first use Eq. 3 with $SF = max(|X_f|)/2^8 - 1$. X_f represents the weight or activation tensor containing the x_f elements. The Winograd algorithm (Eq. 4) can be rewritten as Eq. 7.

$$a^l = A^T \left[\mathtt{Q}(G\mathtt{Q}(w^l)G^T) \odot \mathtt{Q}(B^T \mathtt{Q}_\mathtt{C}(a^{l-1})B) \right] A \quad (7)$$

3.3. Clipping Factors in the Winograd Domain

We investigate Winograd algorithms to fully understand the main reasons that make them challenging to deploy on edge hardware. For this, we analyze the distribution of numerical values for different quantized layers for weights and activations before and after Winograd transformation. The data range for activations gets bigger after transformation because of the enlargement factor introduced in Sec. 3.2. Moreover, by analyzing the distribution of the transformed activations and weights within the range, we observe that only a small interval contains 99.9% of the data, while the rest of the range is very sparsely used (Fig. 3). Quantizing such a big floating-point range to 8-bit causes an under-utilization of the 2^8 discrete, quantized values available. Moreover, small magnitude values (99.9%) in the floating-point representation are mapped to close or same values in the quantized representation, causing under-utilization and a high rounding error. This is visualized in Fig. 2a. To address this issue, we introduce the concept of **clipping factors** ($\boldsymbol{\alpha_t}$) within the Winograd domain. The transformed range is clipped to α_t according to layer statistics, in particular, $[-\alpha_t, +\alpha_t]$ is defined as the range that contains 99.9% of the available data distribution. We allow weights and activations to have independent clipping factors (i.e. α_{ta} and α_{tw}). With this approach, the transformed inputs and weights better exploit the available discrete, quantized values (Fig. 2b).

The algorithm in Eq. 7 can be reformulated as in Eq. 8, where $\mathtt{Q}_{\mathtt{Cw}}$ denotes the clipping in the Winograd domain.

$$a^l = A^T \left[\mathtt{Q}_{\mathtt{Cw}}(G\mathtt{Q}(w^l)G^T) \odot \mathtt{Q}_{\mathtt{Cw}}(B^T \mathtt{Q}_\mathtt{C}(a^{l-1})B) \right] A \quad (8)$$

Trainable Clipping Factors α_{ta}, α_{tw}: Although clipping factors help in better exploiting the quantized range, computing them in a static, handcrafted way is a sub-optimal solution. Therefore, we make α_{ta}, α_{tw} dynamically adjustable through gradient descent training with the aim of minimizing the numerical error brought by Winograd transformations and quantization. Firstly, we clip the floating-point range to $[-\alpha_t, +\alpha_t]$ (Eq. 9).

$$x_c = 0.5(|x_f + \alpha_t| - |x_f - \alpha_t|) = \begin{cases} -\alpha_t, & x_f \in (-\infty, -\alpha_t) \\ x_f, & x_f \in [-\alpha_t, +\alpha_t) \\ +\alpha_t, & x_f \in [+\alpha_t, +\infty) \end{cases} \quad (9)$$

Then, we linearly quantize x_c on the available 8-bits quantized range:

$$x_q = \mathtt{Round}(x_c \cdot \frac{2^{(8-1)} - 1}{\alpha_t}) \quad (10)$$

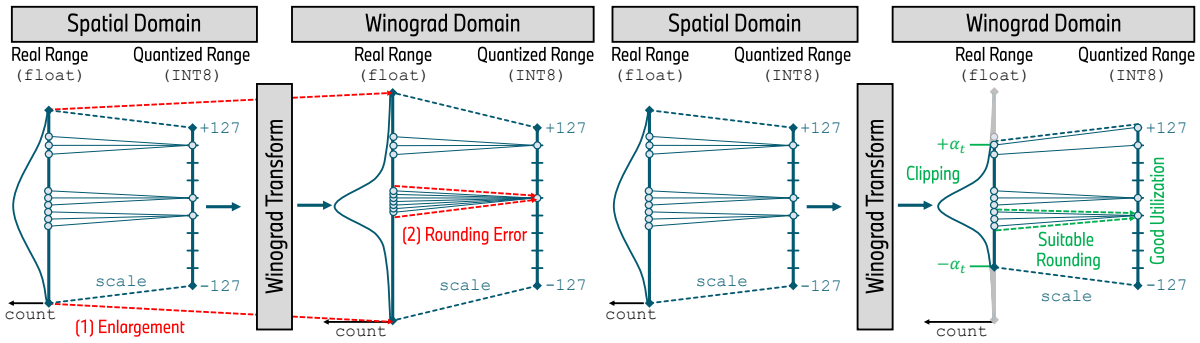
During backpropagation, the gradient $\frac{\delta x_q}{\delta \alpha_t}$ is evaluated using STE [11] to approximate $\frac{\delta x_q}{\delta x_f} = 1$:

$$\frac{\delta x_q}{\delta \alpha_t} = \frac{\delta x_q}{\delta x_f} \frac{\delta x_f}{\delta \alpha_t} = \begin{cases} -1, & x_f \in (-\infty, -\alpha_t) \\ 0, & x_f \in [-\alpha, +\alpha_t) \\ +1, & x_f \in [+\alpha_t, +\infty) \end{cases} \quad (11)$$

In summary, for each layer of a CNN model, we have three different scalar clipping factors: c, used for clipping input activations [14], α_{ta}, used for clipping transformed activations in the Winograd domain, and α_{tw}, adopted to limit the range for transformed weights in the Winograd domain. In terms of hardware, our novel approach requires **only two scalar** scaling factors per **layer**: one for the activation transform and one for standard output quantization (since weights are quantized offline), resulting in a negligible computation overhead. Initialization of clipping factors is a crucial operation that affects the prediction quality during training. In order to properly initialize α_{ta} and α_{tw}, we evaluate the data distribution during warm-up, selecting the initial α values so that 99.9% of the data values (transformed weights and activations) are within the range $[-\alpha_t, +\alpha_t]$. We do not apply L2 regularization loss on any of our clipping factors, since it penalizes high magnitude values for clipping instead of their effect on accuracy.

3.4. Complex Numbers based Winograd Algorithm

We use complex interpolation points to compute Winograd transformation matrices (A, B, G), as proposed by



(a) Standard Winograd transformation.

(b) Proposed Trainable Clipping in Winograd Transformation.

Figure 2. Comparison of the (a) standard Winograd quantized transformation against (b) our Winograd quantized transformation that leverages clipping factors to better exploit the quantized range.

[26]. In supplementary (S1.B), we report the complex-Winograd matrices B^T, G and A^T for the $F(4,3)$ Winograd algorithm. Comparing the complex matrices with those of standard $F(4,3)$ (S1.A), we observe that the magnitude of values involved in the transform is reduced, resulting in a smaller enlargement factor (γ).

Although the complex number representation is not hardware friendly, by exploiting the properties of complex conjugates and the Karatsuba algorithm, a *total* of 46 *real* integer multiplications have to be performed, compared to the 36 in the $F(4,3)$ algorithm [26]. The MAC operation reduction is then only slightly reduced, going down from $4\times$ to $3.13\times$. With proper tile organization there is no memory overhead (supplementary S2). Complex-based Winograd represents an interesting alternative to the standard Winograd algorithm presented in Sec. 3.2. We analyze the data distribution for the complex-Winograd algorithm to elaborate on why it outperforms standard Winograd when considering the numerical instability problem. In Table 2, we compared standard and complex Winograd variants of $F(4,3)$. The smaller integer values of the complex Winograd transformation matrices reduce the enlargement factor from $100\times$ to $16\times$. As discussed in Sec. 3.2, the smaller the enlargement factor is, the smaller the average rounding error becomes when the floating-point range is mapped to the quantized range. The weight memory overhead is the same as in standard $F(4,3)$, while the overall MAC reduction for the ResNet-18 model is only slightly affected ($2.11\times$ vs. $2.45\times$).We embed, for the first time, the complex-Winograd algorithm *in the training loop* as presented in Sec. 3.2, thereby improving its prediction quality. We then combine it with the trainable clipping factors introduced earlier to achieve further improvements. Ultimately, this enables the acceleration of complex 8-bit Winograd on hardware *without accuracy degradation* for the first time. Clipping factors are initialized according to the method described in Sec. 3.3, and the same clipping factors are used for imaginary and real parts.

Table 2. Comparison between Winograd $F(4,3)$ and complex-Winograd $F(4,3)$ algorithms.

Algorithm	γ	**Weight Memory**	#MAC **Savings** **Theoretical**	**ResNet-18**
F(4,3)	**100**$\times$	**4**$\times$	**4**$\times$	**2.45**$\times$
F(4,3) complex	16$\times$	4$\times$	3.13$\times$	2.11$\times$

4. Experiments

This section presents the results of applying the proposed techniques on ResNet-20, ResNet-18 [17], and DeepLabv3+ [12] evaluated on CIFAR-10 [20], ImageNet [27] and CityScapes [15] datasets. If not otherwise mentioned, all the training hyperparameters are adopted from the base implementation.

4.1. Transformed Weight Distribution

In Sec. 3.3, we described our approach to reduce numerical error in the Winograd domain by resorting to clipping factors for better exploitation of the quantized range. We compare our approach with post-training quantization (PTQ), where the model is trained using standard quantized convolution and the Winograd algorithm is enabled at inference time. We also compare against a vanilla Winograd-aware training approach (WAT), where the Winograd numerical error is considered at training time. Fig. 3 reports a comparison among the three approaches, showing the transformed weights distributions for a ResNet-20 model [17]. The range of numerical values (x-axis) is plotted against the number of times the value appears in the layer (y-axis, logarithmic scale). The trend described below is similar for all the layers, therefore we reported the one with the highest number of weight values. In all three distributions, most of the values are concentrated in a narrow interval. Winograd-awareness allows better exploitation of the available quantized range, as seen for the WAT (middle) and our trainable-clip approach (bottom). This is indicated by fully using the range between the symmetric clipping fac-

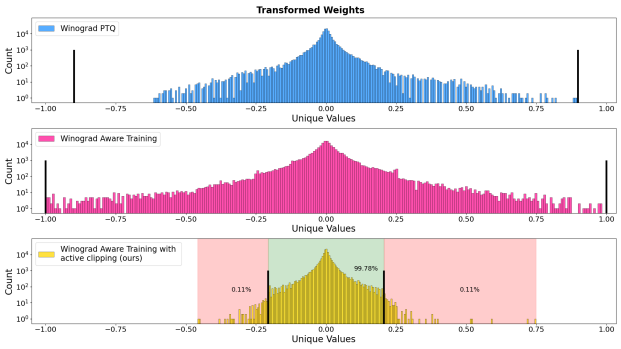


Figure 3. Transformed weight distribution of last layer of ResNet-20 for $F(4,3)$ algorithm for PTQ (top), WAT (middle) and our Winograd-aware trainable clipping approach (bottom).

tors (black bars). The PTQ Winograd-transformed weights (top) do not fully exploit the range allocated, as they show an asymmetric distribution. In summary, PTQ unaware of the Winograd transform has a severe disadvantage when being executed on integer arithmetic units as it does not effectively use the quantized range. WAT improves the utilization of the range, but leads to more rounding error due to its wider quantized range. Finally, our trainable clipping parameter WAT achieves a narrower numerical distribution to reduce the rounding error *and* fully exploits the quantized range. In supplementary S.3, we report distributions of multiple layers and we also show the effectiveness of clipping for transformed activations.

4.2. Effectiveness of Clipping Factors for Winograd

The ablation study for ResNet-20 [17] on the CIFAR-10 dataset [20] is reported in Table 3. For each experiment we report the accuracy value and the overall MAC reduction brought by Winograd. We first evaluated the effectiveness of our trainable clipping approach against a post-training quantized network. We restored the standard 8-bit quantized model (QConv) and enabled the Winograd $F(4,3)$ algorithm in the forward pass (WinoQConv). We froze the model weights and calibrated only the clipping factors for each layer and batch normalization parameters. Although there is still an accuracy degradation, clipping improves prediction quality by +46.75 p.p. compared to the standard PTQ Winograd $F(4,3)$ algorithm. To further reduce the accuracy gap, we then enabled WAT. Vanilla WAT alone achieves 89.69% accuracy and improves to 90.89% with our approach with trainable-clipping.

We then enabled the complex-Winograd algorithm in training. We evaluated the effectiveness of the full 8-bit complex algorithm with and without clipping factors. In both cases, it is interesting to note that the complex-Winograd algorithm not only closes the accuracy gap but achieves even better accuracy than the standard quantized model (+0.53 p.p. and +0.9 p.p.). This improvement is due to the higher representation capabilities of the complex domain. Although no complex operations are performed in hardware since only integer multiplications are executed (supplementary S.2), the real and the imaginary part can be combined to represent more than 256 unique values on 8-bit hardware with slightly lower MAC reductions (Sec. 3.4).

Table 3. Influence of WinoVidiVici on ResNet-20-CIFAR-10.

Method	$N_{W/A}$	QAT/ WAT	Winograd Algorithm	Winograd Complex	Winograd Clipping	Winograd Saving	Top-1 [%]
Conv [17]	32	✗	-	-	-	-	91.61
QConv [14]	8	✓	-	-	-	-	91.39
WinoQConv	8	✗	F(4,3)	✗	✗	3.4×	35.36
WinoVidiVici (Ours)	8	✗	F(4,3)	✗	✓	3.4×	82.11
		✓	F(4,3)	✗	✗	3.4×	89.69
		✓	F(4,3)	✗	✓	**3.4×**	**90.89**
		✓	F(4,3)	✓	✗	2.8×	91.92
		✓	F(4,3)	✓	✓	**2.8×**	**92.29**

We then evaluated the efficacy of our approach on more complex tasks. Accuracy values of the ResNet-18 [17] model on the ImageNet [27] dataset are reported in Table 4. In all the experiments, we trained the model for 80 epochs, adopting cosine decay for the learning rate scheduler (initial value 0.08, final value 0.0). We again observe the clear benefits of trainable-clipping factors in the Winograd domain to improve prediction quality. For the Winograd-aware trained model with $F(4,3)$, we improved prediction quality by +3.43 p.p. by adding the trainable-clipping parameters. Accuracy is further improved by the complex Winograd $F(4,3)$ algorithm with clipping factors, achieving +0.45 p.p. with respect to standard convolution 8-bit quantized model.

Table 4. Influence of WinoVidiVici on ResNet-18-ImageNet.

Method	$N_{W/A}$	QAT/ WAT	Winograd Algorithm	Winograd Complex	Winograd Clipping	Winograd Saving	Top-1 [%]
Conv [17]	32	✗	-	-	-	-	71.00
QConv [14]	8	✓	-	-	-	-	70.54
WinoQConv	8	✗	F(4,3)	✗	✗	2.45×	5.45
WinoVidiVici (Ours)	8	✓	F(4,3)	✗	✗	2.45×	65.71
		✓	F(4,3)	✗	✓	**2.45×**	**69.14**
		✓	F(4,3)	✓	✓	**2.11×**	**70.99**

We then evaluated the benefits of the Winograd algorithm on a semantic segmentation task (Table 5). For this experiment, we used the DeepLabv3+ [12] model on the CityScapes [15] dataset. We used a modified version of ResNet-18 as the backbone, where the last two residual blocks (Conv5) are removed. Winograd-awareness with trainable clipping factors again minimizes the accuracy degradation compared to a standard quantized convolution implementation. Further combining the complex-Winograd 8-bit algorithm with trainable clipping matches the prediction quality of a standard convolution 8-bit model (+0.03 p.p.), **fully eliminating the negative effects of numerical instability typically caused by quantized Winograd.**

Table 5. Influence of WinoViniVici on DeepLabv3+-CityScapes.

Method	$N_{W/A}$	QAT/ WAT	Winograd Algorithm	Winograd Complex	Winograd Clipping	Saving	mIoU [%]
QConv [14]	8	✓	-	-	-	-	67.82
WinoViniVici (Ours)	8	✓	F(4,3)	✗	✓	2.56×	66.57
		✓	F(4,3)	✓	✓	2.14×	67.85

4.3. Comparison with State of the Art

In this section we compare our approach with recent state-of-the-art works [8, 9, 13, 16, 22], all aiming to reduce numerical error for the 8-bit quantized Winograd algorithm. For each work, we reported the CNN model, the algorithm, the number of scaling factors (SF) for the transformation in each layer layer, bit-width of the transformed activations and the weights (N_{Wa}, N_{Ww} respectively), the number of MAC Operations (OPs) of the vanilla model and the real MAC reduction brought by the Winograd algorithm. Scaling and quantization operations are not considered in the number of operations since they add a negligible overhead (e.g. 0.2% for ResNet18 model). In the CIFAR-10 comparison, our approach achieves better or equal accuracy compared to all other works without adding any complexity to the algorithm. In particular, the authors of [13] adopted channel balancing and pixel-wise scaling factors within the tile to tackle the accuracy degradation, adding a significant amount of extra scaling factors (36 for transformation per layer). Furthermore, pixel-wise scaling factors force values within the same tile to belong to different ranges, requiring a complicated dequantization operation before inverse transformation. Moreover, output feature maps are not quantized. In [16], element-wise multiplication is performed in 8-bit. However, Winograd *transformation* matrices are not standard anymore (must be represented in floating-point), thereby increasing the computation complexity of the Winograd transformations at runtime. Moreover, having different transformation matrices for each layer is not as hardware-friendly as the standard Winograd matrices (A,B,G) (supplementary S1.A). Similarly, in [9] Winograd matrices are also not standard and the complexity of the algorithm is increased further with additional floating-point matrix multiplications. Note that in [9] and [16], the ResNet-18 model is used, which is much bigger than ResNet-20. The differences in number of operations among the ResNet-18 models is due to the channel-wise scaling coefficient of the model in the approach proposed in [9]. Our full 8-bit standard Winograd algorithms achieve better or equal accuracy to other works, with only one scaling factor and no hardware overhead. Our $F(4,3)$ complex implementation outperforms other ResNet-20 models (+0.54 p.p.) and achieves accuracy comparable to the ResNet-18 model (-0.17 p.p.) with 13.9× fewer total operations.

For the more challenging ImageNet comparison, our approaches outperform other works using ResNet-18 (+1.6 p.p. for trainable-clipping $F(4,3)$, +3.45 p.p. for complex $F(4,3)$). Although other works use different, deeper models such as ResNet-34 and ResNet-50, our method achieves similar accuracy values. Works such as [8, 13] need an extra dequantization operation because of pixel-wise scaling factors within the transformation tile. The work in [13] achieves the best accuracy but resorts to channel-wise *and* pixel-wise scaling factors ($36 \cdot C_o$). Li et al. [22] perform only element-wise matrix multiplication on 8-bit, while using floating-point representation for the other steps in the Winograd algorithm. Moreover, ResNet-50 is composed of 53 layers, of which only 16 are accelerated by Winograd convolution, resulting in lower MAC reductions. This makes ResNet-50 training with Winograd less challenging compared to ResNet-18/-34, as most of its layers are executed with standard convolution.

Table 6. Comparison of our approach with SotA.

Dataset	Model| Method	N_W/N_A	Winograd Algorithm	SF	OPs [10^8]	Saving [×]	Top-1 [%]
CIFAR-10 [20]	ResNet-20 | [13]	8/8	F(4,3)	36	0.41	3.4×	91.75
	ResNet-18* | [9]	8/8	F(4,3)	1	1.39*	2.9×	85.00
		8/9					89.40
		8/8	F(4,3)-flex				91.80
		8/9					92.30
	ResNet-18 | [16]	8/8	F(4,3)-flex	1	5.63	2.45×	92.46
	ResNet-20 | (Ours)	8/8	F(4,3)	**1**	0.41	3.4×	90.89
		8/8	F(4,3)-complex			**2.8**×	**92.29**
ImageNet [27]	ResNet-18 | [13]	8/8	F(4,3)	36	18.1	2.45×	67.54
		8/8		$36{\cdot}C_o$			68.94
	ResNet-34 | [8]	8/8	F(4,3)	1	36.6	2.83×	59.00
		8/8		36			71.10
	ResNet-50 | [22]	8/8	F(4,3)	1	38.6	1.54×	75.53
	ResNet-18 | (Ours)	8/8	F(4,3)	**1**	18.1	2.45×	69.14
		8/8	F(4,3)-complex			2.11×	**70.99**

*: Resnet18 with number of channels coefficient equal to 0.5.

5. Conclusion

In this paper, we proposed two approaches to tackle the numerical instability problem affecting 8-bit quantized execution of Winograd algorithms on edge devices. Using a trainable clip parameter during Winograd-aware training allowed the transform to become more stable on integer hardware. Then, we introduced, for the first time, complex numbers in Winograd-aware training, improving the representation capabilities of the low-bitwidth Winograd operands and enabling the use of complex transformation matrices that yield lower enlargement factors. Combining these two techniques, which do not add hardware overheads, we eliminated the negative effects of Winograd numerical instability on the prediction quality of CNNs. We achieved 2.11× and 2.14× MAC reduction on ResNet-18-ImageNet and DeepLabV3+ on CityScapes, with no prediction quality degradation. In cases where below-8-bit quantization does not bring any performance benefits, the contributions of this work enable the use of accelerated, Winograd-based convolution on a wide range of 8-bit general-purpose hardware without degrading the prediction quality.

References

[1] Nvidia jetson nano developer kit. https://cdn.sparkfun.com/assets/0/7/f/9/d/jetson-nano-devkit-datasheet-updates-us-v3.pdf, 2019. 1, 2, 5

[2] Google coral. https://coral.ai/static/files/Coral-M2-Dual-EdgeTPU-datasheet.pdf, 2020. 1, 2, 5

[3] Implementing the tensorflow deep learning framework on qualcomm's low-power dsp. https://www.edge-ai-vision.com/2017/07/implementing-the-tensorflow-deep-learning-framework-on-qualcomms-low-power-dsp-a-presentation-from-google/, 2020. 1, 5

[4] Snapdragon 8 gen 1 mobile platform. https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/snapdragon-8-gen-1-mobile-platform-product-brief.pdf, 2021. 1, 2, 5

[5] Convolution functions and data type support in cudnn framework. https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html#convolution, 2023. 1

[6] Winograd based convolution in tensorflow framework for 3x3 kernels. https://github.com/tensorflow/tensorflow/blob/210fbf4ccf2b7c6987228000a8a1a39957a5d4f3/tensorflow/lite/delegates/gpu/common/winograd_util.cc, 2023. 1

[7] Syed Asad Alam, Andrew Anderson, Barbara Barabasz, and David Gregg. Winograd convolution for deep neural networks: Efficient point selection, 2022. 1

[8] Renzo Andri, Beatrice Bussolino, Antonio Cipolletta, Lukas Cavigelli, and Zhe Wang. Going further with winograd convolutions: Tap-wise quantization for efficient inference on 4x4 tiles. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 582–598. IEEE, 2022. 3, 8

[9] Barbara Barabasz. Quantaized winograd/toom-cook convolution for dnns: Beyond canonical polynomials base. *arXiv preprint arXiv:2004.11077*, 2020. 3, 4, 8

[10] Barbara Barabasz, Andrew Anderson, Kirk M Soodhalter, and David Gregg. Error analysis and improving the accuracy of winograd convolution for deep neural networks. *ACM Transactions on Mathematical Software (TOMS)*, 46(4):1–33, 2020. 1

[11] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 3, 5

[12] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018. 6, 7

[13] 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 (CVPR)*. 2, 3, 4, 8

[14] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018. 3, 5, 7, 8

[15] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Scharwächter, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset. In *CVPR Workshop on The Future of Datasets in Vision*, 2015. 6, 7

[16] Javier Fernandez-Marques, Paul Whatmough, Andrew Mundy, and Matthew Mattina. Searching for winograd-aware quantized networks. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 14–29, 2020. 2, 3, 4, 8

[17] 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*, pages 770–778, 2016. 4, 6, 7

[18] Jingbo Jiang, Xizi Chen, and Chi-Ying Tsui. A reconfigurable winograd cnn accelerator with nesting decomposition algorithm for computing convolution with large filters, 2021. 1

[19] Minsik Kim, Cheonjun Park, Sungjun Kim, Taeyoung Hong, and Won Woo Ro. Efficient dilated-winograd convolutional neural networks. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2711–2715, 2019. 1

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6, 7, 8

[21] 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. 1, 4

[22] Guangli Li, Zhen Jia, Xiaobing Feng, and Yida Wang. Lowino: Towards efficient low-precision winograd convolutions on modern cpus. In *50th International Conference on Parallel Processing*, pages 1–11, 2021. 2, 4, 8

[23] Xingyu Liu, Jeff Pool, Song Han, and William J. Dally. Efficient sparse-winograd convolutional neural networks, 2018. 1

[24] Zhi-Gang Liu and Matthew Mattina. Efficient residue number system based winograd convolution. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX*, pages 53–68. Springer, 2020. 2

[25] Partha Maji, Andrew Mundy, Ganesh Dasika, Jesse Beu, Matthew Mattina, and Robert Mullins. Efficient winograd or cook-toom convolution kernel implementation on widely used mobile cpus, 2019. 1

[26] Lingchuan Meng and John Brothers. Efficient winograd convolution via integer arithmetic. *arXiv preprint arXiv:1901.01965*, 2019. 2, 3, 4, 6

[27] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 6, 7, 8

[28] Chen Yang, Yizhou Wang, Xiaoli Wang, and Li Geng. Wra: A 2.2-to-6.3 tops highly unified dynamically reconfigurable

accelerator using a novel winograd decomposition algorithm for convolutional neural networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(9):3480–3493, 2019. 1, 5

[29] Tao Yang, Yunkun Liao, Jianping Shi, Yun Liang, Naifeng Jing, and Li Jiang. A winograd-based cnn accelerator with a fine-grained regular sparsity pattern. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 254–261, 2020. 1

[30] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 3, 5

S.1 Winograd Matrices

Comparison among Winograd transform matrices (A^T, B^T, G) for four algorithms ($F(4,3)$, complex-$F(4,3)$, $F(6,3)$ and $F(2,3)$). Although $F(6,3)$ S1.C algorithm provides a bigger MAC reduction ($5\times$) compared to other algorithms, its matrices present non-integer values that require floating-point units to perform transformations, increasing the computational complexity. $F(2,3)$ S1.D transform matrices contain integer hardware-friendly coefficients that minimize the numerical error discussed in this paper, however, MAC reduction is limited compared to other algorithms ($2.25\times$). We identified the $F(4,3)$ S1.A algorithm as the best compromise in terms of MAC reduction ($4\times$), memory overhead for offline weights transformation and numerical error. In this paper we provide a novel technique to recover the accuracy degradation introduced by 8-bit quantized Winograd algorithms. Our approach performs even better on the complex-$F(4,3)$ S1.B algorithm, where, at the cost of a *slightly* lower MAC reduction ($3.13\times$), we fully recover the accuracy degradation for the full-8 bit Winograd algorithm.

F(4,3)

$$B^T = \begin{bmatrix} 4 & 0 & -5 & 0 & 1 & 0 \\ 0 & -4 & -4 & 1 & 1 & 0 \\ 0 & 4 & -4 & -1 & 1 & 0 \\ 0 & -2 & -1 & 2 & 1 & 0 \\ 0 & 2 & -1 & -2 & 1 & 0 \\ 0 & 4 & 0 & -5 & 0 & 1 \end{bmatrix} G = \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ -\frac{1}{6} & -\frac{1}{6} & -\frac{1}{6} \\ -\frac{1}{6} & \frac{1}{6} & -\frac{1}{6} \\ \frac{1}{24} & \frac{1}{12} & \frac{1}{6} \\ \frac{1}{24} & -\frac{1}{12} & \frac{1}{6} \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & 0 \\ 0 & 1 & 1 & 4 & 4 & 0 \\ 0 & 1 & -1 & 8 & -8 & 1 \end{bmatrix} \tag{S1.A}$$

F(4,3) complex

$$B^T = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & -1 & 1 & -1 & 1 & 0 \\ 0 & -j & -1 & j & 1 & 0 \\ 0 & j & -1 & -j & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 \end{bmatrix} G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{j}{4} & -\frac{1}{4} \\ \frac{1}{4} & -\frac{j}{4} & -\frac{1}{4} \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & j & -j & 0 \\ 0 & 1 & 1 & -1 & -1 & 0 \\ 0 & 1 & -1 & -j & j & 1 \end{bmatrix} \tag{S1.B}$$

F(6,3)

$$B^T = \begin{bmatrix} 1 & 0 & -21/4 & 0 & 21/4 & 0 & -1 & 0 \\ 0 & 1 & 1 & -17/4 & -17/4 & 1 & 1 & 0 \\ 0 & -1 & 1 & 17/4 & -17/4 & -1 & 1 & 0 \\ 0 & 1/2 & 1/4 & -5/2 & -5/4 & 2 & 1 & 0 \\ 0 & -1/2 & 1/4 & 5/2 & -5/4 & -2 & 1 & 0 \\ 0 & 2 & 4 & -5/2 & -5 & 1/2 & 1 & 0 \\ 0 & -2 & 4 & 5/2 & -5 & -1/2 & 1 & 0 \\ 0 & -1 & 0 & 21/4 & 0 & -21/4 & 0 & 1 \end{bmatrix} G = \begin{bmatrix} 1 & 0 & 0 \\ -2/9 & -2/9 & -2/9 \\ -2/9 & 2/9 & -2/9 \\ 1/90 & 1/45 & 2/45 \\ 1/90 & -1/45 & 2/45 \\ 32/45 & 16/45 & 8/45 \\ 32/45 & 16/45 & 8/45 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 1 & 4 & 4 & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 1 & -1 & 8 & -8 & \frac{1}{8} & -\frac{1}{8} & 0 \\ 0 & 1 & 1 & 16 & 16 & \frac{1}{16} & \frac{1}{16} & 0 \\ 0 & 1 & -1 & 32 & -32 & \frac{1}{32} & -\frac{1}{32} & 1 \end{bmatrix} \tag{S1.C}$$

F(2,3)

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} G = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix} \tag{S1.D}$$

S.2 Complex tile organization

In Fig. 4, the complex-Winograd $F(4,3)$ algorithm is shown. Green and yellow cells represent real and complex values, respectively. Each matrix in the complex-Winograd domain contains 10 pairs of complex conjugate values and 16 real values. Each complex conjugate pair (x_{c1} and x_{c2}) is defined as reported in equation S2.A.

$$x_{c1} = x_r + jx_j \qquad\qquad x_{c2} = x_r - jx_j \tag{S2.A}$$

x_r and x_j represent the real and imaginary parts, respectively. Exploiting the complex conjugates property, we can store only the real part and the imaginary part, and build (if necessary), the complex conjugate by adding and subtracting the two values. This is shown in in Fig. 5, where the blue and yellow parts can be combined to reproduce the needed complex conjugates accordingly. Moreover, the number of *real* multiplications required to perform the complex element-wise matrix multiplication (EWMM) should be $16 + 4 \times 20 = 96$. However, the 20 complex multiplications can be grouped into 10 pairs of complex conjugate multiplications and by using the Karatzuba algorithm, each complex multiplication takes only three real multiplications. Therefore, the total number of *real* multiplications needed to perform complex EWMM can be rewritten as: $16 + 3 \times 10 = 46$.

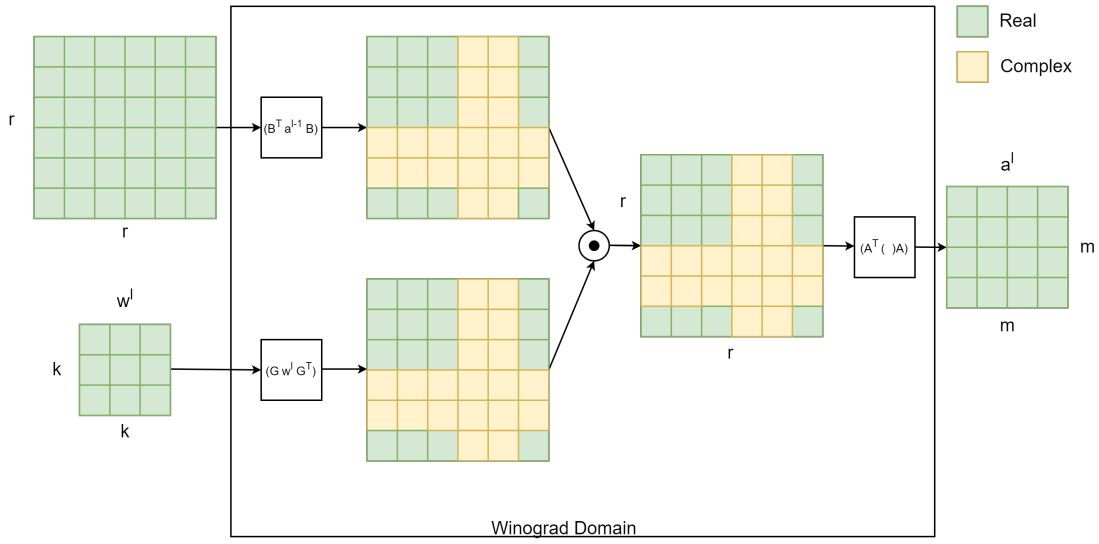


Figure 4. Complex Winograd $F(4,3)$ algorithm. Green and yellow elements represent real and complex values, respectively.

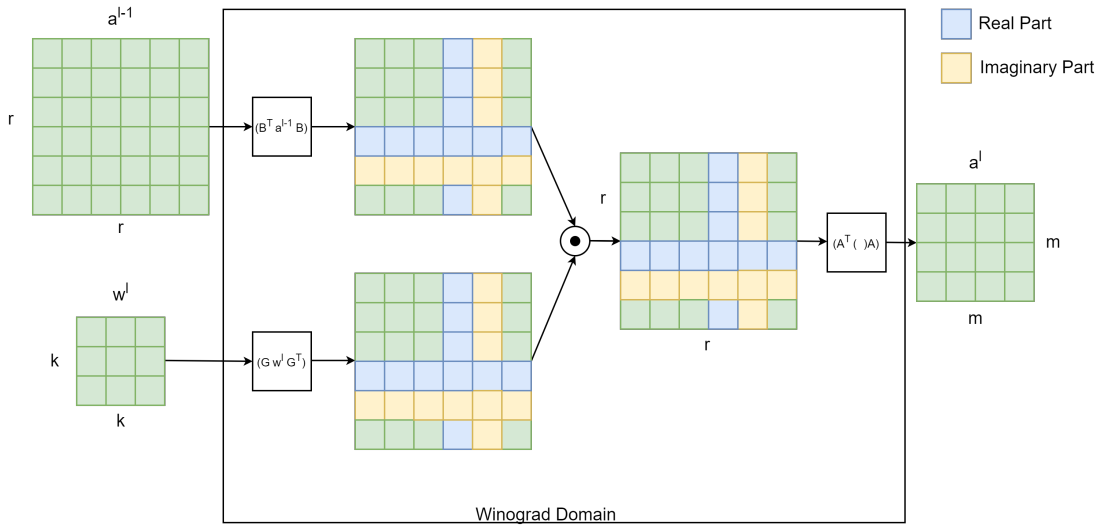
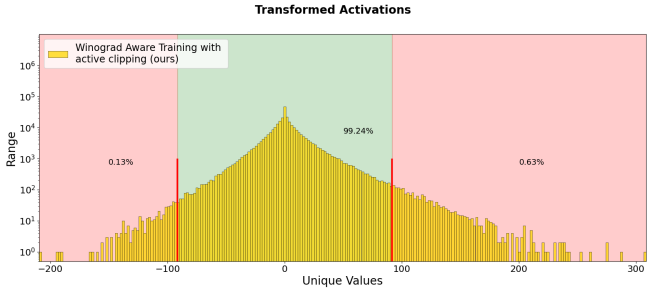


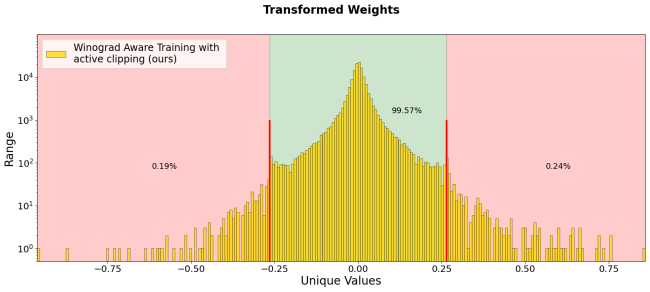
Figure 5. Efficient Complex Winograd $F(4,3)$ algorithm representation. Complex conjugates values can be stored as real (blue) and imaginary parts (yellow).

S.3 Effectiveness of Clipping Factors

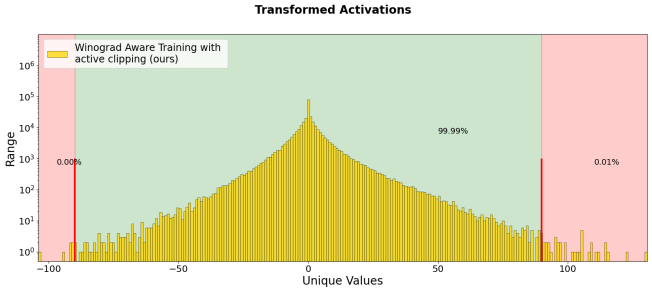
In Fig. 6, we show further examples on the distributions of transformed weights and transformed activations when using the proposed method with Winograd $F(4,3)$ algorithm for three layers of the ResNet-20 model (layers 15, 16, and 17, respectively) trained on CIFAR-10. For activations, the overflow factor increases the numerical range in the Winograd domain, causing a huge quantization error that leads to severe accuracy degradation. Our approach allows to dynamically limit the distribution, guaranteeing a better exploitation of the quantized range. Our method also effectively clips the transformed weights, maintaining over 99% of the numbers appearing in the transform. More generally, the clipping range highlighted in green sufficiently covers all the necessary data to achieve a transformation with no accuracy degradation when accelerating full 8-bit Winograd convolution on edge hardware.



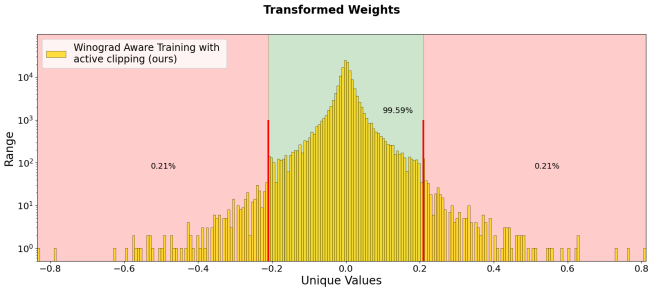
(a) Transformed activation numerical distribution of layer 15.



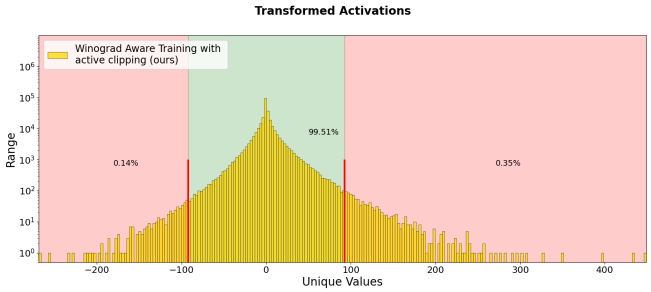
(b) Transformed weight numerical distribution of layer 15.



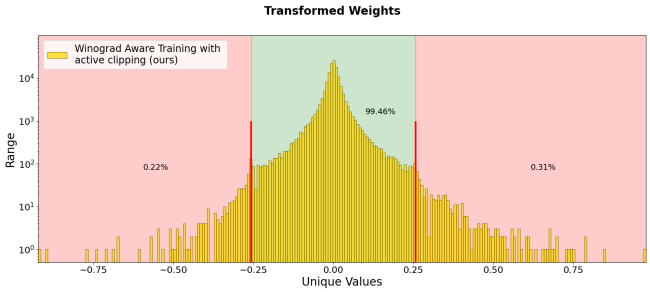
(c) Transformed activation numerical distribution of layer 16.



(d) Transformed weight numerical distribution of layer 16.



(e) Transformed activation numerical distribution of layer 17.



(f) Transformed weight numerical distribution of layer 17

Figure 6. Numerical distributions of example layers for transformed weights and activations of ResNet-20 on CIFAR-10. The values in the clipped range (green) sufficiently contain the information needed to maintain high-accuracy full 8-bit Winograd.