SCIENCE CHINA
Information Sciences

• Supplementary File •

Low-Rank Winograd Transformation for 3D Convolutional Neural Networks

Ziran QIN[1], Mingbao LIN[2], Huabin LIU[1], John SEE[3], Gui ZOU[1] & Weiyao LIN[1*]

[1]*School of Electronic, Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai* 200240, *China*
[2]*Skywork AI, Singapore* 118222, *Singapore*
[3]*School of Mathematical and Computer Sciences, Heriot-Watt University Malaysia, Putrajaya* 62200, *Malaysia*

Appendix A Related Work

Appendix A.1 Efficient Convolution Algorithms

Efficient convolution algorithms have been developed to improve the computational complexity of convolutional operations. The Winograd algorithm [11] and fast Fourier transform (FFT) [12] are two particularly prominent algorithms for accelerating CNNs. Pioneering work [2, 13] established FFT's potential as a computational kernel, which successfully accelerated deep neural networks. Furthering the contributions of Winograd's minimal filtering algorithm [11], Lavin et al. [1] adapted it for 2D CNNs, resulting in significant acceleration and superior performance over FFT for small convolutions. The Winograd algorithm was further extended by Wang et al. [14] to accelerate the inference of 3D CNNs. The effectiveness of the Winograd convolution is also evidenced by its integration into mainstream acceleration frameworks. Despite their ability to accelerate computations, these fast convolution algorithms inherently do not reduce the model size or parameter count, offering no storage savings.

Network pruning represents a technique orthogonal to efficient convolution algorithms. To further reduce computational demands, efficient convolution algorithms combined with network pruning could be considered. This approach integrates the computational efficiencies of algorithms like the Winograd algorithm with the model-size reduction benefits of network pruning, promising not only faster but also more compact neural network models.

Appendix A.2 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 [3, 9, 15], higher-order information [8, 16–18], 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 [19–21], activation sparsity [22], lasso regression-based channel selection [23], and rank of feature maps [24]. In contrast to weight pruning, filter pruning advantages in acceleration but causes more performance drops. Therefore, block pruning, where a block of weights is removed simultaneously, has received recent research focus [10, 25] 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 obtained by pruning [7]. To tackle this problem, some studies performed pruning operations directly in the Winograd domain to avoid the influence of Winograd kernel transformation.

Appendix A.3 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 the Winograd domain. Liu et al. [26] 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. Li et al. [5] introduced Winograd layers in exchange for the standard convolutional layers. The pruning and training are simultaneously conducted in the Winograd layers. Thus, the dimension inconsistency issue is eliminated and the sparsity in Winograd domain also increases. Liu et al. [6] introduced the ReLU operation to the Winograd domain to derive sparse transformed activations. It improves the possibility of sparse element-wise products in the Winograd domain. Yu et al. [7] specified that different locations of the Winograd layers contribute differently to the output activations. Despite the progress, these studies lead to hardware-unfriendly irregular

* Corresponding author (email: wylin@sjtu,edu.cn)

sparse patterns, causing imbalanced workloads among the data flows. To leverage the multiplication reduction from sparsity, some methods [27, 28] devised sparse patterns that benefit more from speedups on specialized hardware.

Despite these efforts to optimize Winograd-domain pruning for 2D CNNs, extending these methods to 3D CNNs introduces substantial challenges. Firstly, 3D Winograd transformation causes considerable parameter increase. Taking F(2, 3)-based Winograd algorithm as an example, a typical 3D convolutional kernel of shape $3 \times 3 \times 3$ is often replaced by a $4 \times 4 \times 4$ Winograd kernel, leading to 2.37× more parameters while it is only 1.78× for 2D case. More parameters from Winograd transformation do not always benefit model capacity but cause model redundancy. Also, the increase in trainable parameters poses a serious challenge to the cost of resources. Secondly, existing methods fail to accelerate Winograd transformation while conducting pruning in the Winograd domain. Similar to the weight pruning [3, 8, 9], prior implementations derive irregular sparse weight matrices, which received very limited speed gains since the irregularity barely takes advantage of vector processing architectures such as single instruction multiple data (SIMD), and poorly utilizes memory buses [10].

In light of these challenges, our study introduces a novel approach that significantly reduces the quantity of training parameters introduced by the Winograd transformation, while also proposing an acceleration-friendly sparse pattern for the Winograd model. Our method integrates the Winograd algorithm with network pruning, substantially reducing model computational demands and achieving tangible accelerations.

Appendix B Preliminaries

We first introduce some necessary preliminaries of the Winograd transformation for 3D CNNs. Consider a 3D convolution involving a kernel $\mathcal{G} \in \mathbb{R}^{C_o \times C_i \times r_d \times r_h \times r_w}$ and a 4D input data $\mathcal{I} \in \mathbb{R}^{C_i \times D_i \times H_i \times W_i}$, where C_o and C_i represent the output and input channels, (r_d, r_h, r_w) forms the kernel size and D_i, H_i, W_i correspond to the depth, height, and width of the input $\mathcal{I}$. This operation generates an output feature map $\mathcal{O} \in \mathbb{R}^{C_o \times D_o \times H_o \times W_o}$, where D_o, H_o, and W_o represent the depth, height, and width of $\mathcal{O}$, respectively.

The above 3D convolution can be decomposed into several basic convolutions and optimized using the 3D Winograd algorithm [14]. The input data $\mathcal{I}$ is disassembled into overlapping tiles $\mathcal{I}_1, \mathcal{I}_2, ...$, where each tile $\mathcal{I}_k \in \mathbb{R}^{C_i \times t_d \times t_h \times t_w}$ is a sub-matrix of $\mathcal{I}$. Each tile $\mathcal{I}_t$ is convoluted with $\mathcal{G}$, yielding a basic output tile $\mathcal{O}_t \in \mathbb{R}^{Co \times m_d \times m_h \times m_w}$ that is a sub-matrix of $\mathcal{O}$. Any two output tiles $\mathcal{O}_k$ and $\mathcal{O}_s$ are non-overlapping, and the output feature map $\mathcal{O}$ can be obtained by reassembling the output tiles in order. Notice in what follows, we introduce $r = r_d = r_h = r_w$ and $t = t_d = t_h = t_w$ for brevity since current networks often have a uniform kernel size such as $3 \times 3 \times 3$ across different dimensions, and also the basic output tile is characterized with $m = m_d = m_h = m_w$. We denote the disassembled $\mathcal{O}$, $\mathcal{I}$ as $\tilde{\mathcal{O}} \in \mathbb{R}^{T \times C_o \times m \times m \times m}$, $\tilde{\mathcal{I}} \in \mathbb{R}^{T \times C_i \times t \times t \times t}$, where $T = D_i H_i W_i / t^3$ is the number of disassembled tiles. For computational efficiency, the 3D Winograd transformation is applied to optimize the convolution operation as follows:

$$\tilde{\mathcal{O}}(k, n, :, :, :) = \mathscr{T}_O\Big(\sum_{c=0}^{C_i - 1} \mathscr{T}_K\big(\mathcal{G}(n, c, :, :, :)\big) \odot \mathscr{T}_I(\tilde{\mathcal{I}}(k, c, :, :, :)\Big), \tag{B1}$$

where $\odot$ denotes the element-wise product. In Eq.(B1), the kernel $\mathcal{G}(n, c, :, :, :)$ and input tiles $\tilde{\mathcal{I}}(k, c, :, :, :)$ are transformed into the Winograd domain of the same shape using the Winograd kernel transformation $\mathscr{T}_K(x) = (KxK^T)^R K^T$ and the Winograd input transformation $\mathscr{T}_I(x) = (B^T x B)^R B$, respectively. The Winograd-domain kernel and input tiles are then multiplied in an element-wise manner, the results of which are transformed back to the spatial domain using the Winograd output transformation $\mathscr{T}_O(x) = \big((A^T x A)^R A\big)^R$. Here, $(\cdot)^R$ notation denotes clockwise dimension rotation; K, B, and A are transformation matrices determined by $F(m \times m \times m, r \times r \times r)$. A thorough comprehension of Winograd transformation and the specific formats for transformation matrices can be referred to Lavin et al. [1] and Wang et al. [14].

In this study, we extend the concept of Winograd layer [5] to 3D. Specifically, we introduce a Winograd-domain weight $\mathcal{G}_W \in \mathbb{R}^{C_o \times C_i \times t \times t \times t}$ and initialize it with $\mathscr{T}_K(\mathcal{G})$ to replace original Winograd kernel transformation. In this way, we can reduce the huge computational cost in 3D Winograd transformation by sparsifying the $\mathcal{G}_W$ (*i.e.*, performing ***sparse training***). In the following, we refer to the model with Winograd layers as the *Winograd model* and the one with convolutional layers as the *spatial model* for short.

Appendix C Proposed Method

Appendix C.1 Low-Rank Winograd Transformation

The sparse training process of the Winograd model follows a two-step pipeline similar to the spatial model: (1) pruning unimportant weights and (2) retraining the model to restore its accuracy. However, the 3D Winograd layer will introduce a greater number of parameters for training compared to the 1D&2D counterparts, presenting significant challenges for sparse training. To solve this, we first propose a low-rank Winograd transformation to reduce the number of trainable parameters in the 3D Winograd layer.

For a better illustration, we first rearrange the Winograd kernel $\mathcal{G}_W \in \mathbb{R}^{C_o \times C_i \times t \times \times t}$, the input tiles $\tilde{\mathcal{I}} \in \mathbb{R}^{T \times C_i \times t \times t \times t}$, and the output tiles $\tilde{\mathcal{O}} \in \mathbb{R}^{T \times C_o \times m \times m \times m}$ into 2D matrices $G_W \in \mathbb{R}^{C_o C_i \times t^3}$, $\tilde{I} \in \mathbb{R}^{T C_i \times t^3}$, and $\tilde{O} \in \mathbb{R}^{T C_o \times m^3}$ accordingly. After rearrangement, the Winograd transformations can be expressed as matrix multiplications: $\mathscr{T}_K(x) = xT_K$,$\mathscr{T}_I(x) = xT_I$, and $\mathscr{T}_O(x) = xT_O$, where $T_K \in \mathbb{R}^{r^3 \times t^3}$, $T_I \in \mathbb{R}^{t^3 \times t^3}$, and $T_O \in \mathbb{R}^{t^3 \times m^3}$ are rearranged transformation matrices[1)].

1) A comprehensive derivation of rearranged transformation matrices T_K, T_I, and T_O is available in Appendix E.

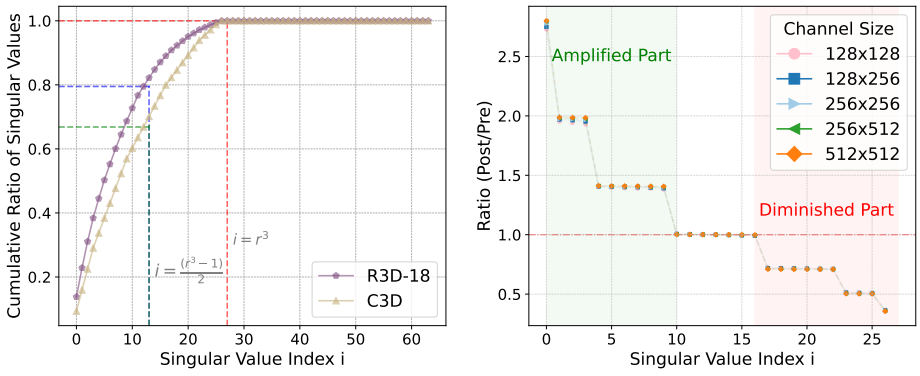


Figure C1 Singular value analysis. (a) Cumulative proportion of singular values of G_W. G_W is extracted from the last Winograd layer of R3D-18 and C3D. (b) Ratio of singular value changes post/pre Winograd transformation for the r^3 largest values. The G_W is obtained through the Winograd transformation of G, which is randomly initialized with different channel sizes.

The forward process of the rearranged 3D Winograd layer is defined as:

$$\tilde{O} = \left(G_W \tilde{\odot} V\right) T_O. \tag{C1}$$

For simplicity in description, we denote the result of Winograd input transformation $\tilde{I}T_I$ as V and use $\tilde{\odot}$ to represent the consecutive operations of element-wise product and summation over the output channel in Eq. (B1). The rearranged Winograd-domain weight matrix G_W is directly inherited from the rearranged pre-trained spatial-domain weight matrix $G \in \mathbb{R}^{C_o C_i \times m^3}$ transformed into the Winograd domain, *i.e.*, $G_W = GT_K$. To this end, the problem of pruning the 3D Winograd layer is converted into a task of sparsifying the weight matrix G_W.

To better guide the sparsifying process, we first conduct redundancy analysis on G_W using singular value decomposition (SVD). The terms in G_W occupy a t^3-dimensional subspace where it can be determined as $G_W = \sum_{i=0}^{t^3-1} \sigma_i \vec{u_i} \vec{v_i}^T$, where $\vec{u_i}$, $\vec{v_i}$ and σ_i represent the left, right singular vectors and the i-th largest singular value, respectively. The singular value analysis is further illustrated in Fig. C1.

In Fig. C1(a), we observe that the cumulative proportion of the first r^3 ($r^3 = 27$ in Fig. C1) singular values of G_W nearly encompasses the total sum of the singular values. Specifically, the first half within r^3 holds the majority of this cumulative proportion, which indicates that significant information is primarily concentrated in the subspace of the first half. Fig. C1(b) further shows that after Winograd kernel transformation, the leading singular values are amplified, while the trailing ones are diminished in the first r^3 singular values, indicating a shift in the importance of subspaces towards the head. These observations underscore the effects of over-parameterized weights in the Winograd domain, suggesting that training within the full domain is unnecessary and weight updates should concentrate along the principal directions within the entire Winograd space.

Given the pre-trained Winograd-domain weight G_W, we denote the fine-tuned weight as $G_W + \Delta G_W$, where $\triangle G_W \in \mathbb{R}^{C_o C_i \times t^3}$ denotes the updates from the G_W to the eventual fine-tuned $\bar{G}_W$. Recent studies [29,30] showed that the update ΔG_W is supposed to have a low "intrinsic rank" if the pre-trained weight G_W is over-parameterized. In light of this, we freeze the pre-trained dense Winograd weight G_W first and then achieve low-rank update ΔG_W by a low-rank decomposition $\Delta G_W = G_r G_c$ where $G_r \in \mathbb{R}^{C_o C_i \times s}$ and $G_c \in \mathbb{R}^{s \times t^3}$ ($s \ll t^3$). Therefore, our low-rank Winograd transformation can be finally described as:

$$\tilde{O} = \left((G_W + G_r G_c) \tilde{\odot} V\right) T_O. \tag{C2}$$

During sparse training, we freeze G_W, and only update G_r and G_c. In this fashion, the amount of trainable parameters is reduced from $C_o C_i t^3$ to $C_o C_i s + s t^3$ in the Winograd domain. To enhance the training efficiency of G_r and G_c, and put the focus on the primary direction of G_W, we initialize G_r by $G_r(:, i) = \alpha \sigma_i \vec{u_i}$ and G_c by $G_c(i, :) = \vec{v_i}^T$, where α is a scalar hyper-parameter that controls the amplitude of the update and is set to 0.1.

Appendix C.2 Low-Rank Oriented Sparse Granularity

The above low-rank Winograd transformation has significantly reduced the trainable parameters. Furthermore, we introduce the notion of low-rank oriented sparsity to decrease the computation cost resulting from the element-wise product during *inference*. By virtue of the transformation in Eq. (C2), the element-wise products can be reduced if most elements in the resulting $G_W + G_r G_c$ are zeros. Li et al. [5] imposed a sparse constraint upon G_W given that only G_W is involved in the element-wise product of Eq. (C1). However, sparse constraints often cause irregular weight matrices that receive little acceleration, a scenario which prevents the practicality of our settings in Eq. (C2). Instead, we propose the notion of low-rank oriented sparsity to obtain effectual speedups. Our motivation mainly stems from the work of Yu et al. [7] where the measurement of the element importance of 2D Winograd kernel using a score matrix and removal of low-scoring weights can lead to distinct sparsity at different locations. We also intend to measure the weight importance but in a more regular way. For ease of presentation, we denote our dense 3D Winograd weight as $G_W + G_r G_c = [\alpha_1, \alpha_2, ..., \alpha_{t^3}] \in \mathbb{R}^{C_o C_i \times t^3}$

where each column $\alpha_i \in \mathbb{R}^{C_o C_i \times 1}$. Our sparse granularity consists of a single column location in $G_W + G_r G_c$. In other words, pruning based on column locations results in removing the entire column elements.

Specifically, the process of sparse training to achieve low-rank oriented sparsity consists of two stages: location scoring and weight retraining. The aim of these stages is to filter out pruned target column locations and to recover the pruned model performance, respectively. In the former stage, we freeze the Winograd parameter G_W and initialize trainable parameters G_r and G_c as introduced in Sec. Appendix C.1. Then, we introduce a score sequence $S \in \mathbb{R}^{t^3}$, which is initialized with zeros, to evaluate location importance. Alike to Taylor pruning [38], we opt to accumulate the location magnitude and gradient in each training iteration to update the values of S:

$$\begin{aligned} S^t =& S^{t-1} + \frac{1}{C_i^2 C_o^2}\Big(\sum_{u=0}^{C_i C_o - 1} |G_W + G_r G_c|^{t-1}(u,:)\Big) \\ &\odot \Big(\sum_{v=0}^{C_i C_o - 1} |\frac{\partial \mathcal{L}}{\partial G_r}\frac{\partial \mathcal{L}}{\partial G_c}|^{t-1}(v,:)\Big), \end{aligned} \tag{C3}$$

where the superscript t denotes the training iteration. Here, no additional parameters are introduced while determining S. With the score sequence S, we can finally derive a location set $\mathcal{P} = \{p_1, p_2, ..., p_l\}$ that contains locations whose scores within the top-l largest, leading to a sparsity rate of $(t^3 - l)/t^3$. Then, we can obtain a binary mask $M \in \mathbb{R}^{t^3}$ as:

$$M(i) = \begin{cases} 1, & i \in \mathcal{P}, \\ 0, & \text{Otherwise.} \end{cases} \tag{C4}$$

The location scoring stage feeds back a fixed binary mask M. In the retraining stage, M is applied to remove low-scoring column locations and we only need to fine-tune the trainable parameters G_r and G_c in the pruned model. Therefore, the computation of input tiles becomes:

$$\tilde{O} = \Big(\big((G_W + G_r G_c) \odot M \big) \tilde{\odot} V \Big) T_O. \tag{C5}$$

After retraining, G_W is finally updated by $G_W = (G_W + G_r G_c) \odot M = [\vec{0}, \alpha_{p_1}, \vec{0}, \alpha_{p_2}, \cdots, \alpha_{p_l}, \vec{0}]$, where $\vec{0}$, α_p represent the pruned column and the reserved important column, respectively. Unlike Taylor pruning, which targets entire filters, our method prunes units based on specific positions within Winograd kernels, applying a consistent pruning pattern across corresponding positions in all kernels, thereby achieving a more regularized form of sparsity. Fig. 1(b) in the main article gives an illustrative example of the sparse training process of our method.

Appendix C.3 Inference Speedup and Compression Mechanism

Unlike the irregular sparse patterns of previous studies [5–7], the proposed low-rank oriented sparse granularity removes the entire column elements. Therefore, it can functionally support practical speedups in inference by performing multiplication operations on non-zero columns in the implementation. The pruned model can be easily compressed by storing the corresponding reserved columns. We illustrate the inference process after applying the low-rank oriented sparse granularity in Fig. 1(c) of the letter article, detailing the process as follows.

Similar to G_W, the transformed input tiles can be represented as t^3 columns: $V = [\beta_1, \beta_2, ..., \beta_{t^3}]$, where $\beta_i \in \mathbb{R}^{TC_i \times 1}$, and output transformation matrix can be represented as t^3 rows: $T_O = [\xi_1; \xi_2; ...; \xi_{t^3}]$ where $\xi_i \in \mathbb{R}^{1 \times m^3}$. Based on Eq. (C1), the output can be given as:

$$\begin{aligned} \tilde{O} &= \Big([\vec{0}, \alpha_{p_1}, \vec{0}, \alpha_{p_2}, \cdots, \alpha_{p_l}, \vec{0}] \tilde{\odot} V \Big) T_O \\ &= (\alpha_{p_1} \tilde{\odot} \beta_{p_1}) \xi_{p_1} + (\alpha_{p_2} \tilde{\odot} \beta_{p_2}) \xi_{p_2} + \cdots + (\alpha_{p_l} \tilde{\odot} \beta_{p_l}) \xi_{p_l} \\ &= (\bar{G}_W \tilde{\odot} \bar{V}) \bar{T}_O. \end{aligned} \tag{C6}$$

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 $\bar{G}_W = [\alpha_{p_1}, \alpha_{p_2}, ..., \alpha_{p_l}] \in \mathbb{R}^{C_o C_i \times l}$ as well as the location set $\mathcal{P}$ to extract the columns $\bar{V} = [\beta_{p_1}, \beta_{p_2}, ..., \beta_{p_l}] \in V$ and rows in $\bar{T}_O = [\xi_{p_1}; \xi_{p_2}; ...; \xi_{p_l}] \in T_O$ which are operated with the corresponding columns in G_W. The cost of data extraction is negligible compared to the large percentage of reduction on element-wise products, *i.e.*, (l/t^3). We follow the steps in 3D ResNet [32] and C3D [33] to generate training samples with 16-frame length, which are cropped to 112×112. For R3D and C3D, we replace all the 3D convolutional layers with $3 \times 3 \times 3$ kernels and a stride of 1 (except the first layer) with 3D Winograd layers ($t = 4$) and prune on 3D Winograd layers with our proposed low-rank Winograd transformation. The sparsity of 3D Winograd layers is defined as $(t^3 - l)/t^3$, where l denotes the number of remaining columns after pruning. The training epochs are set to 50, of which the first 2 epochs are used for location scoring and the remaining ones are used for weight retraining. The initial learning rate is 1e-4 for C3D and 5e-4 for R3D during location scoring and is divided by 10 for every 15 epochs during weight retraining. Stochastic gradient descent (SGD) serves as the optimizer and cross-entropy loss is adopted.

Table D1 Results of different pruning methods on UCF101, where * denotes our re-implementation and LR represents low-rank Winograd transformation, respectively. The experiment is conducted using C3D (baseline accuracy 81.6%) and R3D-18 (baseline accuracy 83.5%) and the speedup ratio is computed by GFLOPs reduction.

Model	Methods	Speedup	Accuracy(%)↓
C3D	FP* [38]	1.6×	1.6
	FP* [38]	3.7×	10.1
	DPR [39]	2.0×	3.3
	DPR [39]	4.0×	6.6
	RT3D [25]	2.6×	1.0
	RT3D [25]	3.6×	1.4
	MRP [40]	3.8×	0.4
	MRP [40]	4.3×	1.1
	Ours (w/o LR)	4.3×	0.8
	Ours (w LR)	**4.3**×	**0.6**
	Ours (w/o LR)	5.8×	1.3
	Ours (w LR)	**5.8**×	**0.9**
R3D-18	FP* [38]	1.6×	2.6
	FP* [38]	4.0×	8.5
	DPR* [39]	2.0×	3.4
	DPR* [39]	4.0×	5.0
	MRP [40]	3.2×	0.1
	MRP [40]	3.8×	1.2
	Ours (w/o LR)	4.3×	0.2
	Ours (w LR)	**4.3**×	**0.1**
	Ours (w/o LR)	5.0×	0.7
	Ours (w LR)	**5.0**×	**0.5**

Table D2 Results of different pruning patterns on UCF101 and HMDB51 datasets. The experiment is conducted using R3D-18 and R3D-34.

Model	Pruning Domain	Sparsity	Regularity	Pruning Pattern	Accuracy (%)↓	
					UCF101	HMDB51
R3D-18		-	-	-	- (83.5%)	- (55.5%)
	Spatial	0.40	Irregular	Weight	0.2	0.1
		0.40	Regular	Filter	2.6	2.6
		0.75	Irregular	Weight	0.7	0.2
		0.75	Regular	Filter	8.5	9.6
	Winograd	0.40	Regular	**Ours**	0.2	0.1
		0.75	Regular	**Ours**	2.9	3.6
R3D-34		-	-	-	- (85.6%)	- (57.0%)
	Spatial	0.40	Irregular	Weight	0.3	0.1
		0.40	Regular	Filter	3.2	3.3
		0.75	Irregular	Weight	0.8	0.4
		0.75	Regular	Filter	9.2	10.6
	Winograd	0.40	Regular	**Ours**	0.4	0.1
		0.75	Regular	**Ours**	1.9	4.2

Appendix D Experiments

Appendix D.1 Experimental Setup

Our methodology is applied on 3D CNN models including 3D ResNet [32] (denoted as R3D) and C3D [33], which consist of plentiful 3D convolution layers with $3 \times 3 \times 3$ kernels and a stride of 1. We use pre-trained R3D and C3D on the Kinetics dataset [34] and Sports-1M dataset [35] and fine-tune upon the UCF101 [36] and HMDB51 [37] datasets, results of which serve as the dense models.

Appendix D.2 Result Analysis

Pruning Performance. We compare the proposed method with state-of-art methods of pruning 3D CNNs in the different domains, such as FP (Filter Pruning) with Taylor-FO [38], RT3D (Group Pruning) [25], DPR (Stripe Pruning) [39] in

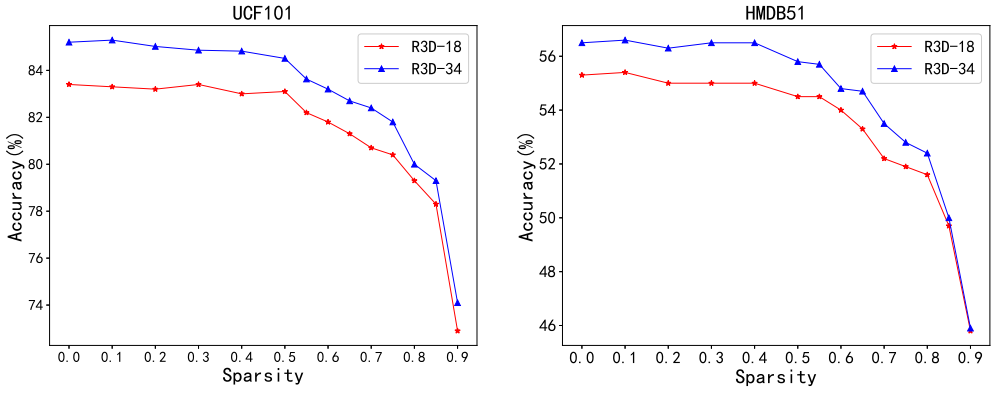


Figure D1 Results of R3D-18 and R3D-34 pruned with our proposed low-rank oriented sparse granularity under different sparsity levels. Experiments are conducted on UCF101(left) and HMDB51(right).

Table D3 Inference latency among different strategies. The experiment is conducted using C3D on a Redmi Note 10 mobile phone equipped with MediaTek 700 CPUs.

Acceleration Strategy	Sparsity	Accuracy(%)	CPU(ms)	Acceleration Ratio
Img2col	-	81.6	2970	-
Winograd	-	81.6	1211	2.5×
MRP	0.5	80.0	915	3.3×
	0.7	79.4	934	3.2×
Ours	0.0	81.6	1144	2.6×
	0.3	81.0	873	3.4×
	0.5	80.7	789	3.8×
	0.7	79.8	715	4.2×
	0.9	70.6	594	5.0×

the spatial domain, and MRP [40] in the Winograd domain. Table D1 displays their results. As can be seen, compared with pruning in the Winograd domain, spatial-domain pruning methods suffer the most performance degradation while achieving the same or even lower speedups. In contrast, our proposed Winograd-domain pruning pattern achieves the highest speedup ratio and even obtains the highest accuracy for both C3D and R3D-18. For example, when pruning C3D, our method achieves 5.8× speedups with an accuracy loss of 0.9% while MRP achieves 4.3× speedups with an accuracy loss of 1.1%, and RT3D achieves only 3.6× speedups with an accuracy loss of 1.4%. Similarly, our method also outperforms other methods by a large margin when pruning R3D-18. In addition, employing low-rank transformation in sparse training enhances the performance of the sparse models on both C3D and R3D-18 architectures, which emphasizes the effectiveness of the low-rank transformation.

We extend our comparison to include various pruning patterns and employ the same metrics as the baselines to evaluate the significance of different granularity units. The results are shown in Table D2. Due to the fine granularity of weight pruning, it achieves optimal performance. However, weight pruning leads to irregular sparse weights, which complicates achieving practical acceleration, especially on mobile devices. On the other hand, while filter pruning maintains regular granularity, it incurs significant performance penalties. In contrast, our method finds a middle ground: at a sparsity level of 0.40, it achieves performance comparable to weight pruning. Furthermore, it consistently outperforms filter pruning across all sparsity levels. The regularized granularity of our pruning approach not only enables practical acceleration but also provides a superior balance between performance and usability.

To further investigate the effect of sparsity in our proposed method, we evaluate low-rank oriented sparse granularity on R3D-18 and R3D-34 on UCF101 and HMDB51 datasets. Fig. D1 demonstrates the performance results of R3D-18 and R3D-34 under different sparsity ratios. As can be seen, our method is capable of maintaining performance drops within 1% when the sparsity ratio < 0.5. However, due to the extremely regular sparse granularity of our pruning pattern, our method degrades drastically if the sparsity ratio > 0.8.

Acceleration Performance. The acceleration capacity of our proposed method is evaluated on CPUs-based platforms. We first compare our proposed method against both Img2col and Winograd algorithms, which are commonly employed to accelerate convolutional operations. Additionally, we contrast the proposed approach with MRP [40], as it bears the closest resemblance. For a fair comparison, the inference code for our method and the compared methods are all implemented based on the mobile inference framework of Tencent ncnn[2)] and optimized by advanced SIMD (Single Instruction, Multiple Data). We respectively deploy C3D with different acceleration methods to obtain the end-to-end network inference latency on the Redmi Note10 platform. Table D3 highlights the impressive speed gains achieved by the proposed method. In comparison

2) https://github.com/Tencent/ncnn.

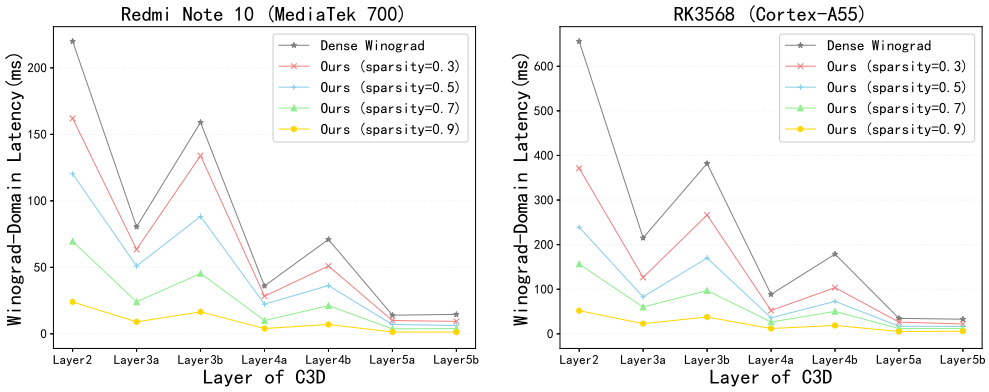


Figure D2 Comparison of the inference latency in Winograd domain between our low-rank Winograd convolution and dense Winograd convolution. The experiment is conducted using different layers of C3D on a Redmi Note 10 mobile phone equipped with a MediaTek 700 CPU (left), and the computing platform of RK3568 equipped with Cortex-A55 CPUs (right).

Table D4 Results of sparse models (pruned in the proposed sparse pattern) trained without and with low-rank Winograd transformation. The experiments are conducted using R3D-18 pruned with different sparsity levels on UCF101.

	Trainable Parameter (Mb)	Memory Overhead (Mb)	Sparsity				
			0.1	0.3	0.5	0.7	0.9
w/o Low-rank	170.0	732.6	83.3	83.3	82.9	80.7	73.4
w Low-rank	28.6	348.0	83.4	83.4	83.1	80.7	73.3

to Img2col, the Winograd algorithm [14] significantly reduces multiplications, achieving a 2.5× speedup. Besides, our approach further minimizes inference latency, attaining around 3.4× and 5.0× acceleration ratios at sparsity levels of 0.3 and 0.9, respectively. In contrast to MRP, our method offers enhanced performance due to our optimization which addresses redundancies in the 3D Winograd layers and allows integration of the advanced sparse training and granularity.

For Winograd convolution, the element-wise products in the Winograd domain occupy a vast majority of the inference time. The core acceleration mechanism of our method is to reduce a large number of operations in the Winograd domain by pruning Winograd-domain weights. Due to our proposed regular sparse pattern, extracting the corresponding arithmetic data only introduces a very small amount of overhead which makes sure that the sparsity obtained by pruning can be converted into actual speedups. Fig. D2 manifests the Winograd-domain inference latency of our method and its dense Winograd counterpart. As can be seen from the table, our proposed pruning pattern effectively translates the sparsity into actual speedups in the Winograd domain across different layers of C3D.

Reduction of Trainable Parameters and Memory Overhead. One of the major advantages of our low-rank Winograd transformation is that its space-efficient matrices significantly reduce both trainable parameters and memory overhead during pruning in the Winograd domain. Additionally, our low-rank Winograd transformation enhances the model's performance after pruning by eliminating redundant training parameters. Table D4 compares the results for pruning R3D-18 in our proposed sparse pattern with and without low-rank Winograd transformation on the UCF101 dataset. As can be observed, low-rank Winograd transformation offers a reduction factor of 5.94× *w.r.t.* trainable parameters and 2.11× *w.r.t.* memory overhead while achieving better performance when sparsity < 0.7. In the case of a large proportion (sparsity > 0.7) of redundant parameters pruned in the Winograd domain, low-rank Winograd transformation can still achieve nearly the same performance as the full Winograd-domain parameters. Results in Table D4 well validate the effectiveness of low-rank Winograd transformation for pruning the redundant Winograd-domain parameters and reducing the space requirement.

Appendix D.3 Ablation Study

Rank Selection. To further explore the effect of ranks on low-rank Winograd transformation, we have tried different settings of rank s for G_r and G_c. For R3D, we set ranks based on different blocks, and for C3D, we set ranks based on different layers. Specifically, for a rank set $\mathbb{S} = \{s_1, \cdots, s_l\}$, s_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. Table D5 shows the effect of rank setting. As can be observed from the displayed table, a modest increase in ranks will improve the performance of low-rank Winograd transformation and deeper layers tend to require larger ranks than shallow layers. Considering the overall effect, we finally choose $\mathbb{S} = \{2, 4, 8, 12\}$ for R3D model and $\mathbb{S} = \{1, 1, 2, 4, 8, 12, 12\}$ for C3D model to perform our experiments in this study.

Metric of Location Importance. In the stage of location scoring, the score sequence S updated by different metrics will produce different pruned columns, result of which greatly affects the performance of the sparse model. We compare several different metrics on C3D and R3D-18 models in Table D6. The results show that when selecting pruned columns, gradient-based score (S_{grad}) has a greater impact on the assessment of selecting locations than magnitude-based score ($|G_W|$ and $|G_r G_c|$), while the combination of the two gives the best results.

Table D5 Comparison of performance on R3D-18 and C3D pruned with different rank settings (with sparsity = 0.75), where ⋆ denotes our chosen rank setting. The experiment is conducted on UCF101.

Model	Rank Setting	Trainable Parameters	Accuracy (%)
R3D-18	{2,2,2,2}	5.3M	80.0
	{8,8,8,8}	21.3M	80.4
	{12,12,12,12}	31.9M	80.4
	{12,8,4,2}	7.4M	80.2
	{2,4,8,12}⋆	28.6M	80.6
C3D	{2,2,2,2,2,2,2}	7.8M	78.2
	{8,8,8,8,8,8,8}	31.3M	78.4
	{12,12,12,12,12,12,12}	46.9M	79.3
	{12,12,8,4,2,1,1}	9.9M	77.9
	{1,1,2,4,8,12,12}⋆	34.7M	79.3

Table D6 Comparison of performance on R3D-18 and C3D pruned with different indicators of location importance (with sparsity = 0.75), where $S_{grad} = \frac{\partial \mathcal{L}}{\partial G_r} \frac{\partial \mathcal{L}}{\partial G_c}$ and ⋆ denotes our chosen indicator. The experiment is conducted on UCF101.

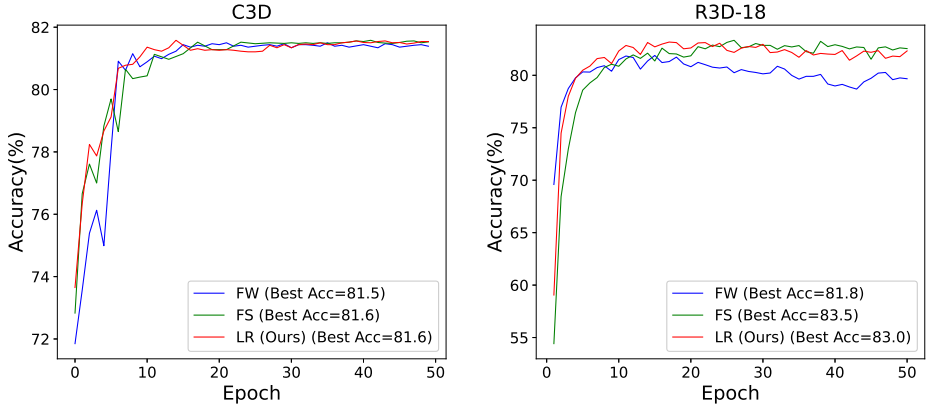
Model	Metric				
	$\lvert G_r G_c \rvert$	$\lvert G_W + G_r G_c \rvert$	$\lvert S_{grad} \rvert$	$\lvert G_r G_c \rvert \odot \lvert S_{grad} \rvert$	$\lvert G_W + G_r G_c \rvert \odot \lvert S_{grad} \rvert \star$
C3D	74.5	76.6	78.5	78.4	79.2
R3D-18	73.6	76.9	79.8	80.4	80.6

Appendix D.4 Effectiveness of the low-rank Winograd transformation

We further evaluate the effectiveness of the low-rank Winograd transformation during dense training and analyze its validity.

Low-Rank Winograd Transformation for Dense Training. We conduct experiments to evaluate the performance of low-rank transformation for dense training. The experiment is conducted by pre-training R3D and C3D on the Kinetics dataset [34] and Sports-1M dataset [35], and fine-tuning upon the UCF101 via different training methods. We compare the results of fine-tuning upon the full spatial domain, vanilla full Winograd domain, and our proposed low-rank Winograd domain.

Fig. D3(a) shows the fine-tuning results on R3D-18. As can be seen, the vanilla Winograd transformation performs the worst almost across the whole fine-tuning stage. Quantitatively, it results in a 1.7% drop in accuracy at the end, which well demonstrates our claim that increasing parameters introduced by 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. D3(b) continues the results on C3D. Similar to R3D, the vanilla Winograd transformation suffers the most performance drops, around 0.1% over the spatial model. On the contrary, by removing the redundancy, our low-rank transformation achieves the same performance as the spatial model. These results again demonstrate the value and the feasibility of our method.



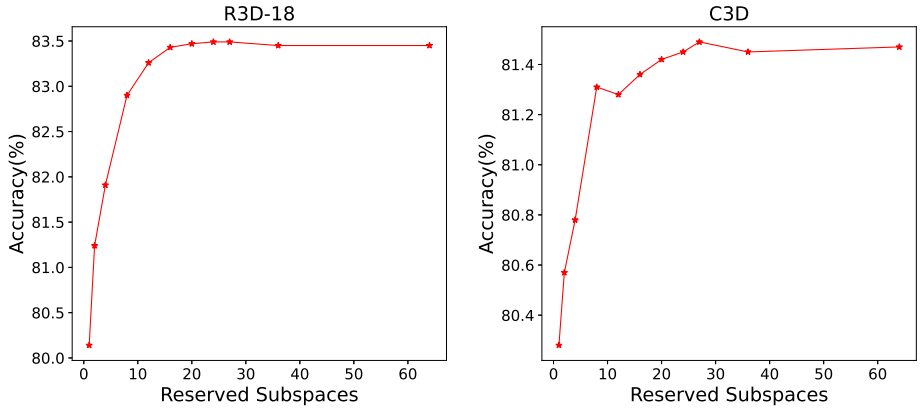
(a) R3D-18 (b) C3D

Figure D3 Comparison among fine-tuning in the full Winograd domain (FW), fine-tuning in the full spatial domain (FS), and fine-tuning in the low-rank Winograd domain (LR) on R3D-18 and C3D. The experiment is conducted on the UCF101 dataset.

Validity of Low-Rank Winograd Transformation. In this section, we analyze the validity of low-rank Winograd

transformation. We fine-tuned R3D-18 and C3D in the vanilla full Winograd domain on UCF101. After fine-tuning, we can get the updates $\triangle G_W$ from the pre-trained weight G_W to the eventual fine-tuned $\bar{G}_W$ by $\triangle G_W = \bar{G}_W - G_W$. And then we perform singular value decomposition on $\triangle G_W$. By this way, $\triangle G_W$ can be represented by t^3 subspaces: $\triangle G_W = \sum_{i=0}^{t^3-1} \triangle\sigma_i \triangle\vec{u_i} \triangle\vec{v_i}^T$, where $\triangle\sigma_i$, $\triangle\vec{u_i}/\triangle\vec{v_i}$ are the i-th singular value, left/right singular vector of $\triangle G_W$. The magnitude of $\triangle\sigma_i$ can be regarded as the importance of the subspace $\triangle\vec{u_i}\triangle\vec{v_i}^T$.

To explore how the model performance would be affected if only a few parts of the subspace were retained, we directly test the accuracy of the model by adding $\sum_{i=0}^{s-1} \triangle\sigma_i \triangle\vec{u_i} \triangle\vec{v_i}^T$ to the pre-trained weight G_W, where s is the number of reserved subspaces. The result is shown in Fig. D4. As can be observed, the accuracy of the model does not decrease significantly until $s < 8$, when s reduces from 64 to 27, the model accuracy even increases. This suggests that a large part of the space introduced by the Winograd transformation may have hindered the training of the Winograd model. The training process only needs to focus on a small portion of the subspaces, which just fits our proposed low-rank Winograd transformation.



(a) R3D-18 (b) C3D

Figure D4 The accuracy of R3D-18 and C3D on UCF101 when only the top-s subspaces of $\triangle G_W$ are reserved.

Table D7 Experimental results of the proposed method on X3D. The experiment is conducted using X3D-S on the Kinetics dataset (baseline accuracy 72.9%). The speedup ratio is computed by GFLOPs reduction.

Model	Speedup	Accuracy(%)↓
X3D	2.2×	0.0
	2.7×	0.1

Table D8 Comparison of computational complexity required for convolution designs under varying input sizes.

Kernel Type	Input Size ($D \times H \times W$)		
	8×32×32	8×64×64	8×128×128
Standard 3D	145.8K	583.2K	2.33M
(2+1) D	81.0K	324.0K	1.30M
Winograd	43.2K	172.8K	691.2K
Ours (Sparsity = 0.4)	25.9K	103.7K	414.7K
Ours (Sparsity = 0.7)	13.0K	51.8K	207.4K

Appendix D.5 More Information

To demonstrate the broader applicability and generalization capability of our method beyond standard 3D convolution kernels (with demonstrated effectiveness and practicality in Appendix D.2), we extend our evaluation to networks employing 3D convolutions with non-uniform stride patterns and present a comprehensive analysis of computational complexity across different convolution configurations. Specifically, we evaluated our method on X3D [41], which features a distinctive implementation of 3D convolutions with different temporal and spatial strides (spatial stride = 2, temporal stride = 1). As shown in Table D7, our method demonstrates robust performance on X3D, achieving a 2.2× speedup with no accuracy degradation and a 2.7× speedup with minimal accuracy loss (0.1%). These results validate the effectiveness of our approach across different convolution configurations.

We further conducted a detailed computational complexity analysis for different convolution designs under varying input sizes, using multiplication operations as the primary metric for computational complexity. We compare our optimized 3D kernel with three baseline kernel types: standard 3D convolution ($3 \times 3 \times 3$, stride = 1), which is widely adopted for three-dimensional data processing; Winograd-optimized 3D convolution; and (2+1)D convolution, which factorizes 3D convolution into spatial ($1 \times 3 \times 3$) and temporal ($3 \times 1 \times 1$) components for improved efficiency. For fair comparison, all computation costs were analyzed excluding input and output channels. As illustrated in Table D8, although (2+1)D convolution reduces computational cost through decomposition, Winograd-based 3D convolution achieves even greater efficiency, requiring approximately 70% fewer operations than standard convolution across all input sizes. Our method further reduces computational complexity by applying pruning techniques to Winograd convolution, with the reduction scaling proportionally with input dimensions. We demonstrate this with two carefully selected sparsity levels: sparsity = 0.4, which represents an optimal trade-off point that fully preserves model performance, and sparsity = 0.7, which aggressively reduces kernel operations for deployment scenarios where moderate accuracy trade-offs are acceptable. These results demonstrate the superior efficiency of our optimized kernel design.

Appendix E Derivations of the Rearranged Winograd Transformation and Transformation Matrices.

To derive the rearranged Winograd transformation in Eq. (C1), we need to derive the form of the Winograd transformations after rearranging $\mathcal{G}$, $\tilde{\mathcal{I}}$. We first introduce the conclusion, after rearranging $\mathcal{G}$, $\tilde{\mathcal{I}}$, the Winograd transformations $\mathscr{T}_K(\cdot)$, $\mathscr{T}_I(\cdot)$, and $\mathscr{T}_O(\cdot)$ are supposed to be modified accordingly:

$$\begin{aligned}
&\mathscr{T}_K(\mathcal{G}) = (K\mathcal{G}K^T)^R K^T,\ \mathcal{G} \in \mathbb{R}^{C_o \times C_i \times r \times r \times r} \rightarrow \\
&\mathscr{T}_K(G) = GT_K,\ G \in \mathbb{R}^{C_o C_i \times r^3}, && \text{(E1)} \\
&\mathscr{T}_I(\tilde{\mathcal{I}}) = (B^T\tilde{\mathcal{I}}B)^R B,\ \tilde{\mathcal{I}} \in \mathbb{R}^{T \times C_i \times t \times t \times t} \rightarrow \\
&\mathscr{T}_I(\tilde{I}) = \tilde{I}T_I,\ \tilde{I} \in \mathbb{R}^{TC_i \times t^3}, && \text{(E2)} \\
&\mathscr{T}_O(\mathcal{V}) = \left((A^T\mathcal{V}A)^R A\right)^R,\ \mathcal{V} \in \mathbb{R}^{T \times C_o \times t \times t \times t} \rightarrow \\
&\mathscr{T}_O(V) = VT_O,\ V \in \mathbb{R}^{TC_o \times t^3}, && \text{(E3)}
\end{aligned}$$

where T_K, T_I, and T_O are transformation matrices that we need to derive.

We start with the 2D version. Giving a convolution weight $g \in \mathbb{R}^{r \times r}$ and it is transformed into Winograd weight

$g_W \in \mathbb{R}^{t\times t}$ by $g_W = KgK^T$. Here, we introduce an equation derived by Yu et al. [7]:

$$g_W(j,k) = \sum_{v=0}^{r-1}\sum_{w=0}^{r-1} K(j,v)K(k,w)g(v,w). \tag{E4}$$

Eq. (E4) indicates that each element of the Winograd weight can be represented by elements of the convolution weight. This conclusion still holds 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 $(K\mathcal{G}K^T)^R K^T$. We further divide $(K\mathcal{G}K^T)^R K^T$ into three steps: $\mathcal{Q} = K\mathcal{G}K^T$,$\hat{\mathcal{Q}} = (\mathcal{Q})^R$, and $\mathcal{G}_W = \hat{\mathcal{Q}}K^T$. For $\mathcal{Q} = K\mathcal{G}K^T$, we have:

$$\mathcal{Q}(i,j,k) = \sum_{v=0}^{r-1}\sum_{w=0}^{r-1} K(j,v)K(k,w)\mathcal{G}(i,v,w), \tag{E5}$$

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

$$\hat{\mathcal{Q}}(j,k,i) = \sum_{v=0}^{r-1}\sum_{w=0}^{r-1} K(j,v)K(k,w)\mathcal{G}(i,v,w). \tag{E6}$$

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

$$\begin{aligned}\mathcal{G}_W(x,y,z) &= \sum_{u=0}^{r-1} K(z,u)\hat{\mathcal{Q}}(x,y,u)\\ &= \sum_{u=0}^{r-1}\sum_{v=0}^{r-1}\sum_{w=0}^{r-1} K(x,v)K(y,w)K(z,u)\mathcal{G}(u,v,w),\end{aligned} \tag{E7}$$

where $0 \leqslant x,y,z \leqslant t-1$. We then rearrange $\mathcal{G}$ and $\mathcal{G}_W$ into vectors:

$$\begin{aligned}&\mathcal{G} \in \mathbb{R}^{r\times r\times r} \to G = [a_1, a_2, \cdots, a_{r^3}], G \in \mathbb{R}^{1\times r^3},\\ &\mathcal{G}_W \in \mathbb{R}^{t\times t\times t} \to G_W = [b_1, b_2, \cdots, b_{t^3}], G_W \in \mathbb{R}^{1\times t^3}.\end{aligned} \tag{E8}$$

Let us describe Eq. (E7) in another way where each position in $\mathcal{G}_W$ (G_W) can be calculated by the combination of coefficients of positions in $\mathcal{G}$ (G):

$$b_i = a_1c_{1i} + a_2c_{2i} + \cdots + a_{r^3}c_{r^3i}. \tag{E9}$$

Therefore, G_W and G can be related by matrix multiplication:

$$[b_1, b_2, \cdots, b_{t^3}] = [a_1, a_2, \cdots, a_{r^3}] \cdot \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1t^3}\\ c_{21} & c_{22} & \cdots & c_{2t^3}\\ \vdots & \vdots & \ddots & \vdots\\ c_{r^31} & c_{r^32} & \cdots & c_{r^3t^3}\end{bmatrix}, \tag{E10}$$

and it can be further abbreviated as:

$$G_W = \mathscr{T}_K(G) = GT_K,\ T_K = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1t^3}\\ c_{21} & c_{22} & \cdots & c_{2t^3}\\ \vdots & \vdots & \ddots & \vdots\\ c_{r^31} & c_{r^32} & \cdots & c_{r^3t^3}\end{bmatrix}. \tag{E11}$$

Combining with Eq. (E7), each element of T_K (kernel transformation matrix) can be calculated as follows:

$$\begin{aligned}T_K(i,j) &= K(x,v)\cdot K(y,w)\cdot K(z,u),\\ i &= r^2u + rv + w,\\ j &= t^2x + ty + z,\end{aligned} \tag{E12}$$

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

So far, we have complemented the derivation of Eq. (E1). The above process can also be applied to derive Eq. (E2) and Eq. (E3) and acquire transformation matrices T_I and T_O. Finally, we obtain Eq. (E13) by substituting Eq. (E1), Eq. (E2), and Eq. (E3) into Eq. (3) of Sec.3.1:

$$\tilde{O}(kn,:) = \Big(\sum_{c=0}^{C_i-1} G(C_in+c,:)T_K \odot \tilde{I}(kC_i+c,:)T_I\Big)T_O. \tag{E13}$$

We denote the Winograd input transformed result $\tilde{I}T_I$ as V and use $\tilde{\odot}$ to represent the consecutive operations of element-wise product and summation over the output channel in Eq. (E13) for ease of the following representation. Eq. (E1) can be simplified as:

$$\tilde{O} = \left(GT_K \tilde{\odot} V\right) T_O. \tag{E14}$$

Finally, by using the rearranged Winograd-domain weight $\mathcal{G}_W \in \mathbb{R}^{Co \times C_i \times t \times t \times t}$ to directly perform element-wise products with the Winograd-domain input tiles, we complete the derivation of Eq. (C1).

Appendix F Discussion

Through our experimental results, we have demonstrated that the low-rank Winograd transformation can successfully extend the Winograd algorithm to 3D dimensions and effectively combine it with network pruning to significantly reduce the computational load of 3D CNNs and accelerate them in real-edge scenarios. However, there are areas for improvement or expansion in our current method, which we aim to address in future work:

- Although our current work has achieved significant reductions in parameters and computational costs of 3D CNNs, it primarily focuses on specific kernel sizes. Moving forward, we plan to explore how to further optimize this transformation method to accommodate a broader range of kernel sizes and different types of deep learning models, such as those with larger kernels.
- We have validated our proposed method on mobile CPUs, demonstrating the practical acceleration of 3D CNNs. Theoretically, this method is also applicable to hardware devices such as GPUs and FPGAs. A future effort worth pursuing is the development of operators for these hardware platforms to enable broader deployment and application of our proposed method.
- While our proposed pruning pattern is more effective for weights in the Winograd domain than in the spatial domain, we believe that applying this pruning pattern to network layers with location-sensitive characteristics, such as attention layers, might be a promising research direction.

We believe our method presents great possibilities and opportunities for optimizing 3D CNNs and offers viable approaches for deploying efficient 3D CNNs across various platforms. We plan to explore more possibilities in the field of model optimization and acceleration in the future.

References

1 Lavin A, Gray S. Fast algorithms for convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, 4013–4021
2 Mathieu M, Henaff M, LeCun Y. Fast training of convolutional networks through ffts. 2013. ArXiv:1312.5851
3 Frankle J, Carbin M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: Proceedings of the International Conference on Learning Representations (ICLR), 2018
4 Luo J H, Wu J X, Lin W Y. Thinet: A filter level pruning method for deep neural network compression. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2017, 5058–5066
5 Li S, Park J, Tang P T P. Enabling sparse winograd convolution by native pruning. 2017. ArXiv:1702.08597
6 Liu X Y, Pool J, Han So, et al. Efficient sparse-winograd convolutional neural networks. In: Proceedings of the International Conference on Learning Representations (ICLR), 2018
7 Yu J C, Park J S, Naumov M. Spatial-winograd pruning enabling sparse winograd convolution. 2019. ArXiv:1901.02132
8 LeCun Y, Denker J, Solla S. Optimal brain damage. In: Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 1989, 598–605
9 Han S, Pool J, Tran J, et al. Learning both weights and connections for efficient neural network. In: Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2015, 1135–1143
10 Lin M B, Zhang Y X, Li Y C, et al. 1xN Pattern for Pruning Convolutional Neural Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2022
11 Winograd S. Arithmetic complexity of computations. Siam, 1980
12 Cooley J W, Tukey J W. An algorithm for the machine calculation of complex Fourier series. Mathematics of computation, 1965, 297–301
13 Vasilache N, Johnson J, Mathieu M, et al. Fast convolutional nets with fbfft: A GPU performance evaluation. 2014 ArXiv:1412.7580
14 Wang Z L, Lan Q, He H J, et al. Winograd algorithm for 3D convolution neural networks. International Conference on Artificial Neural Networks (ICANN), 2017, 609–616
15 Han S, Mao, H Z, Dally W J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In: Proceedings of the International Conference on Learning Representations (ICLR), 2016
16 Hassibi B, Stork D G, Wolff G J. Optimal brain surgeon and general network pruning. In: Proceedings of the IEEE International Conference on Neural Networks (ICNN), 1993, 293–299
17 Dong X, Chen S Y, Pan S. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In: Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2017, 4860–4874
18 Lee N, Ajanthan T, Torr P H S. Snip: Single-shot network pruning based on connection sensitivity. In: Proceedings of the International Conference on Learning Representations (ICLR), 2019
19 Li H, Kadav A, Durdanovic I, et al. Pruning filters for efficient convnets. In: Proceedings of the International Conference on Learning Representations (ICLR), 2017
20 Liu Z, Li J, Shen Z Q, et al. Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2017, 2736–2744
21 He Y, Kang G L, Dong X Y, et al. Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2018, 2234–2240
22 Hu H Y, Peng R, Tai Y W, et al. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. 2016. arXiv:1607.03250
23 He Y H, Z X Y, Sun J. Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2017, 1389–1397
24 Lin M B, J R R, Wang Y, et al. Hrank: Filter pruning using high-rank feature map. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2020, 1529–1538
25 Niu W, Sun M H, Li Z G, et al. RT3D: Achieving Real-Time Execution of 3D Convolutional Neural Networks on Mobile Devices. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2021, 9179–9187
26 Liu X Y, Turakhia Y. Pruning of winograd and FFT based convolution algorithm. In: Proceedings of the Convolutional Neural Networks for Visual Recognition, 2016, 1–7
27 Lu L Q, Liang Y. SpWA: An efficient sparse winograd convolutional neural networks accelerator on FPGAs. In: Proceedings of the Annual Design Automation Conference (ADA), 2018, 1–6
28 Yang T, Liao Y K, Shi J P, et al. A Winograd-based CNN accelerator with a fine-grained regular sparsity pattern. In: Proceedings of the International Conference on Field-Programmable Logic and Applications (FPL), 2020, 254–261
29 Hu E J, Shen Y L, Wallis P, et al. Lora: Low-rank adaptation of large language models. In: Proceedings of the International Conference on Learning Representations (ICLR), 2022
30 Li Y C, Luo F, Tan C Q, et al. Parameter-Efficient Sparsity for Large Language Models Fine-Tuning. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2022, 4223–4229
31 Molchanov P, Mallya A, Tyree S, et al. Importance estimation for neural network pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, 11264–11272
32 Hara K, Kataoka H, Satoh Y. 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), 2018, 6546–6555
33 Tran D, Bourdev L, Fergus R, et al. Learning spatiotemporal features with 3d convolutional networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2015, 4489–4497
34 Carreira J, Zisserman A. Quo v, action recognition? a new model and the kinetics dataset. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, 6299–6308
35 Karpathy A, Toderici G, Shetty S, et al. Large-scale video classification with convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, 1725–1732
36 Soomro K, Zamir A R, Shah M. UCF101: A dataset of 101 human actions classes from videos in the wild. 2012. ArXiv:1212.0402
37 Kuehne H, Jhuang H, Garrote E, et al. HMDB: a large video database for human motion recognition. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2011, 2556–2563
38 Molchanov P, Tyree S, Karras T, et al. Pruning convolutional neural networks for resource efficient inference. In: Proceedings of the International Conference on Learning Representations (ICLR), 2017
39 Zhang Y X, Wang H A, Luo Y, et al.. Three-dimensional convolutional neural network pruning with regularization-based method. In: Proceedings of the IEEE International Conference on Image Processing (ICIP), 2019, 4270–4274
40 Qin Z R, He H Y, Lin W Y. 3D Winograd Layer with Regular Mask Pruning. In: Proceedings of the IEEE International Conference on Multimedia and Expo Workshops (ICMEW), 2022, 1–6
41 Feichtenhofer C. X3D: Expanding Architectures for Efficient Video Recognition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, 203–213