LOW-RANK WINOGRAD TRANSFORMATION FOR 3D CONVOLUTIONAL NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper focuses on Winograd transformation in 3D convolutional neural networks (CNNs) that are more over-parameterized compared with the common 2-D version. The over-increasing Winograd parameters not only exacerbate training complexity but also barricade the practical speedups due simply to the volume of element-wise products in the Winograd domain. We attempt to reduce trainable parameters by introducing a low-rank Winograd transformation, a novel training paradigm that decouples the original large tensor into two less storage-required trainable tensors, leading to a significant complexity reduction. Built upon our low-rank Winograd transformation, we take one step ahead by proposing a low-rank oriented sparse granularity that measures column-wise parameter importance. By simply involving the non-zero columns in the element-wise product, our sparse granularity is empowered with the ability to produce a very regular sparse pattern to acquire effectual Winograd speedups. To better understand the efficacy of our method, we perform extensive experiments upon 3D CNNs. Results manifest that our low-rank Winograd transformation well outperforms the vanilla Winograd transformation. We also show that our proposed low-rank oriented sparse granularity permits practical Winograd acceleration compared with the vanilla counterpart.

1 INTRODUCTION

Compared to their 2D counterparts, 3D convolutional neural networks (CNNs) have received substantial accuracy increases in many video processing tasks, as a result of their superior capacity of extracting spatio-temporal features within video frames. Unfortunately, the supreme performance is gained at the price of large amounts of computing resources for both training and inference, primarily because the 3D kernels are more computationally intensive. However, many restrictions such as runtime, memory and power budge prevent 3D CNNs from running on many real-world devices.

Fast convolution algorithms such as Winograd convolution (Lavin & Gray, 2016) and fast Fourier transform (FFT) (Mathieu et al., 2013) can greatly reduce the computational cost of the spatial convolution. The principle of Winograd convolution and FFT is to replace spatial convolution operations with element-wise product and discard redundant multiplications in convolution. Conventional FFT based convolution is fast for large filters, therefore most recent attention focuses on Winograd convolution principally for the small 3×3 filters adopted by state-of-the-art CNNs. Another potential line to tackle the over-parameterized issue is network pruning that reduces network complexity by removing unnecessary units (Frankle & Carbin, 2018; Luo et al., 2017). It seems that Winograd convolution and network pruning can be well combined to further save computation costs. However, they are not naturally compatible since the sparsity property from network pruning is diminished after the kernel transformation of the Winograd algorithm. To tackle this incompatibility, (Li et al., 2017b) performed pruning operations upon Winograd domain while (Liu et al., 2018) added the ReLU function after the Winograd transformation to increase the sparsity of element-wise product. However, both studies do not take into account a reality that Winograd kernel is position-sensitive as later demonstrated by (Li et al., 2017b) in which an importance factor matrix is further utilized to gauge the significance of different kernel positions.

In spite of the aforementioned progress, current investigations mostly give attention to the Winograd transformation in 2D CNNs $\big($see Fig. 1(a)$\big)$. A direct extension of these methods to 3D CNNs is

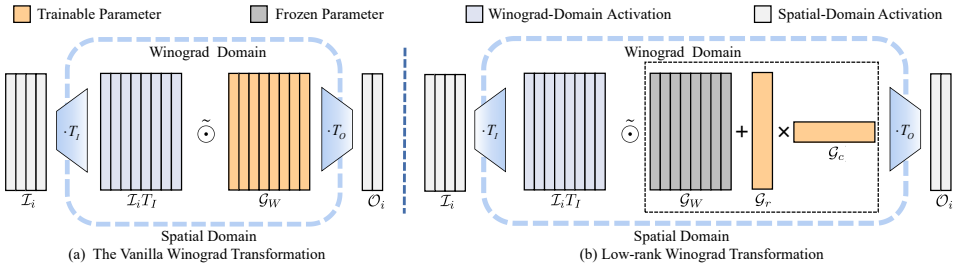


Figure 1: Comparison between (a) the vanilla Winograd transformation and (b) our low-rank Winograd transformation. We decouple the whole Winograd weights into two smaller matrices, leading to significant reduction in trainable parameters.

inapplicable, as we analyze, for two issues. First, 3D Winograd transformation causes considerable parameter increase. Taking F(2, 3)-based Winograd algorithm as an example, a typical 3D convolutional kernel with a shape of $3 \times 3 \times 3$ is often replaced by a $4 \times 4 \times 4$ Winograd kernel, leading to $2.37\times$ more parameters while it is only $1.78\times$ for 2D case. More parameters from Winograd transformation do not always benefit model capacity but causing model redundancy as analyzed in Sec. 3.2 and verified in Sec. 4.2. Also, the over-increasing trainable parameters pose a serious challenge to the capability of training machine. Second, existing methods fail to accelerate Winograd transformation even though conducting pruning upon Winograd domain. Similar to the weight pruning (LeCun et al., 1989; Han et al., 2015b; Frankle & Carbin, 2018), prior implementations derive irregular sparse weight matrix, which receives very limited speed gains since the irregular sparsity barely takes advantage of vector processing architectures such as single instruction multiple data (SIMD), and poorly utilizes memory buses (Lin et al., 2022). Therefore, it remains unsolved to excavate Winograd transformation for acceleration, in particular to 3D CNNs primarily for the ever-increasing element-wise product in the Winograd domain.

In this paper, we put forward a novel Winograd transformation for 3D CNNs towards solving the above issues. Considering the over-increasing parameters in 3D CNNs, as shown in Fig. 1(b), we introduce a low-rank Winograd transformation method that represents the updating matrix (variation from a pre-trained Winograd weight tensor to the final fine-tuned one) with two smaller matrices. In this fashion, we concentrate on updating weights in the main directions of the whole Winograd space during fine-tuning process, leading to superior performance over the vanilla Winograd transformation, or even better performance than the original spatial model. Besides, the two less storage-required matrices lead to significant reduction on trainable Winograd parameters, also much smaller than the original training parameters in the spatial domain. With regard to Winograd transformation acceleration, we further present a low-rank oriented sparse granularity that quantifies importance of each tensor column. The rationale behind this is to derive a more regular sparse pattern by simply involving the non-zero columns in the element-wise product of Winograd domain. To that effect, we introduce a scoring sequence to continuously accumulate the the magnitude and gradient of column position in each training iteration as the importance assessment, and finally remove all weights in compliance with the low-scored columns. Practical speedups are observed from our low-rank oriented sparse granularity. For example, under the pruning rates of 50% and 70%, we obtain $1.80\times$ and $3.35\times$ speedup gains on C3D model compared to the vanilla Winograd transformation (see Fig. 6).

2 Related Work

Spatial-Domain Pruning. Spatial-domain pruning is the practice of removing parameters from an existing network. It may entail removing individual parameters, *a.k.a.* weight pruning, or parameters in groups such as filter pruning and block pruning. For weight pruning, individual weights are measured by a certain criterion such as weight magnitude (Han et al., 2015b;a; Frankle & Carbin, 2018), higher-order information (LeCun et al., 1989; Hassibi et al., 1993; Dong et al., 2017) (Lee et al., 2019) and so on. These methods are demonstrated to well preserve model performance. However, the resulting irregular sparse matrix requires specialized hardware/libraries to achieve practical speedups. For filter pruning, the entire filters are removed by standards such as ℓ_1/ℓ_2-

norm (Li et al., 2017a; Liu et al., 2017; He et al., 2018), activation sparsity (Hu et al., 2016), lasso regression-based channel selection (He et al., 2017), and rank of feature maps (Lin et al., 2020). In contrast to weight pruning, filter pruning advantages in acceleration but causing more performance drops. Therefore, block pruning, where a block of weights is removed simultaneously, has received recent research focus (Meng et al., 2020; Niu et al., 2021; Lin et al., 2022) for its better performance than filter pruning as well as hardware-friendly deployment than weight pruning. However, vanilla spatial pruning methods cannot be directly combined with the Winograd convolution because the Winograd transformation diminishes the sparsity resulting from pruning (Yu et al., 2019).

Winograd-Domain Pruning. Though vanilla spatial pruning fails to cooperate with the Winograd convolution, its main pruning principles have been extended to remove parameters in Winograd domain. (Liu & Turakhia, 2016) removed Winograd-domain kernels, meanwhile they retained kernels from the original network. However, dimension inconsistency arises since the Winograd-domain kernels are of a higher dimension than the spatial-domain kernels. To solve this issue, (Li et al., 2017b) introduced Winograd layers in exchange for the standard convolutional layers. The pruning and training are simultaneously conducted in the Winograd layers. In this fashion, the dimension inconsistency issue is eliminated and the sparsity in Wiongrad domain also increases. (Liu et al., 2018) introduced the ReLU operation to the Winograd domain to derive sparse transformed activations. At the same time, it improves the possibility of sparse element-wise product in the Winograd domain. (Yu et al., 2019) specified that different positions of the Winograd layers contribute differently to the output activations. Despite the progress, these studies lead to hardware-unfriendly irregular sparse patterns, causing imbalanced workloads among the data flows. To leverage the multiplication reduction from sparsity, (Lu & Liang, 2018; Yang et al., 2020) devised sparse patterns that benefit more from the practical speedups on specialized hardware.

3 METHODOLOGY

3.1 3D WINOGRAD

Giving a 3D convolution weight $\mathcal{G} \in \mathbb{R}^{c_{out} \times c_{in} \times r_1 \times r_2 \times r_3}$ where c_{out} and c_{in} denote the input and output channel, and (r_1, r_2, r_3) forms the kernel size, it is convoluted with a 3D image $\mathcal{I} \in \mathbb{R}^{c_{in} \times d_{in} \times h_{in} \times w_{in}}$ where d_{in}, h_{in} and w_{in} respectively denote the image depth, height and width. The convolutional result is a 3D feature map $\mathcal{O} \in \mathbb{R}^{c_{out} \times d_{out} \times h_{out} \times w_{out}}$, where d_{out}, h_{out} and w_{out} respectively denote the feature map depth, height and width. Each element of $\mathcal{O}$ is computed in the spatial convolution with a stride of 1 as:

$$\mathcal{O}_{[m,n,x,y]} = \sum_{c=1}^{c_{in}} \sum_{u=1}^{r_1} \sum_{v=1}^{r_2} \sum_{w=1}^{r_3} \mathcal{G}_{[m,c,n,x,y]} \mathcal{I}_{[c,n+u,x+v,y+w]}. \tag{1}$$

By contrast, the Winograd convolution disassembles the input $\mathcal{I}$ into several overlapping tiles $\{\mathcal{I}_1, \mathcal{I}_2, ...\}$, in which each tile is a sub-matrix of $\mathcal{I}_i \in \mathcal{I}$ and has a shape of $c_{in} \times t_1 \times t_2 \times t_3$ where $t_1 = d_{out} + r_1 - 1$, $t_2 = h_{out} + r_2 - 1$, $t_3 = w_{out} + r_3 - 1$. A thorough comprehension can be referred to (Lavin & Gray, 2016). Notice in what follows, we introduce $r = r_1 = r_2 = r_3$ and $t = t_1 = t_2 = t_3$ for brevity since current networks have a uniform kernel size such as $3 \times 3 \times 3$ across different dimensions, and also the output feature is characterised with $d_{out} = h_{out} = w_{out}$. Similar to Eq. (1), each tile $\mathcal{I}_i$ can be convoluted separately with the weight $\mathcal{G}$, resulting in a basic output tile $\mathcal{O}_i$ that is a sub-matrix of $\mathcal{O}$ and has a shape of $c_{out} \times m \times m \times m$ where $m = d'_{out} = h'_{out} = w'_{out}$. Note that, any $\mathcal{O}_i$ and $\mathcal{O}_j$ are non-overlapping and the spatial convolution result $\mathcal{O}$ can be obtained by reassembling $\{\mathcal{O}_1, \mathcal{O}_2, ...\}$ in order.

Unfortunately, the spatial convolution is computationally intensive. Therefore, (Lavin & Gray, 2016) introduced Winograd's minimal filtering algorithm (Winograd, 1980) (Winograd algorithm) to reduce the complexity of convolution over each basic tile $\mathcal{I}_i$. The principle of Winograd is to replace convolution operations with element-wise product. Conventional studies (Vincent et al., 2017; Kim et al., 2019; Meng & Brothers, 2019; Alam et al.) focus on 1D Winograd algorithm abbreviated as $F(m, r)$ and 2D Winograd version as $F(m \times m, r \times r)$. This paper is devoted to 3D Winograd algorithm $F(m \times m \times m, r \times r \times r)$. The entire procedure can be generally formulated as:

$$\mathcal{O}_{i[m,:,:,:]} = \mathscr{T}_O\Big(\sum_{c=1}^{c_{in}} \big(\mathscr{T}_K(\mathcal{G}_{[m,c,:,:,:]}) \odot \mathscr{T}_I(\mathcal{I}_{i[c,:,:,:]}) \big) \Big), \tag{2}$$

where $\odot$ stands for element-wise product. In Eq. (2), the kernel $\mathcal{G}_{[m,c,:,:,:]}$ and input tile $\mathcal{I}_{i_{[c,:,:,:]}}$ are individually converted into the Winograd domain of the same shape by the Winograd kernel transformation $\mathscr{T}_K(x) = \mathscr{R}(GxG^T)G^T \in \mathbb{R}^{c_{out}\times c_{in}\times t\times t\times t}$ and input transformation $\mathscr{T}_I(x) = \mathscr{R}\big(B^TxB\big)B \in \mathbb{R}^{c_{in}\times t\times t\times t}$. Finally, the Winograd-domain kernel and input tile are multiplied in an element-wise manner, results of which are transformed back to the vanilla spatial domain by the Winograd inverse transformation $\mathscr{T}_O(x) = \mathscr{R}\big(\mathscr{R}(A^TxA)A\big) \in \mathbb{R}^{c_{out}\times c_{in}\times m\times m\times m}$. Herein, G, B and A are three transformation matrices determined by $F(m\times m\times m, r\times r\times r)$. Their specific formats can turn to (Lavin & Gray, 2016). $\mathscr{R}(\cdot)$ denotes clock-wise dimension rotation. Considering a 3-D matrix $mat_{[z,y,x]}$, a toy example for $\mathscr{R}(\cdot)$ is illustrated as: $\mathcal{R}(mat_{[z,y,x]}) = mat_{[y,x,z]}$.

Compared with the vanilla spatial-domain 3D convolution, 3D Winograd convolution $F(m\times m\times m, r\times r\times r)$ reduces the multiplication from $r^3\times m^3$ to $(m+r-1)^3$. Given the property that the transformation matrices G, B and A are filled with zero elements or identical/opposite elements and the operations in $\mathscr{T}_K(\cdot)$, $\mathscr{T}_I(\cdot)$, and $\mathscr{T}_O(\cdot)$ can be replaced by additions, the major computation bottleneck comes from the element-wise product.

Particularly, (Li et al., 2017b) introduced a 2D Winograd layer parameterized by a weight tensor $\mathcal{G}_W \in \mathbb{R}^{c_{out}\times c_{in}\times t\times t}$ to replace the Winograd kernel transformation. The element-wise operation costs are expected to decrease from deriving a sparse $\mathcal{G}_W$. In this paper, we extend the Winograd layer to 3D and introduce $\mathcal{G}_W \in \mathbb{R}^{c_{out}\times c_{in}\times t\times t\times t}$ to replace the Winograd-domain kernel for the element-wise product with the Winograd-domain input tile. Eq. (2) can be rewritten as:

$$\mathcal{O}_{i_{[m,:,:,:]}} = \mathscr{T}_O\big(\sum_{c=1}^{c_{in}} \mathcal{G}_{W[m,c,:,:,:]} \odot \mathscr{T}_I(\mathcal{I}_{i[c,:,:,:]})\big). \tag{3}$$

Different from the 1D or 2D Winograd layer, more parameters are introduced in 3D Winograd layer, which raises a formidable challenge to not only train the increasing parameters, but also speedup the element-wise product. These two issues are respectively solved in this paper by introducing a low-rank Winograd transformation in Sec. 3.2 and a low-rank oriented sparse granularity in Sec. 3.3, which are also two core contributions of this paper.

In what follows, we refer to the model with Winograd layers as the Winograd model and the one with convolutional layers as the spatial model.

3.2 Low-Rank Winograd Transformation

The Winograd model shares a two-step training pipeline similar to the spatial model, including inheriting weights from a pre-trained model, and fine-tuning on the downstream task. The former can be accomplished by using the Winograd-transformed spatial weights as the pre-trained weights, *i.e.*, $\mathcal{G}_W = \mathscr{T}_K(\mathcal{G})$. However, it is challenging to fine-tune a 3D Winograd model primarily due to the increasing trainable parameters and expensive element-wise product. When looking back on the Winograd weight $\mathcal{G}_W$, we observe that the over-increasing parameters do not always benefit the performance gains.

Before diving into an in-depth analysis, we first rearrange the 3D spatial kernel $\mathcal{G} \in \mathbb{R}^{c_{out}\times c_{in}\times t\times t\times t}$ and the basic input tile $\mathcal{I}_i \in \mathbb{R}^{c_{in}\times t\times t\times t}$ into 2D matrices $\mathcal{G} \in \mathbb{R}^{c_{out}c_{in}\times r^3}$ and $\mathcal{I}_i \in \mathbb{R}^{c_{in}\times t^3}$. The Winograd transformations $\mathscr{T}_K(\cdot)$, $\mathscr{T}_I(\cdot)$, and $\mathscr{T}_O(\cdot)$ are supposed to be modified accordingly:

$$\mathscr{T}_K(\mathcal{G}) = \mathscr{R}(G\mathcal{G}G^T)G^T,\ \mathcal{G}\in\mathbb{R}^{c_{out}\times c_{in}\times r\times r\times r} \rightarrow \mathscr{T}_K(\mathcal{G}) = \mathcal{G}T_K,\ \mathcal{G}\in\mathbb{R}^{c_{out}c_{in}\times r^3}, \tag{4}$$

$$\mathscr{T}_I(\mathcal{I}_i) = \mathscr{R}(B^T\mathcal{I}_iB)B,\ \mathcal{I}_i\in\mathbb{R}^{c_{in}\times t\times t\times t} \rightarrow \mathscr{T}_I(\mathcal{I}_i) = \mathcal{I}_iT_I,\ \mathcal{I}_i\in\mathbb{R}^{c_{in}\times t^3}, \tag{5}$$

$$\mathscr{T}_O(\mathcal{U}_i) = \mathscr{R}\big(\mathscr{R}(A^T\mathcal{U}_iA)A\big),\ \mathcal{U}_i\in\mathbb{R}^{c_{out}\times t\times t\times t} \rightarrow \mathscr{T}_O(\mathcal{U}_i) = \mathcal{U}_iT_O,\ \mathcal{U}_i\in\mathbb{R}^{c_{out}\times t^3}, \tag{6}$$

where $\mathcal{U}_i = \sum_{c=1}^{c_{in}} \mathscr{T}_K(\mathcal{G}_{[m,c,:,:,:]}) \odot \mathscr{T}_I(\mathcal{I}_{i[c,:,:,:]})$ is the operational result in the Winograd domain, and $T_K \in \mathbb{R}^{r^3\times t^3}$, $T_I \in \mathbb{R}^{t^3\times t^3}$ are transformation matrices that map $\mathcal{G}$ and $\mathcal{I}_i$ from the spatial domain to the Winograd domain and $T_O \in \mathbb{R}^{t^3\times m^3}$ maps $\mathcal{U}_i$ to the spatial domain. The whole forward process for the 3D Winograd layer can be modified from Eq. (3) as:

$$\mathcal{O}_i = \Big(\sum_{c_2=0}^{c_{out}-1}\big(\sum_{c_1=0}^{c_{out}-1}\mathcal{G}_{W[c_1\cdot c_{in}:(c_1+1)c_{in},:]} \odot (\mathcal{I}_iT_I)\big)\Big)_{[c_2\cdot c_{in}:(c_2+1)c_{in},:]}\Big)T_O, \tag{7}$$

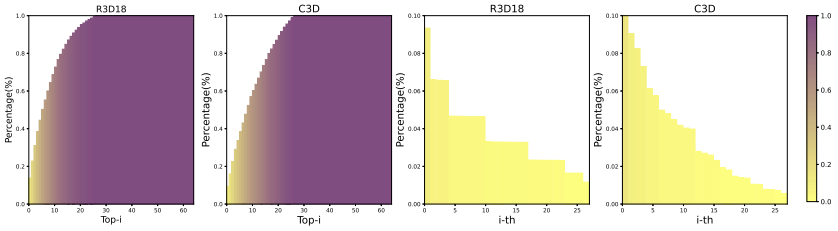


Figure 2: Visualisation results for singular values of layer5b in C3D and layer4b_2b in R3D-18. The two pictures on the left denote the proportion of the sum of the top-i singular values to the total sum of singular values. The two pictures on the right denote the proportion of i-th singular value to the total sum of singular values.

which leads to $\mathcal{O}_i \in \mathbb{R}^{c_{out} \times m^3}$[1]. For ease of the following representation, we sometimes use $\tilde{\odot}$ to represent the consecutive operations of element-wise product and summation over the output channel in Eq. (7), which therefore can be reformulated as:

$$\mathcal{O}_i = \big(\mathcal{G}_W \tilde{\odot} (\mathcal{I}_i T_I)\big) T_O. \tag{8}$$

Then, we analyze the over-increasing parameters by performing singular value decomposition (SVD) on the rearranged Winograd-domain weight matrix $\mathcal{G}_W \in \mathbb{R}^{c_{out}c_{in} \times t^3}$. In this fashion, $\mathcal{G}_W$ can be represented by the t^3-dimensional subspace as: $\mathcal{G}_W = \sum_{i=0}^{t^3-1} \sigma_i u_i v_i^T$, where σ_i and u_i/v_i indicate the i-th largest singular value and the corresponding left/right singular vector. Fig. 2 visualizes the singular values where two phenonema can be observed. First, among all the t^3 singular values, larger-magnitude ones are concentrated in the top-r^3 (r^3=27 in Fig. 2), which is exactly the number of weight elements in the spatial domain. Second, among the top-r^3 singular values, those in the front part are much larger than those in the back part. Such phenomena suggest the existence of over-parameterized weights in the Winograd domain. Therefore, a more efficient training way is urgent for 3D Winograd.

Given the pre-trained Winograd-domain weight $\mathcal{G}_W$, we denote the fine-tuned weight as $\mathcal{G}_W + \Delta\mathcal{G}_W$, where $\triangle\mathcal{G}_W \in \mathbb{R}^{c_{out}c_{in} \times t^3}$ denotes the updates from the initial pre-trained $\mathcal{G}_W$ to the eventual fine-tuned $\bar{\mathcal{G}}_W$. (Hu et al., 2021) showed that the update $\Delta\mathcal{G}_W$ is supposed to have a low "intrinsic rank" if the pre-trained weight $\mathcal{G}_W$ is over-parameterized. This indicates that fine-tuning in the Winograd domain raises attention to the main directions of the whole Winograd space. In light of this, we freeze the pre-trained Winograd weight $\mathcal{G}_W$ first, and then achieve low-rank update $\Delta\mathcal{G}_W$ by a low-rank decomposition $\Delta\mathcal{G}_W = \mathcal{G}_r\mathcal{G}_c$ where $\mathcal{G}_r \in \mathbb{R}^{c_{out}c_{in} \times k}$ and $\mathcal{G}_c \in \mathbb{R}^{k \times t^3}$ ($k \ll t^3$). Therefore, our low-rank Winograd transformation can be finally described as:

$$\mathcal{O}_i = \big((\mathcal{G}_W + \mathcal{G}_r\mathcal{G}_c) \tilde{\odot} (\mathcal{I}_i T_I)\big) T_O, \tag{9}$$

During training, we freeze $\mathcal{G}_W$, and upgrade $\mathcal{G}_r$ and $\mathcal{G}_c$ only. In this fashion, the amount of trainable parameters are reduced from $c_{out}c_{in}t^3$ to $c_{out}c_{in}k + kt^3$ in the Winograd domain. In order to train $\mathcal{G}_r$ and $\mathcal{G}_c$ more effectively, we initialize $\mathcal{G}_r$ by: $\mathcal{G}_{W r_{[:,i]}} = \alpha\sigma_i u_i$ and $\mathcal{G}_c$ by: $\mathcal{G}_{W c_{[i,:]}} = v_i^T$, where α is a scalar hyperparameter that controls the amplitude of the update.

3.3 Low-rank Oriented Sparse Granularity

In addition to trainable parameter reduction, we further attempt to decrease the computation cost from the element-wise product at inference time. By virtue of our low-rank Winograd transformation in Eq. (7), the element-wise multiplications can be lessened if most elements in the resulting $\mathcal{G}_W + \mathcal{G}_r\mathcal{G}_c$ are zeros. (Li et al., 2017b) imposed a sparse constraint upon $\mathcal{G}_W$ given that only $\mathcal{G}_W$ involves in the element-wise product of Eq. (3). However, sparse constraints often cause irregular weight matrix that receives little acceleration, a simple extension of which prevents the practicality in our settings of Eq. (7).

[1] A comprehensive derivation of Eq. (4), Eq. (5) and Eq. (6) as well as the formats of transformation matrices T_K, T_I and T_O can be referred to the appendix.

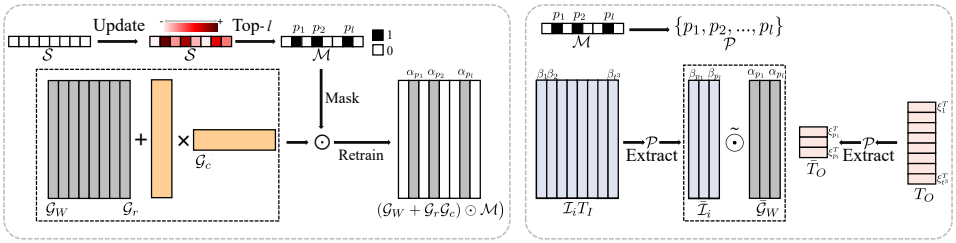


Figure 3: (a) The training workflow to our low-rank oriented sparse granularity. (b)The inference workflow after applying our low-rank oriented sparse granularity.

Sparse Granularity. Instead, in this paper, we devise a low-rank oriented sparsity to purchase effectual speedups. Our motive mainly stems from (Yu et al., 2019) that measured the element importance of 2D Winograd kernel using a score matrix and removed low-scored weights accordingly which leads to a distinct sparsity at different locations. We also intend to measure the weight importance, but at a more regular pattern. For ease of presentation, we denote our dense 3D Winograd weight as $\mathcal{G}_W + \mathcal{G}_r\mathcal{G}_c = [\alpha_1, \alpha_2, ..., \alpha_{t^3}] \in \mathbb{R}^{c_{out}c_{in} \times t^3}$ where each column $\alpha_i \in \mathbb{R}^{c_{in}c_{out} \times 1}$. Our sparse granularity consists of a single column position in $\mathcal{G}_W + \mathcal{G}_r\mathcal{G}_c$. In other words, pruning based on column locations results in removing the entire column elements.

The implementation of our low-rank oriented sparsity is of two stages including location scoring and weight retraining, respectively to filter out pruned target column positions and to recover the pruned model performance. In the former stage, we freeze the Winograd parameter $\mathcal{G}_W$ and initialize trainable parameters $\mathcal{G}_r$ and $\mathcal{G}_c$ as introduced in Sec. 3.2. Then, we introduce a score sequence $\mathcal{S} \in \mathbb{R}^{t^3}$, which is initialized with zeros, to evaluate location importance. Alike to Taylor pruning Molchanov et al. (2019), we opt to accumulate the position magnitude and gradient in each training iteration to be served as the values of score sequence:

$$\mathcal{S}^t = \begin{cases} 0, & t = 0, \\ \mathcal{S}^{t-1} + \frac{1}{c_{in}^2 c_{out}^2}\Big(\sum_{i=0}^{c_{in}c_{out}-1} |\mathcal{G}_W + \mathcal{G}_r\mathcal{G}_c|_{[i,:]}^t\Big) \odot \Big(\sum_{u=0}^{c_{in}c_{out}-1} |\frac{\partial \mathcal{L}}{\partial \mathcal{G}_r}\frac{\partial \mathcal{L}}{\partial \mathcal{G}_c}|_{[u,:]}^t\Big), & t > 1. \end{cases} \tag{10}$$

where the superscript t represents weight/magnitude and score sequence at the t-th training iteration. In this fashion, no additional parameters are introduced during determining $\mathcal{S}$.

With the score sequence $\mathcal{S}$, we can finally derive a location set $\mathcal{P} = \{p_1, p_2, ..., p_l\}$ that contains locations with their scores within the top-l largest, leading to a pruning rate of $(t^3 - l)/t^3$. Then, we can obtain a binary mask $\mathcal{M} \in \mathbb{R}^{t^3}$ as:

$$\mathcal{M}_{[i]} = \begin{cases} 1, & \mathcal{S}_{[i]} \in \mathcal{P}, \\ 0, & \text{Otherwise}. \end{cases} \tag{11}$$

The location scoring stage feeds back a fixed binary mask $\mathcal{M}$. In the stage of retraining, $\mathcal{M}$ is applied to remove low-scored column locations and we only need to fine-tune the trainable parameters $\mathcal{G}_r$ and $\mathcal{G}_c$ to recover the accuracy of pruned model. Therefore, the computation of each basic input tile $\mathcal{I}_i$ becomes:

$$\mathcal{O}_i = \Big(\big((\mathcal{G}_W + \mathcal{G}_r\mathcal{G}_c) \odot \mathcal{M}\big)\tilde{\odot}(\mathcal{I}_i T_I)\Big)T_O. \tag{12}$$

Fig. 3 gives an illustrative example of our low-rank oriented sparse granularity.

Speedup Mechanism. Unlike the irregular sparse patterns (Li et al., 2017b), our low-rank oriented sparse granularity result in a very regular sparse pattern. Therefore, it can well support practical speedups in the inference by simply involving non-zero columns with the multiplication in the code implementation. We detail it below.

Similar to $\mathcal{G}_W + \mathcal{G}_r\mathcal{G}_c = [\alpha_1, \alpha_2, ..., \alpha_{t^3}] \in \mathbb{R}^{c_{out}c_{in} \times t^3}$, we also rearrange the transformed input $\mathcal{I}_i T_I = [\beta_1, \beta_2, ..., \beta_{t^3}]$ where $\beta_i \in \mathbb{R}^{c_{out}c_{in}}$ denotes the i-th column input vector, and output

transformation matrix $T_O = [\xi_1^T; \xi_2^T; ...; \xi_{t^3}^T]$ where ξ_i is the i-th row of T_O. According to Eq. (12), the output can be derived as:

$$\begin{aligned}\mathcal{O}_i &= \big([\mathbf{0}, \alpha_{p_1}, \mathbf{0}, \alpha_{p_2}, \cdots, \alpha_{p_l}, \mathbf{0}] \tilde{\odot} \mathcal{V}_i\big) T_O \\ &= [\mathbf{0}, \alpha_{p_1} \tilde{\odot} \beta_{p_1}, \mathbf{0}, \alpha_{p_2} \tilde{\odot} \beta_{p_l}, \cdots, \alpha_{p_l} \tilde{\odot} \beta_{p_l}, \mathbf{0}] T_O \\ &= (\alpha_{p_1} \tilde{\odot} \beta_{p_1}) \xi_{p_1}^T + (\alpha_{p_2} \tilde{\odot} \beta_{p_2}) \xi_{p_2}^T + \cdots + (\alpha_{p_l} \tilde{\odot} \beta_{p_l}) \xi_{p_l}^T \\ &= (\bar{\mathcal{G}}_W \tilde{\odot} \bar{\mathcal{I}}_i) \bar{T}_O.\end{aligned} \tag{13}$$

Recall that $\mathcal{P} = \{p_1, p_2, ..., p_l\}$ contains column locations with their scores within the top-l largest. Therefore, in the inference stage, we only need to store a compact Winograd weight $\bar{\mathcal{G}}_W = [\alpha_{p_1}, \alpha_{p_2}, ..., \alpha_{p_l}] \in \mathbb{R}^{c_{out} c_{in} \times l}$ as well as the location set $\mathcal{P}$ to extract the corresponding column vectors $\bar{\mathcal{I}}_i = [\beta_{p_1}, \beta_{p_2}, ..., \beta_{p_l}] \in \mathcal{I}_i T_I$ and row vectors in $\bar{T}_O = [\xi_{p_1}^T; \xi_{p_2}^T; ...; \xi_{p_l}^T] \in T_O$. The cost of vector extraction is negligible compared to the large percentage of reduction on element-wise product, *i.e.*, $(t^3 - l)/t^3$.

4 Experimentation

4.1 Experiment Setup

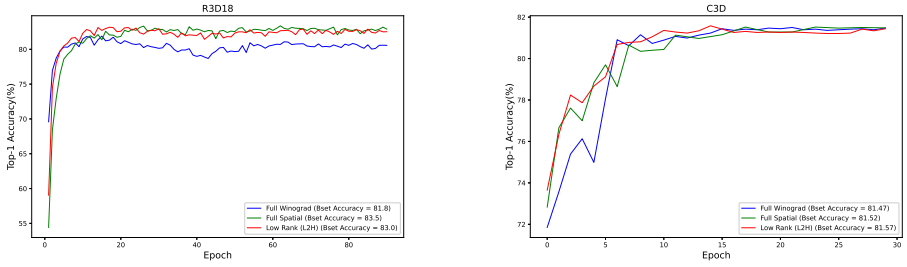
We conduct experiments on 3D CNN spatial models including 3D Resnet (Hara et al., 2018) (denoted as **R3D**) and **C3D** (Tran et al., 2015) that consist of plentiful layers with $3 \times 3 \times 3$ kernels and a stride of 1. For R3D, we replace all the 3D convolutional layers whose kernel size is $(3, 3, 3)$ and stride is 1 with 3D Winograd layers, result of which is denoted as **WR3D**. For C3D, we replace all the 3D convolutional layers except the first layer with 3D Winograd layers, result of which is denoted as **WC3D**. We first respectively pre-train R3D and C3D on the Kinetics dataset (Carreira & Zisserman, 2017) and Sports-1M dataset (Karpathy et al., 2014). Then, we conduct Winograd transformation upon the pre-trained spatial weights and inherit them to WR3D and WC3D as a constant of $\mathcal{G}_W$ (see the beginning of Sec. 3.2). Further, we fine-tune WR3D and WC3D upon the UCF101 dataset (Soomro et al., 2012). The fine-tuning consists of updating the low-rank matrices $\mathcal{G}_r$ and $\mathcal{G}_c$ as well as deriving their scoring sequences, and retraining sparse WR3D and WC3D to recover the performance. We refer the readers to AppendixA.2 for more details regarding the experiment setup.

4.2 Performance Results

4.2.1 Low-Rank Winograd Transformation

One of the supreme advantage of our low-rank Winograd transformation is that the two less storage-required matrices significantly reduce the trainable parameters. In what follows, we demonstrate our performance supremacy.

We choose R3D models with depth of 18 for dense training. Then, in Fig. 4(a), we compare the results of fine-tuning upon the spatial domain, vanilla Winograd domain and our proposed low-rank Winograd domain. As can be seen, the vanilla Winograd transformation performs the worst



(a) Fine-tuning upon pre-trained R3D model (b) Fine-tuning upon pre-trained C3D model

Figure 4: Fine-tuning performance comparison on UCF101 dataset.

Table 1: Results of different pruning patterns on UCF101 and HMDB51 datasets.

Model	Pruning Rate	Regularity	Pruning Pattern	FLOPs (G)	Accuracy (%)	
					UCF101	HMDB51
R3D-18	0%	-	-	5.31	83.5	55.5
R3D-18	40%	Irregular	Weight	4.57	82.9	55.6
R3D-18	40%	Regular	Filter	3.18	80.9	53.9
R3D-18	75%	Irregular	Weight	3.53	82.8	55.5
R3D-18	75%	Regular	Filter	1.32	75.0	45.9
WR3D-18	40%	Irregular	Weight	1.45	82.6	54.6
WR3D-18	40%	Regular	Low-rank **(Ours)**	1.06	83.0	55.0
WR3D-18	75%	Irregular	Weight	1.00	82.3	53.9
WR3D-18	75%	Regular	Low-rank **(Ours)**	0.44	80.6	51.9
R3D-34	0%	-	-	9.70	85.6	57.0
R3D-34	40%	Irregular	Weight	8.06	85.1	56.7
R3D-34	40%	Regular	Filter	5.81	82.4	53.7
R3D-34	75%	Irregular	Weight	5.18	84.8	56.6
R3D-34	75%	Regular	Filter	2.42	76.4	46.4
WR3D-34	40%	Irregular	Weight	2.62	85.3	55.9
WR3D-34	40%	Regular	Low-rank **(Ours)**	1.96	84.8	56.5
WR3D-34	75%	Irregular	Weight	1.71	84.6	56.2
WR3D-34	75%	Regular	Low-rank **(Ours)**	0.82	81.7	52.8

almost across the whole fine-tuning stage. Quantitatively, it causes 1.7% accuracy drops in the end, which well demonstrates our claim that more parameters from Winograd transformation do not always benefit model capacity but cause model redundancy. In contrast, our low-rank Winograd transformation manifests supreme performance in comparison with the vanilla version, even on par with the spatial model. The performance gains mostly come from the fact that we drive weight updating towards the main directions of the whole Winograd space. Fig. 4(b) continues the results on C3D. Similar to R3D, the vanilla Winograd transformation suffers the most performance drops, around 0.05% over the spatial model. On the contrary, by removing the redundancy, our low-rank transformation increases the accuracy of spatial model from 81.52% to 81.57%. These results again demonstrate the value of our method.

4.2.2 Low-Rank Oriented Sparse Granularity

In this subsection, we study the performance of low-rank oriented sparse granularity across R3D-18 and R3D-34 on UCF101 dataset. Besides, additional experiments are also performed on HMDB51 (Kuehne et al., 2011). For comparison, we show different pruning patterns in the spatial domain and Winograd domain across pruning rates of 40% and 75%. For the spatial models R3D-18 and R3D-34, the pruning granularity includes filters pruning that discards entire spatial filters, and weight pruning that removes individual spatial weights. As for the Winograd model WR3D-18 and WR3D-34, the pruning granularity includes weight pruning that imposes a sparse constraint upon the Winograd weights (Li et al., 2017b), as well as our proposed low-rank oriented sparse pattern. All the Winograd models are trained under the same settings described in Sec. A.2. We count the floating-point operations (FLOPs) of Winograd layers and corresponding convolutional layers, and report the model accuracy for comparison.

Table 1 shows the experimental results. Under the pruning rate of 75%, weight pruning in the spatial domain presents the best performance with top-1 accuracy losses of 0.7% in UCF101, 0.01% in HMDB51, 0.8% in UCF101 and 0.4% in HMDB51 when pruning R3D-18 and R3D-34, respectively. Compared with the results under the pruning rate of 40%, weight pruning in the spatial domain and the Winograd domain under the pruning rate of 75% both maintained good accuracy due to its fine-grained pruning granularity. However, weight pruning achieves the worst FLOPs reduction due to its irregular pruning rate across different layers. For example, under the pruning rate of 75%, the average sparsity of the fourth block of R3D-18 is 81.7%, while the sparsity of the first block is only 24.4%, and the FLOPs of the corresponding blocks are 0.45G and 2.77G, respectively. A similar situation occurs with weight pruning in the Winograd domain. Despite the fact that weight pruning can obtain accuracy under high pruning ratio, it cannot effectively reduce the operations of model. Compared with weight pruning, model pruned regularly in filter granularity can reduce more Flops, but suffers the highest performance degradation of 2.6% in UCF101 and 1.6% in HMDB51, and 3.2% in UCF101 and 3.3% in HMDB51 when pruning R3D-18 and R3D-34, respectively, under the pruning rate of 40%.

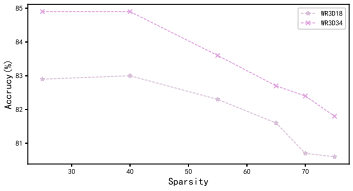


Figure 5: Results for R3D-18 R3D-34 pruned with our proposed low-rank oriented sparse granularity under different pruning rates.

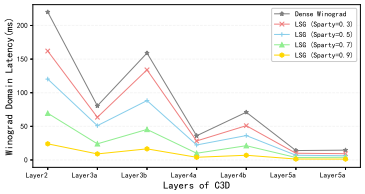


Figure 6: The inference latency in the Winograd domain of pruned WC3D with low-rank oriented sparse granularity.

Our proposed low-rank oriented sparse granularity can reduce more FLOPs operations on top of the performance gains from Winograd models and it is also capable of maintaining performance at a certain pruning rate. Under the pruning rate of 40%, WR3D-18 and WR3D-34 with low-rank oriented sparse granularity can obtain the similar accuracy performance compared with weight pruning in the spatial or Winograd domain, and WR3D-18 even obtain a better reuslt in UCF101. Compared with filter pruning, weight pruning in the spatial domain and Winograd domain, our method achieves 3.0×, 4.31× and 1.37×, 2.96×, 4.11× and 1.34× FLOPs reduction in R3D-18 and R3D-34, respectively. And compared with weight pruning in the spatial or Winograd domain under the pruning rate of 75%, the ratio of FLOPs reduction can be further expanded to 8.02× and 2.27×, 6.31× and 2.09× in R3D-18 and R3D-34, respectively, but it comes with a certain degree of accuracy loss. We further apply different pruning rate to sparsifying WR3D-18 and WR3D-34 with low-rank oriented sparse granularity. As it can be seen, our method is capable of maintaining performance within a pruning rate of 40%.

4.2.3 Acceleration Performance

Furthermore, we deploy WC3D model pruned with our proposed low-rank oriented sparse granularity on mobile phone with MediaTek 700 processor with total 8 to evaluate the acceleration capacity. We comparethe inference time in the Winograd domain of the pruned C3D model with several degrees of sparsity and the dense Winograd layer is regarded as the performance benchmark. For a fair comparison, the dense and our regularly pruned Winograd layer are both optimized by advanced SIMD (Single Instruction, Multiple Data). All experiments run on 100 rounds with 8 threads on CPU and the results are shown in Fig. 6. Although our sparse approach introduces a certain amount of computational overhead into the process of extracting the input data, the model is able to improve the speed of winograd domain operations by 1.30×, 1.80×, 3.35× and 9.42× under the sparsity of 30%, 50%, 70% and 90% respectively, which demonstrates that our pruning pattern effectively translates the sparsity into actual speedup.

4.3 Ablation STUDY

We further show the ablation studies for rank selection and indicators of location importance in A.4 and A.5, respectively.

5 Conclusion

Here, we have presented a novel low-rank Winograd transformation to reduce the over-parameterized issue in 3D CNNs. We decouple the original Winograd weight matrix into two less storage-required matrices, leading to remarkable trainable parameter reduction. The low-rank constraint well eliminates the redundant parameters and drives the updating towards main directions of the whole Winograd space. Consequently, our low-rank Winogrard transformation leads to better performance increase. In addition, we have also introduced a low-rank oriented sparse granularity to purchase effectual speedups. It models column-wise importance of the Winograd weight matrix and removes the low-scored ones. In this fashion, the sparsity tends to be more regular, which therefore better supports the practical acceleration in comparison with the existing irregular sparsity.

REFERENCES

Syed Asad Alam, Andrew Anderson, Barbara Barabasz, and David Gregg. Winograd convolution for deep neural networks: Efficient point selection. *ACM Transactions on Embedded Computing Systems (TECS)*.

Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6299–6308, 2017.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. 30, 2017.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. pp. 1135–1143, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. 28, 2015b.

Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6546–6555, 2018.

Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, pp. 293–299. IEEE, 1993.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2234–2240, 2018.

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1725–1732, 2014.

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)*, pp. 2711–2715. IEEE, 2019.

Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *2011 International conference on computer vision*, pp. 2556–2563. IEEE, 2011.

Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4013–4021, 2016.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 1989.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations (ICLR)*, 2019.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017a.

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

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1529–1538, 2020.

Mingbao Lin, Yuxin Zhang, Yuchao Li, Bohong Chen, Fei Chao, Mengdi Wang, Shen Li, Yonghong Tian, and Rongrong Ji. 1xn pattern for pruning convolutional neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2022.

Xingyu Liu and Yatish Turakhia. Pruning of winograd and fft based convolution algorithm. In *Proc. Convolutional Neural Netw. Vis. Recognit.*, pp. 1–7, 2016.

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

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2736–2744, 2017.

Liqiang Lu and Yun Liang. Spwa: An efficient sparse winograd convolutional neural networks accelerator on fpgas. In *Proceedings of the 55th Annual Design Automation Conference (ADA)*, pp. 1–6, 2018.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5058–5066, 2017.

Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

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

Lingchuan Meng and John Brothers. Efficient winograd convolution via integer arithmetic. In *International Conference on Learning Representations (ICLR)*, 2019.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11264–11272, 2019.

Wei Niu, Mengshu Sun, Zhengang Li, Jou-An Chen, Jiexiong Guan, Xipeng Shen, Yanzhi Wang, Sijia Liu, Xue Lin, and Bin Ren. Rt3d: Achieving real-time execution of 3d convolutional neural networks on mobile devices. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 35, pp. 9179–9187, 2021.

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4489–4497, 2015.

Kevin Vincent, Kevin Stephano, Michael Frumkin, Boris Ginsburg, and Julien Demouth. On improving the numerical stability of winograd convolutions. In *International Conference on Learning Representations Workshop (ICLRW)*, 2017.

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

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)*, pp. 254–261. IEEE, 2020.

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

A Appendix

A.1 Derivations of Eq. (4) to Eq. (6)

We start with a 2D case. To transform a convolution weight $g \in \mathbb{R}^{r\times r}$ to Winograd weight $g_W \in \mathbb{R}^{t\times t}$, we have $g_W = GgG^T$. Here, we introduce an equation derived by (Yu et al., 2019):

$$g_{W[j,k]} = \sum_{v=0}^{r-1}\sum_{w=0}^{r-1}(G_{[j,v]}G_{[k,w]}g_{[v,w]}). \tag{14}$$

Eq. (14) indicates that each element of the Winograd weight can be represented by elements of the convolution weight. This conclusion also applies in the 3D case.

Back to the 3D-version, the 3D convolution weight $\mathcal{G} \in \mathbb{R}^{r\times r\times r}$ is transformed into Winograd weight $\mathcal{G}_W \in \mathbb{R}^{t\times t\times t}$ by $\mathscr{R}(G\mathcal{G}G^T)G^T$. We further divide $\mathscr{R}(G\mathcal{G}G^T)G^T$ into three steps: $\mathcal{Q} = G\mathcal{G}G^T$,$\hat{\mathcal{Q}} = \mathscr{R}(\mathcal{Q})$, and $\mathcal{G}_W = \hat{\mathcal{Q}}G^T$.

For $\mathcal{Q} = G\mathcal{G}G^T$, we have:

$$\mathcal{Q}_{[i,j,k]} = \sum_{v=0}^{r-1}\sum_{w=0}^{r-1}(G_{[j,v]}G_{[k,w]}\mathcal{G}_{[i,v,w]}), \tag{15}$$

where $0 \le j,k \le t-1, 0 \le i \le r-1$. Then we rotate $\mathcal{Q}$ clockwise to $\hat{\mathcal{Q}}$, element of which can be further represented by:

$$\hat{\mathcal{Q}}_{[j,k,i]} = \sum_{v=0}^{r-1}\sum_{w=0}^{r-1}(G_{[j,v]}G_{[k,w]}\mathcal{G}_{[i,v,w]}). \tag{16}$$

After that, each element of $\mathcal{G}_W$ can be calculated by elements in $\hat{\mathcal{Q}}$ and G:

$$\mathcal{G}_{W[x,y,z]} = \sum_{u=0}^{r-1}(G_{[z,u]}\hat{\mathcal{Q}}_{[x,y,u]}) = \sum_{u=0}^{r-1}\sum_{v=0}^{r-1}\sum_{w=0}^{r-1}(G_{[x,v]}G_{[y,w]}G_{[z,u]}\mathcal{G}_{[u,v,w]}), \tag{17}$$

where $0 \le x,y,z \le t-1$.

We then rearrange $\mathcal{G}$ and $\mathcal{G}_W$ into vectors:

$$\begin{aligned}&\mathcal{G} \in \mathbb{R}^{r\times r\times r} \to \mathcal{G} = [a_1, a_2, \cdots, a_{r^3}], \mathcal{G} \in \mathbb{R}^{1\times r^3},\\ &\mathcal{G}_W \in \mathbb{R}^{t\times t\times t} \to \mathcal{G}_W = [b_1, b_2, \cdots, b_{t^3}], \mathcal{G}_W \in \mathbb{R}^{1\times t^3}.\end{aligned} \tag{18}$$

Let us describe Eq. (17) in another way where each position in $\mathcal{G}_W$ can be calculated by combination of coefficients of positions in $\mathcal{G}$:

$$b_i = a_1c_{1i} + a_2c_{2i} + \cdots + a_{r^3}c_{r^3i}. \tag{19}$$

Therefore, $\mathcal{G}_W$ and $\mathcal{G}$ can be related by matrix multiplication:

$$[b_1, b_2, \cdots, b_{t^3}] = [a_1, a_2, \cdots, a_{r^3}] \cdot \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1t^3} \\ c_{21} & c_{22} & \cdots & c_{2t^3} \\ \vdots & \vdots & \ddots & \vdots \\ c_{r^31} & c_{r^32} & \cdots & c_{r^3t^3} \end{bmatrix}, \tag{20}$$

and it can be further abbreviated as:

$$\mathcal{J}_K(\mathcal{G}) = \mathcal{G}T_K,\ T_K = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1t^3} \\ c_{21} & c_{22} & \cdots & c_{2t^3} \\ \vdots & \vdots & \ddots & \vdots \\ c_{r^3 1} & c_{r^3 2} & \cdots & c_{r^3 t^3} \end{bmatrix}. \tag{21}$$

Combining with Eq. (17), each element of T_K (kernel transformation matrix) can be calculated as follows:

$$\begin{aligned} T_{K[i,j]} &= G_{[x,v]} \cdot G_{[y,w]} \cdot G_{[z,u]}, \\ i &= r^2 u + rv + w, \\ j &= t^2 x + ty + z, \end{aligned} \tag{22}$$

where $0 \leq i \leq r^3 - 1, 0 \leq j \leq t^3 - 1, 0 \leq u, v, w \leq r - 1$ and $0 \leq x, y, z \leq t - 1$.

So far, we have complemented the derivation of Eq. (4). The above process can be also applied to derive Eq. (5) and Eq. (6) and acquire transformation matrices T_I and T_O.

A.2 IMPLEMENTATION DETAILS

The batch size for R3D and WR3D is set to 32 and it is 16 for C3D and WC3D. We pre-train R3D for 90 epochs with an initial learning rate $1e^{-3}$ decayed by 0.1 every 30 epochs. For WR3D, the rank is set by {2,4,8,12} for blocks. For WC3D, the rank is set by {1,1,2,4,8,12,12} for layers. To fine-tune WR3D, a total of 50 epochs are given, including 5 epochs for learning score sequence and updating low-rank matrices, and the rest 45 epochs for retraining. The initial learning rate is set to $5e^{-4}$ divided by 10 at epoch 15, 30. As for C3D, we directly borrow an existing model pre-trained from Sports-1M. The fine-tuning lasts for 30 epochs with a learning rate initialized with $1e^{-4}$ and changed to $1e^{-5}$ at the 10-th epoch. The first 6 epochs are used for score sequence and low-rank matrices. The remaining ones are used to retrain the WC3D. In addition, stochastic gradient descent (SGD) serves as our optimizer and cross-entropy loss is adopted to guide our model learning.

A.3 RANKS FOR 3D WINOGRAD LAYER

A.3.1 THE VALIDITY OF LOW-RANK WINOGRAD TRANSFORMATION

Is that necessary to search in the entire transformed Winograd space for a suitable solution for the downstream task? We have tried a simple test to answer above question. Firstly, we fine-tune a WC3D model with full Winograd weight. Then we rearrange updating $\triangle\mathcal{G}_W$ into $\triangle\mathcal{G}_W \in \mathbb{R}^{c_{out}c_{in} \times t^3}$ and perform singular value decomposition on it: $\triangle\mathcal{G}_W = USV^T$. By this way, $\triangle\mathcal{G}_W$ can be represented by t^3 subspaces: $\triangle\mathcal{G}_W = \sum_{i=0}^{t^3-1} \sigma_0 u_0 v_0^T$, where σ_i, u_i/v_i are the i-th singular value, left/right singular vector of $\triangle\mathcal{G}_W$. The magnitude of σ_i can be regard the importance of the subspace $u_i v_i^T$. We speculate how it would effect the model performance if only the few part of the subspaces were retained. Therefore, we directly test the accuracy of the model by adding $\sum_{i=0}^{r-1} \sigma_i u_i v_i^T$ to the pretrained weight, where r is the nunber of reserved subspaces. The result is shown in Table. 2. As it can be observed, when the number of reserved subspaces is reduced from 64 to 27, the effect of the model increased instead of decreasing. And the accuracy of the model does not decrease significantly until r = 8. This suggests that a part of the space introduced by the Winograd transformation may have hindered the training. The training process only need to focus on a portion of the subspaces. Our low-rank Winograd transformation is able to fit such process.

Table 2: The accuracy of WC3D when only the top-r subspaces of $\triangle\mathcal{G}_W$ are retained.

Rank	64	36	27	24	20	16	12	8	4	2	1
Accuracy (%)	81.47	81.45	81.49	81.45	81.42	81.36	81.28	81.31	80.78	80.57	80.28

A.4 EFFECT OF RANK SELECTION

To further explore the effect of ranks on model performance, we have tried different combinations of ranks. The ranks are set based on different layer blocks in WR3D, while WC3D are set based on different layers. Specifically, for a rank set $\mathbb{S} = \{r_1, \cdots, r_l\}$, r_i denotes the concrete rank for i-th Winograd layer or i-th block containing Winograd layer and l denotes the total number of Winograd layers/blocks. The results shown in Table. 3 indicate that a modest increase in ranks will improve the performance of the model, while deeper layers tend to require larger ranks than shallow layers.

Model	Rank Set	Trainable Parameters	Accuracy (%)
Set3	{2,2,2,2}	5.3M	80.0
Set2	{8,8,8,8}	21.3M	80.4
Set1	{12,12,12,12}	31.9M	80.2
Set4	{12,8,4,2}	7.4M	80.2
Set5	{2,4,8,12}	28.6M	80.6
Set1	{12,12,12,12,12,12,12}	46.9M	79.3
Set2	{2,2,2,2,2,2,2}	7.82M	78.2
Set3	{1,1,2,4,8,12,12}	34.67M	79.3
Set4	{12,12,8,4,2,1,1}	9.89M	77.9

Table 3: Pruning Results for WR3D and WC3D with different rank combinations. (with sparisity $= 0.75$)

A.5 INDICATORS OF LOCATION IMPORTANCE

Different assessment indicators can have very different effects during selecting locations. Table. 4 shows the pruning (sparsity $= 0.75$) results under different indicators of location importance. Gradient has a greater impact on the assessment of importance than magnitude, while the combination of the twos gives the best results.

Indicator	$\lvert M_{rc}\rvert$	$\lvert M_W + M_{rc}\rvert$	$\lvert G\rvert$	$\lvert M_{rc}\rvert \odot \lvert G\rvert$	$\lvert M_W + M_{rc}\rvert \odot \lvert G\rvert$
WR3D-18	73.6	76.9	79.8	80.4	80.6
WC3D	74.5	76.6	78.5	78.4	79.2

Table 4: Pruning Results (with sparisity $= 0.75$) for WR3D and WC3D under different indicators of location importance.