Enabling Sparse Winograd Convolution by Native Pruning

Sheng Li [*1] Jongsoo Park [*1] Ping Tak Peter Tang [*1]

Abstract

Sparse methods and the use of Winograd convolutions are two orthogonal approaches each of which significantly accelerates convolution computations in modern CNNs. Sparse Winograd merges these two and thus has the potential to offer a combined performance benefit. Nevertheless, training convolution layers so that the resulting Winograd kernels are sparse has not hitherto been very successful. We introduce here a novel construction of Sparse Winograd by introducing a substitute for the convolution layer that we call, naturally, the Winograd layer. Such a construction allows us to prune parameters natively in the Winograd domain. This paper presents the technical details related to the Winograd layer and our train-and-prune procedures. We achieved more than 90% sparsity in the Winograd parameters while maintaining the original accuracy of AlexNet on ImageNet dataset. Our detailed performance model also projects a speedup over convolution by dense Winograd kernels in excess of twofold.

1. Introduction

Convolution neural networks (CNN) have achieved undisputed success in many practical applications. These deep neural networks typically contain multiple layers, many (though not all) of which perform the namesake computation of convolution. A convolution layer is an architecture whose connection between an input and output tensor is via a number of convolution kernels, and the basic arithmetic operations are that of multiply-accumulate. Because over 90% of the computation during inference and training of recent CNN designs is in convolutions (Krizhevsky et al., 2012; Szegedy et al., 2015), different compute kernels have been devised to speed up this core operation. Recently, kernels based on transform methods such as FFT (Mathieu et al., 2013; Vasilache et al., 2015) or Winograd transformation (Winograd, 1980; Lavin & Gray, 2016) have proved to be successful. For the typical small convolution sizes (e.g., 3x3) that arise in CNNs, the Winograd-kernel approach is more effective and has demonstrated more than twofold speed up over well-implemented spatial convolution approaches.

The pursuit of faster convolution never stops. Besides the transform methods, sparse methods with pruned/compressed/sparse models demonstrate remarkable potential to accelerate convolutions while reducing model size (LeCun et al., 1989; Lebedev & Lempitsky, 2016; Liu et al., 2015). For example, the paper (Park et al., 2017) demonstrated that CNN with well pruned spatial convolution kernels make convolution layers in inference as much as threefold faster provided the sparse convolution operation is implemented competently. In principle, sparse methods and transform methods are orthogonal, and thus synergies between them can be leveraged to further accelerate convolutions. Particularly, applying sparse methods to Winograd convolution holds the most potential to achieve highest convolution speed for common convolution kernel sizes.

Nevertheless, pruning Winograd kernels poses challenges: Firstly, even a well pruned spatial convolution kernel will always lead to a Winograd kernel with much reduced sparsity because the Winograd transformation will introduce more non-zeros in the Winograd domain. More importantly, the linear transform that maps spatial to Winograd parameters is non-invertible because the inverse transformation is an overdetermined system. Thus any training and pruning method that needs to make use of both sets of parameters in conjunction will cause inconsistency, which in turn leads to major accuracy loss when achieving acceptable sparsity.

In this paper, we show that pruning Winograd parameters becomes successful when we replace the convolution layer in question by a Winograd layer, eliminating the need to use both the spatial and Winograd parameters in conjunction. Figure 1 illustrates the architecture of a Winograd

[*]Equal contribution, and authors are listed in alphabetical order. [1]Intel Labs, Intel Cooperation, USA. Email Addresses: Sheng Li <sheng.r.li@intel.com>, Jongsoo Park <jongsoo.park@intel.com>, Ping Tak Peter Tang <peter.tang@intel.com>.

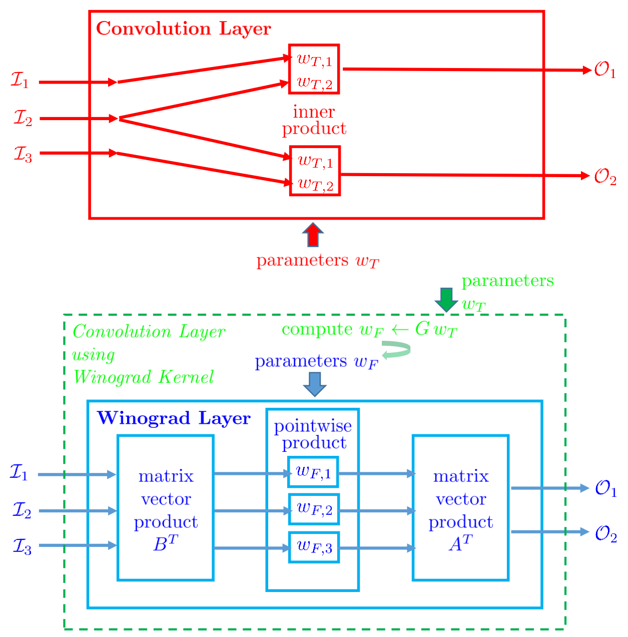


Figure 1. The red schematic on the top illustrates a traditional convolution layer. This example takes the inputs $\mathcal{I}_j$, $j = 1, 2, 3$, and kernel parameters $w_{T,j}$, $j = 1, 2$, and produces the output $\mathcal{O}_j$, $j = 1, 2$. The blue schematic on the bottom illustrates a corresponding Winograd layer. Note that it takes three independent kernel inputs $w_{F,j}$, $j = 1, 2, 3$. Given any inputs $\mathcal{I}_j$, the Winograd layer produces two outputs $\mathcal{O}_j$. The standard Winograd-as-a-kernel implements a convolution layer using the Winograd computation method, and is illustrated by the green dotted line schematic. It takes the spatial convolution parameters (two coffients) w_T and the input $\mathcal{I}$. Internally, it applies the (non-invertible) transform to w_T that yields $w_F = G w_T$, which is then used in the Winograd computation. Consequently, the set of w_F used here is at most two dimensional, instead of the Winograd layer that can exploit a three-dimensional input space for w_F.

layer using a simple 1D convolution of a 2-length spatial kernel with 4-length input. The corresponding Winograd kernel is 3-length.

Replacing a convolution layer by a Winograd layer has several advantages. First, from a conceptual point of view, this new layer architecture reflects directly the relationship between the key parameters of the computation (which are the Winograd parameters) and the overall neural network. Second, as alluded to earlier, we have more Winograd than spatial parameters. Thus a Winograd layer in fact has a larger capacity as we are no longer restricted to use only those Winograd parameters that actually correspond to a set of spatial convolution parameters. Last but most important, steering of the training process such as pruning becomes possible and straightforward. We emphasize that this is not the case if one employs the spatial convolution layer: the non-invertible mapping between convolution and Winograd parameters is a major obstacle. While this obstacle may be overcome via approximation for small networks (LeNet) and datasets (MNIST) as reported in (Liu & Turakhia, 2016), our experiments (Section 7.2) show that approximation does not work for larger networks (AlexNet) and datasets (ImageNet).

The main contributions of this paper are as follows.

1. We define and advocate the use of the Winograd layer as an architecture. Details of the forward and backward passes of this layer are derived and training methods aim to prune are devised (Sections 3–5).

2. We formulate a performance model of a Winograd layer whose parameters are possibly pruned (Section 6). The model corroborates the expectation that pruned Winograd layers can outperform state-of-the-art convolution algorithms including both dense winograd convolution and spatial-domain sparse convolution.

3. We demonstrate the effectiveness of training and pruning with the Winograd-layer architecture (Section 7). For instance, we prune AlexNet (Krizhevsky et al., 2012) with its Winograd parameters pruned to more than 90%, while maintaining its original accuracy. This leads to more than $2\times$ projected speedups over dense Winograd convolution, one of the fastest convolution algorithms to date (Lavin & Gray, 2016).

2. Related Work

That convolution can be computed with less arithmetic complexity through the use of transformation is well understood. The papers (Mathieu et al., 2013; Vasilache et al., 2015) appear to be the first ones that detailed the use of FFT as a compute kernel in Deep Learning. This kernel executes the "spatial-domain" convolution by element-wise multiplication in the "frequency-domain". More recently, the paper (Lavin & Gray, 2016) shows convincingly that Winograd transforms outperform FFTs in the common convolution use cases in Deep Learning. These works illustrate the use of transform methods (FFT or Winograd) as computation kernels to speed up a normal inference or training process.

To further improve computation and memory footprint of FFT and Winograd, the research report (Liu & Turakhia, 2016) attempts to prune parameters in the transformed domain. For Winograd parameters, the overall network's architecture is that of the original CNN, but the forward pass is performed using the Winograd kernel with some of the parameters set to zero. A backward pass is performed that updates the original "spatial-domain" parameters. We believe there is an inconsistency in this model caused by the fact that mapping between spatial-domain and Winograd convolution parameters is non-invertible. In general, there is no spatial-domain convolution parameters that correspond to the modified (masked off) Winograd parameters that are used to compute all the forward-pass intermediate quantities, while the backward pass computes the gradient using these intermediate quantities *in conjunction with* the spatial-convolution-kernel parameters. Only experiments with LeNet (LeCun et al., 1998a) on the MNIST dataset (LeCun et al., 1998b) are reported in (Liu & Turakhia, 2016). We also tried a number of similar inexact approaches to navigate the non-invertible situation such as using the pseudo inverse operator and subspace projection. All these methods work reasonably well with LeNet on MNIST. However, on larger networks and datasets such as AlexNet (Krizhevsky et al., 2012) on ImageNet (Deng et al., 2009), significant accuracy loss and/or low sparsiy are inevitable as shown in Section 7.2. Indeed, direct pruning of the parameters in our proposed Winograd layers is how we finally overcome this accuracy problem.

In a conceptual sense, the recent work (Rippel et al., 2015) relates closely to our current work. The authors advocate representing the convolution kernels in frequency domain and detailed the gradient calculation with respect to the frequency parameters. The network architecture, however, remains in the original form as outlined in Section 4 of the paper: When convolution is to be computed, the frequency parameters must first be transformed back to spatial domain in which a regular convolution is performed.

A hardware feature for zero-skipping has been proposed to take advantage of sparsity in Winograd parameters (Park et al., 2016) even when the sparsity is not high enough to benefit from general sparse representations such as compressed sparse row (CSR). This feature is motivated by that pruning in spatial domain does not provide a high enough sparsity in Winograd parameters. Our paper shows that, by directly pruning in Winograd domain, we can obtain high enough sparsity to speedup inference without resorting to such specialized hardware features.

3. Winograd Layers – Definition

Consider a typical convolution layer where the input tensors with C channels of features maps each of dimension $H_i \times W_i$ are transformed into K output channels via a simple unit-stride, unit-dilation linear convolution with kernels of size $r \times s$:

$$\mathcal{I} \in \mathbb{R}^{C \times H_i \times W_i} \to \mathcal{O} \in \mathbb{R}^{K \times H_o \times W_o},$$

$H_o = H_i - r + 1$ and $W_o = W_i - s + 1$, via

$$\mathcal{O}(k,:,:) = \sum_{c=1}^{C} \mathcal{W}(k,c,:,:) \star \mathcal{I}(c,:,:) \qquad (1)$$

where $\star$ stands for 2D linear convolution.

The computation of Equation 1 can be performed using the Winograd transform which has a lower arithmetic complexity than Equation 1 suggests. The details of this computation we present now are crucial to our definition of Winograd layers and their training. First, the convolution $\mathcal{W}(k,c,:,:)\star\mathcal{I}(c,:,:)$ can be broken down into smaller tiles of $\mathcal{I}$ via an overlap procedure that is often used in the context of digital signal processing. We illustrate overlapping by the following one-dimensional example, using Matlab-like array index notation that is self explanatory.

$$W(0:2)\star I(0:5) \to O(0:3)$$

can be computed as

$$W(0:2)\star \begin{bmatrix} I(0:3) \\ I(2:5) \end{bmatrix} \to \begin{bmatrix} O(0:1) \\ O(2:3) \end{bmatrix}.$$

Note that I is broken up into two tiles with some duplicating elements while O is partitioned (without duplication) into two tiles. More generally, given convolution kernels of size $r \times s$ and (small) sizes m, n that divide[1] H_o and W_o, respectively, we can reshape the input and output tensors $\mathcal{I}$, $\mathcal{O}$ into $\tilde{\mathcal{I}}$, $\tilde{\mathcal{O}}$

$$\mathcal{I} \in \mathbb{R}^{C \times H_i \times W_i} \to \tilde{\mathcal{I}} \in \mathbb{R}^{C \times T \times (m+r-1) \times (n+s-1)}$$

and

$$\mathcal{O} \in \mathbb{R}^{C \times H_o \times W_o} \to \tilde{\mathcal{O}} \in \mathbb{R}^{C \times T \times m \times n}.$$

The value T is the number of resulting tiles of the reshaping, $T = H_o W_o/(mn)$. The input tile size is $(m+r-1) \times (n+s-1)$, while the output tile size is $m \times n$. We express the reshaping by two index mapping functions ϕ and ψ

$$\tilde{\mathcal{I}}(c,t,i,j) = \mathcal{I}(c,\phi(t,i,j)), \quad \mathcal{O}(k,i,j) = \tilde{\mathcal{O}}(c,\psi(i,j))$$

where ϕ is many-to-one and maps a 3-tuple to a 2-tuple while ψ is invertible and maps a 2-tuple to a 3-tuple. Using the overlapped form, we have

$$\tilde{\mathcal{O}}(k,t,:,:) = \sum_{c=1}^{C} \mathcal{W}(k,c,:,:)\star\tilde{\mathcal{I}}(c,t,:,:). \qquad (2)$$

Straightforward convolution of a $r \times s$ array U with a $(m+r-1) \times (n+s-1)$ array V takes $mnrs$ multiplications. In contrast, Winograd's method can possibly need only as few as $(m+r-1)(n+s-1)$ multiplications via:

$$U\star V = A_1^T \left[(G_1 U G_2^T) \odot (B_1^T V B_2) \right] A_2.$$

The six matrices A_j, G_j, B_j, $j = 1, 2$, (of consistent dimensions) are independent of U and V, and $\odot$ is element-wise multiplication. In many instances the A_j, B_j, C_j matrices are so simple that applying them requires no multiplications. For example when $r = s = 3$ and $m = n = 2$,

[1] This assumption simplifies the presentation and can be easily eliminated by for example zero padding.

$A_1 = A_2$, $B_1 = B_2$ and $G_1 = G_2$ and are respectively

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 1 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}.$$

Thus, using tiling and Winograd transformation, Equation 1 is equivalent to

$$\tilde{\mathcal{O}}(k,t,:,:) = A_1^T \left[\sum_{c=1}^{C} \left(G_1 \mathcal{W}(k,c,:,:) G_2^T\right) \odot \left(B_1^T \tilde{\mathcal{I}}(c,t,:,:) B_2\right) \right] A_2. \tag{3}$$

Motivated by this, we define a Winograd layer to be a topology specified by a tensor $\mathcal{W}_F \in \mathbb{R}^{K \times C \times (m+r-1) \times (n+s-1)}$ that computes $\mathcal{O}$ from $\mathcal{I}$ via

$$\begin{aligned} &\tilde{\mathcal{I}}(c,t,i,j) = \mathcal{I}(c, \phi(t,i,j)), \\ &\tilde{\mathcal{O}}(k,t,:,:) = \\ &\quad A_1^T \left[\sum_{c=1}^{C} \left(\mathcal{W}_F(k,c,:,:)\right) \odot \left(B_1^T \tilde{\mathcal{I}}(c,t,:,:) B_2\right) \right] A_2, \\ &\mathcal{O}(k,i,j) = \tilde{\mathcal{O}}(k, \psi(i,j)) \end{aligned} \tag{4}$$

Since $m, n > 1$ in practice, $(m+r-1)(n+s-1) > rs$ and a Winograd layer (Equation 4) has a higher capacity than a corresponding convolution layer (Equation 1).

4. Winograd Layers – Backward Propagation

To incorporate a Winograd layer within a standard CNN framework (e.g. Caffe) so as to allow training and inference, it suffices to be able to compute the forward and backward passes. The forward pass is straightforward as it simply follows Equation 4, for which we note that (Lavin & Gray, 2016) details an optimized implementation. For the backward pass, we need to compute the partial derivatives of the scalar loss function L w.r.t. each of the variables $\mathcal{I}(c,i,j)$ and $\mathcal{W}_F(k,c,i,j)$ in terms of the known partial derivatives of L w.r.t. $\mathcal{O}(k,i,j)$, or $\partial L / \partial \mathcal{O}$ in short. We present the derivations here as they pertain to the Winograd layers and are thus unavailable elsewhere.[2]

First, we use this key form of chain rule: Suppose the partial derivatives of a scalar function L w.r.t. an array of variables y_{ij}, $Y \in \mathbb{R}^{\mu \times \nu}$, are known. Moreover, the variables Y are in fact dependent variables of an array of variables x_{ij}, $X \in \mathbb{R}^{\mu' \times \nu'}$ via $Y = U^T X V$ where U, V are constant matrices of commensurate dimensions. The partial derivatives of L w.r.t. X are then given by

$$\frac{\partial L}{\partial X} = U \frac{\partial L}{\partial Y} V^T. \tag{5}$$

Next, introduce intermediate tensors $\tilde{\mathcal{I}}_F$ and $\mathcal{Z}$

$$\begin{aligned} \tilde{\mathcal{I}}_F(c,t,:,:) &= B_1^T \, \tilde{\mathcal{I}}(c,t,:,:) \, B_2, \\ \mathcal{Z}(k,t,:,:) &= \sum_{c=1}^{C} \mathcal{W}_F(k,c,:,:) \odot \tilde{\mathcal{I}}_F(c,t,:,:), \\ \tilde{\mathcal{O}}(k,t,:,:) &= A_1^T \, \mathcal{Z}(k,t,:,:) \, A_2, \end{aligned} \tag{6}$$

for all applicable indices c, k, t.

To obtain $\partial L / \partial \mathcal{W}_F$, apply Equation 5 (the Chain Rule) while noting from Equation 6 that the first two dimensions of $\mathcal{Z}$ are in fact the product of two matrices:

$$\mathcal{Z}(:,:,i,j) = \mathcal{W}_F(:,:,i,j) \tilde{\mathcal{I}}_F(:,:,i,j), \tag{7}$$

for all fixed and applicable i and j. Hence,

$$\begin{aligned} \frac{\partial L}{\partial \tilde{\mathcal{O}}(k,t,i,j)} &= \frac{\partial L}{\partial \mathcal{O}(k, \psi^{-1}(t,i,j))}, \\ \frac{\partial L}{\partial \mathcal{Z}(k,t,:.:)} &= A_1 \, \frac{\partial L}{\partial \tilde{\mathcal{O}}(k,t,:,:)} \, A_2^T, \\ \frac{\partial L}{\partial \mathcal{W}_F(:,:,i,j)} &= \frac{\partial L}{\partial \mathcal{Z}(:,:,i,j)} \, \left(\tilde{\mathcal{I}}_F(:,:,i,j)\right)^T. \end{aligned} \tag{8}$$

Note that the last equality above depends on Equation 7, and the transposition applies to the first two dimensions of $\tilde{\mathcal{I}}_F$.

Similarly, we use the Chain Rule to obtain $\partial L / \partial \mathcal{I}$, noting that $\partial L / \partial \mathcal{Z}$ is already computed previously:

$$\begin{aligned} \frac{\partial L}{\partial \tilde{\mathcal{I}}_F(:,:,i,j)} &= \left(\mathcal{W}_F(:,:,i,j)\right)^T \, \frac{\partial L}{\partial \mathcal{Z}(:,:,i,j)}, \\ \frac{\partial L}{\partial \tilde{I}(c,t,:,:)} &= B_1 \, \frac{\partial L}{\partial \tilde{\mathcal{I}}_F(c,t,:,:)} \, B_2^T, \\ \frac{\partial L}{\partial \mathcal{I}(c,i,j)} &= \sum_{\substack{t,i',j' \text{ where} \\ (i,j)=\phi(t,i',j')}} \frac{\partial L}{\partial \tilde{\mathcal{I}}(c,t,i',j')}. \end{aligned} \tag{9}$$

In summary, the backward propagation of Winograd layers is implemented by Equations 8 and 9. Most of these operations can take advantage of highly optimized BLAS level-3 routines as will be shown in our public code repository (link omitted for now to conform to the double-blind review process).

[2]Note that the Winograd-kernel approach (Lavin & Gray, 2016) simply uses Winograd to accelerate the convolution operations in both forward and backward propagation (with proper adjustments when accelerating backward propagation) and does not consider Winograd as a layer architecture with full fledged backward propagation.

5. Winograd Layers – Training and Pruning

Consider now a L-layer network with a mixture of layers such as convolution, Winograd, fully connected, and pooling. Denote the parameters by the vector $\boldsymbol{\theta} = [\mathbf{w}_{(1)}, \mathbf{w}_{(2)}, \ldots, \mathbf{w}_{(L)}]$ where the subscripts correspond to the layer number. For a given loss function, a standard Stochastic Gradient Descent training updates the parameters via

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta_k \nabla_{\mathcal{B}} E(\boldsymbol{\theta}).$$

The overall goal here is to arrive at a $\boldsymbol{\theta}^*$ such that many of its elements, including those within the Winograd layer, are zero. Pre-training, pruning, and fine-tuning (re-training) are the three major steps that help us toward this goal. More specifically, each step described below consists of critical techniques that improve sparsity, accuracy, and efficiency.

Pre-training: Implications of Domain Selection

The main goal of the pre-training step is to train the neural network to achieve a high accuracy to be used as the starting point for the pruning step. Previously, this pre-training step is done in the spatial convolution domain. With our newly defined Winograd convolution layer, we have the additional option to perform this step directly in Winograd domain.

PRE-TRAIN IN WINOGRAD DOMAIN

With Winograd convolution layers, pre-training directly in Winograd domain is straightforward and has an advantage of potentially getting higher accuracy as shown in Section 7.2 because a Winograd convolution layer has capacity larger than the corresponding spatial convolution layer.

PRE-TRAIN IN SPATIAL DOMAIN

Pre-training in spatial domain also has its own advantage. Particularly, many networks have models pre-trained in spatial domain available, which can save time. However, because of the domain shift from spatial to Winograd, a *smoother* is needed to pre-process the spatial domain models before applying both regularization and thresholding in Winograd domain in the pruning step. Particularly, the smoother further trains the models with a regularization function R applied to the Winograd parameters, which are computed "on-the-fly" based on the G_j matrices in Equation 3:

$$E(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda R(G(\boldsymbol{\theta})),$$

where $G(\boldsymbol{\theta})$ transforms those spatial convolution parameters into the Winograd representation, while leaving others unchanged. The smoother produces $\boldsymbol{\theta}^*$, when finishing pre-processing. Then, we start the pruning step with Winograd layers as below using $\boldsymbol{\theta}^{(\text{start})} \leftarrow G(\boldsymbol{\theta}^*)$.

Pruning with Regularization and Gradient-based Thresholding

Pruning is always done directly in Winograd domain, which is enabled by the Winograd layer. Regularization and gradient-based thresholding are the two techniques used for pruning. Regularization is a common and useful technique to attenuate over-fitting and induce sparsity. In common frameworks such as Caffe, one can easily incorporate different regularization functions on different layers. In essence, one minimizes the energy function of the form

$$E(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + R(\boldsymbol{\theta}),\ R(\boldsymbol{\theta}) = \sum_{\ell=1}^{L} \lambda_\ell \|\mathbf{w}_{(\ell)}\|_{p_\ell}. \quad (10)$$

Common choices of norms are $p_\ell = 1$ or 2, and we use $L1$-norm for the layers to prune.

To achieve higher sparsity, thresholding is used in addition to regularization. The idea of thresholding is to set to zero particular entries within the parameter $\boldsymbol{\theta}^{(k)}$ that are deemed inconsequential (Han et al., 2015; Wen et al., 2016; Guo et al., 2016). Instead of using one uniform threshold to judge the significance of all parameters, the threshold on a particular parameter is dependent on how significantly this parameter affects the underlying loss function L. The thresholding function $T_{\epsilon,\beta}$ is of the form

$$T_{\epsilon,\beta}(w) = \begin{cases} 0 & \text{if } |w| \left(|\frac{\partial L}{\partial w}(\boldsymbol{\theta})| + \beta\right) < \epsilon \\ w & \text{otherwise} \end{cases} \quad (11)$$

We apply the thresholding function to a vector simply by applying it on each element, denoted with the slightly abused notation: $T_{\epsilon,\beta}(\mathbf{w})$. The numbers ϵ and β are part of the "hyper-parameters" of the training procedure. This scheme uses smaller thresholds as the magnitude of gradient increases (threshold is ϵ/β, ϵ, and 0, when the magnitude of gradient is 0, $1 - \beta$, and ∞, respectively). We find that the choice of ϵ=1e-4 and β=0.1 works well. We emphasize that thresholding the parameters including Winograd parameters is now straightforward because of the Winograd parameters are the direct independent parameters of the Winograd layers.

In the Winograd domain, where the L-layer network contains both Winograd layers and other layers (e.g., pooling) with parameters: $\boldsymbol{\theta} = [\mathbf{w}_{(1)}, \ldots, \mathbf{w}_{(L)}]$, the pruning step applies regularization and thresholding as the following:

$$\begin{aligned} E(\boldsymbol{\theta}) &= L(\boldsymbol{\theta}) + \sum_{\ell=1}^{L} \lambda_\ell \|\mathbf{w}_{(\ell)}\|_{p_\ell}, \\ \boldsymbol{\theta}^{(\text{tmp})} &\leftarrow \boldsymbol{\theta}^{(k)} - \eta_k \nabla_{\mathcal{B}} E(\boldsymbol{\theta}^{(k)}), \\ \boldsymbol{\theta}^{(k+1)} &\leftarrow T_{\epsilon,\beta}(\boldsymbol{\theta}^{(\text{tmp})}). \end{aligned} \quad (12)$$

Fine-Tuning: Recover Accuracy Loss

Similar to pruning in spatial domain, pruning Winograd parameters will cause accuracy loss, which necessitates a fine-tuning step to recover the loss. Same as pruning, fine-tuning is only done in the Winograd domain. During fine-tuning, the zero parameters obtained from the pruning step are fixed, while the network is trained to adjust the other non-zeros parameters to recover accuracy loss. We use L2 regularization during fine-tuning. The larger capacity of Winograd domain gives another benefit here. Even with a high sparsity, the Winograd domain still retains more Winograd parameters due to its larger capacity than its spatial counterpart. Therefore, fine-tuning in Winograd domain can be more resilient to high sparsity while maintaining accuracy, so we can use regularization and thresholding more aggressively during pruning.

6. Inference Speed Projection of Sparse Winograd Layer

This section develops a performance model of sparse Winograd layers. Our performance model demonstrates high speedup potential of combining Winograd convolution with sparse computation, motivating sparse Winograd layer. The performance model also pinpoints the range of sparsity where sparse Winograd layer is most beneficial, thereby guiding the pruning process.

Following the notations defined in Section 3, consider a convolution layer with C input channels each with a $H_i \times W_i$ image, K output channels each with a $H_o \times W_o$ image, and CK convolution kernels each of size $r \times s$. For simplicity, assume $H_i = W_i = H_o = W_o = H$ and $r = s$. The number of floating point operations (FLOPs hereafter) in the baseline spatial convolution is

$$F_{\text{baseline}} = 2CKH^2r^2.$$

Unless otherwise noted, we count both multiplication and addition in the unit operation of convolution when calculating FLOPs, hence the coefficient of 2 in the equations.

Next, we consider sparse convolution (still in spatial domain) with kernels whose density of non-zero parameters is x (i.e. $100(1-x)\%$ of parameters are zero).

$$F_{\text{sparse}} = 2\alpha x CKH^2r^2,$$

where α denotes the overhead of sparse computation. In the ideal case of $\alpha = 1$, we will get speedups inversely proportional to x. In practice, there is typically an overhead associated with processing using sparse representations (e.g., $\alpha \approx 3$ has been observed on CPUs (Park et al., 2017)).

In Winograd-based convolution with $m \times m$ tiling,

$$F_{\text{winograd}} = \mathbf{2}\left(Cp^2 + \mathbf{CK} + Km^2\right)\left(\frac{\mathbf{H}}{\mathbf{m}}\mathbf{p}\right)^{\mathbf{2}},$$

where $p = r + m - 1$. The boldfaced term corresponds to the element-wise products in Equation 7 and dominates the total computation when C and K are large enough. The other terms correspond to the transformation of input and output activations between Winograd and spatial domains (multiplications with matrices A_1, A_2, B_1, and B_2). For simplicity, we assume not to take advantage of many -1, 0, and 1 appearing in these matrices.

Finally, we consider sparse Winograd model with non-zero density x.

$$F_{\text{sparse\_winograd}} = \mathbf{2}\left(Cp^2 + \alpha\mathbf{x}\mathbf{CK} + Km^2\right)\left(\frac{\mathbf{H}}{\mathbf{m}}\mathbf{p}\right)^{\mathbf{2}},$$

where the boldfaced term for element-wise products with sparsified parameters is multiplied with αx times, while the other terms for transformation remain the same.[3]

Using the performance model, Figure 2 plots the projected inference speedups of the 3×3 and 5×5 convolution layers of AlexNet for the range of sparsity obtainable with our training method that will be presented in the following section[4]. For a very high sparsity like $x \leq 0.05$, sparse convolution in spatial domain is projected to provide higher speedups than sparse Winograd convolution. This is because the transformation part of sparse Winograd convolution does not benefit from sparsity. However, in reality the achievable sparsity without accuracy loss is higher in Winograd domain than in spatial domain. Therefore, for the range of sparsity we can actually obtain without accuracy loss, sparse Winograd convolution has higher projected speedups than sparse spatial convolution as to be shown in Section 7.

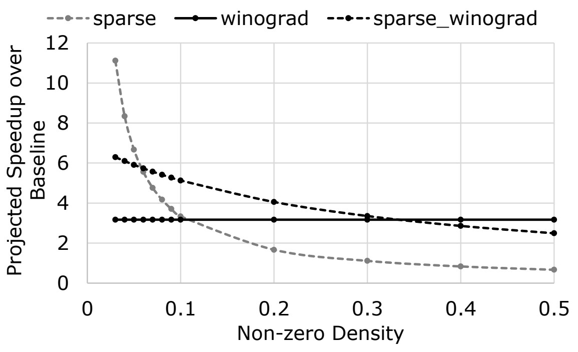
Our performance model can be used to determine whether sparse Winograd convolution will be beneficial for a given

[3] If the parameters associated with a given input or output channels are all zero, we can also reduce the computation associated with transformation, but we find such channels whose associated filters are all zeros are not frequent. There is a method to actively encourage group-wise sparsity with group Lasso regularization (Lebedev & Lempitsky, 2016; Wen et al., 2016), but such method results in lower sparsity (typically ∼50%) than methods for element-wise sparsity (typically ∼90%) to maintain the accuracy.

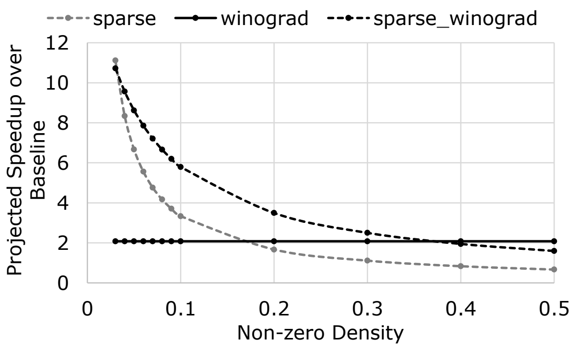
[4] When applying compute reduction techniques such as Winograd convolution and sparse convolution, it is important to check if performance is not memory bandwidth bound. We check the performance is indeed not memory bandwidth bound for the sparsity range plotted in Figure 2, assuming 10 FLOP/Byte ratio of compute throughput to memory bandwidth provided by the underlying hardware.

Table 1. The sparsity, projected inference speedups, and top-1 test accuracy of 3×3 and 5×5 convolution layers of AlexNet resulted from different pruning methods. The accuracy of the original AlexNet is 57.4%. `Training Domain`: the domain where we compute derivatives and optimize using SGD. `Pruned Domain`: the target domain where we apply regularization and thresholding to increase the sparsity, which is also the domain where we measure sparsity. `Method A`: the method of training and pruning directly in Winograd domain described in Sections 3–5. `B`: the method using spatial and Winograd parameters in conjunction similarly to (Liu & Turakhia, 2016). `B1` and `B2` are with different hyper-parameters, `B1` to match the sparsity with method A and `B2` to match the accuracy. `C`: spatial domain pruning results from (Park et al., 2017).

Method	Training Domain	Pruned Domain	Top-1 Accuracy	Sparsity of Convolution Layers				Projected Inference Speedups			
				`conv2`	3	4	5	`conv2`	3	4	5
A (Ours)	Winograd	Winograd	57.3%	90.6%	95.8%	94.3%	93.9%	5.2×	9.4×	6.0×	4.8×
B1	Spatial	Winograd	49.3%	92.8%	94.2%	93.2%	91.1%	5.5×	8.0×	5.6×	4.2×
B2	Spatial	Winograd	57.3%	54.1%	67.5%	62.4%	60.2%	2.6×	2.3×	1.9×	1.7×
C	Spatial	Spatial	57.4%	85.6%	93.1%	91.8%	88.5%	2.3×	4.8×	4.1×	2.9×



(a) conv2



(b) conv3

Figure 2. Projected inference speedups of `conv2` and `conv3` of AlexNet with varying sparsity.

CNN model and underlying hardware. In Figure 2, Sparse Winograd convolution is projected to outperform dense Winograd with sparsity higher than 67% (i.e. $x < 1/3$) because we assume the overhead of sparse computation, α, to be 3. If a convolution layer does not yield a higher sparsity, it is not useful to attempt to prune that layer. In fact, this is usually the case for the first convolution layer. If a processor architecture has smaller overhead, sparse Winograd convolution can benefit at a wider range of sparsity.

7. Results of Pruning Experiments

This section describes detailed training procedures we have experimented, their results on sparsity, accuracy, and inference speedup, and the typical sparsity patterns obtained.

7.1. Pruning Procedure

We implement the forward and backward propagation of Winograd layer described in Section 4 in a branch of Caffe (Jia et al., 2014).[5] We choose to use pre-trained spatial domain model to save overall training time. Particularly, we start with the Caffe reference model from the Caffe model zoo pre-trained in spatial domain using the ImageNet ILSVRC-2012 dataset (we call it AlexNet for simplicity even though it is a slight variation). Since the first convolution layer does not provide high enough sparsity to get speedups (Han et al., 2015; Wen et al., 2016), we do not attempt to prune that layer. We use the gradient-based thresholding described in Section 5 with ϵ=1e-4 and $\beta = 0.1$. We use learning rates of 5e-5 and 1e-4, and regularization factors of 5e-4 and 5e-5 in the pruning and fine-tuning steps, respectively. We use smaller learning rates for the Winograd layers (specifically, 200× smaller learning rate) to ensure convergence.

7.2. Sparsity, Accuracy, and Inference Speedup

Table 1 lists the sparsity, projected inference speedups, and accuracy of `conv2-5` layers in AlexNet. Our method of training and pruning directly in Winograd domain (method A) results in 90.6–95.8% sparsity with only 0.1% top-1 accuracy drop from the reference model, with projected inference speedups of the layers in the range of 4.8–9.4×.

[5] Note that the forward and backward propagation implementation is similar to that of the spatial convolution layers implemented with Winograd as a kernel (actually simpler because transformation of spatial parameters to Winograd layer is not needed). We could have leveraged libraries such as cuDNN and MKL-DNN if they provided a Winograd kernel interface accepting parameters in Winograd domain directly.

Note these projected speedups correspond to more than 2× over dense Winograd (Lavin & Gray, 2016). The actual speedups can be lower than the projected depending on the underlying hardware and the quality of implementation, and we leave the actual implementation for future work. Still, since dense Winograd has demonstrated to be one of the fastest convolution designs for common kernel sizes, the sparse Winograd convolution has the potential to hold a new record of inference speed for convolutions with common kernel sizes.

Method C in Table 1 is from recent spatial domain pruning results (Park et al., 2017). For a similar accuracy, we can obtain higher sparsity in Winograd than spatial domain. In addition, the sparsity we obtain in spatial domain (85.6–93.1%) does not belong to the very high sparsity range where models pruned in spatial domain are projected to outperform models pruned in Winograd domain (see Figure 2). Therefore, with practically obtainable sparsity, sparse Winograd has better speedup and outperforms sparse convolution in spatial domain.

The results shown in Table 1 use gradient-based thresholding described in Section 5, where atop regularization the gradient-based thresholding further reduces the non-zero density of each layer by up to 1.3× without affecting accuracy. The fine-tuning step improves the accuracy from 56.5% to 57.3% in method A. Although not a focus of this paper, we obtain an accuracy higher than the original AlexNet (58.0% vs. 57.4%) when we train in Winograd domain without pruning. This indicates a potential of achieving higher accuracies due to capacity larger than spatial convolution layers.

In addition to the method described in Sections 3 through 5 that trains and prunes convolution parameters directly in Winograd domain, we have tried methods that are similar to (Liu & Turakhia, 2016). Specifically, in one of the methods we have tried, we maintain both spatial convolution parameters and Winograd domain parameters and apply thresholding in Winograd domain in 3 steps:

(1) temporarily transform spatial parameters to Winograd domain,
(2) threshold the Winograd parameters,
(3) find the least-square projection that maps the parameters back to the spatial domain.

Note that, since we have more parameters in Winograd domain, the parameters in Winograd domain cannot be inversely transformed to the spatial domain exactly, hence the least-square projection. This non-invertibility poses various challenges. For example, after step (3), if we transform the convolution parameters again to Winograd domain, the sparsity will be much lower than what we obtain after step (2). Alternatively, step (3) can perform a constrained projection that maintains the sparsity obtained in step (2), but this incurs larger errors in the non-zero values. We have also tried a projection that is in between these two methods. However, due to the inevitable errors introduced from non-invertibility, the best result we were able to obtain is either with 8.1% top-1 accuracy drop (method B1 in Table 1) or with much lower sparsity (method B2). Nevertheless, these methods are all able to obtain more than 90% sparsity with around 98% accuracy for LeNet on MNIST dataset. This demonstrates that LeNet on MNIST dataset is inappropriate for evaluating pruning methods.

Table 2. The most frequenty sparsity patterns of filters in `conv3-5` layers of the pruned AlexNets.

(a) AlexNet pruned in spatial domain (method C in Table 1)

Frequency	66.4%	3.7%	1.7%	1.5%
Pattern	000	000	010	000
	000	010	000	000
	000	000	000	010

(b) AlexNet pruned in Winograd domain (method A in Table 1)

Frequency	72.1%	2.2%	1.2%	1.2%
Pattern	000000	000000	000000	110100
	000000	010000	010000	110100
	000000	000000	000000	000000
	000000	000000	010000	110100
	000000	000000	000000	000000
	000000	000000	000000	000000

7.3. Sparsity Patterns in Spatial and Winograd Domain

Table 2 lists the most frequent sparsity patterns appearing in 3×3 convolution layers of the pruned AlexNet. The most frequent sparsity pattern is all-zero, meaning no connection between the given input and output channel pair. The next most frequent sparsity patterns with one non-zero represent shifting and scaling. Essentially, sparse convolution in spatial domain provides flexibility for each input and output channel pair to choose kernel shapes and shifts. Note that, for the typical G_1 and G_2 matrices used for 3×3 Winograd convolution, even if the spatial-domain filter has only 1 non-zero out of 9 parameters, its Winograd domain counter part will have 11 to 20 non-zeros out of 36 parameters. This explains why pruning in spatial domain does not provide high sparsity in Winograd domain as observed in (Park et al., 2016).

With pruning directly in Winograd domain, we observe sparsity patterns that have no spatial domain counter parts. For example, other than the all-zero filters, the most frequent sparsity pattern shown in Table 2(b) has only one non-zero. This illustrates that CNN architectures with layers directly training parameters in Winograd domain is more expressive.

8. Conclusion

As CNN has become pervasive, the relentless pursuit of fast convolution has inspired new algorithms and techniques for accelerating convolution. Transformation methods, especially Winograd convolution, and sparse methods are among the most successful approaches. This paper is the first to successfully construct sparse Winograd convolution. Moreover, we have demonstrated that our sparse Winograd can achieve 90+% sparsity without accuracy loss, leading to more than 2× projected speedup over dense Winograd according to our performance model. Looking ahead, our next step includes sparse Winograd implementation that materializes the speedup (to be open-sourced together with our current sparse Winograd training code) and application to deeper networks like GoogLeNet (Szegedy et al., 2015) and ResNet (He et al., 2016).

ACKNOWLEDGEMENT

We would like to thank Nitish Shirish Keskar for recommendations on hyper-parameter settings and discussion on method B shown in Table 1. We would also like to thank Nadathur Satish, Chris J. Hughes, and Pradeep Dubey for discussion with them and their support.

References

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

Guo, Yiwen, Yao, Anbang, and Chen, Yurong. Dynamic Network Surgery for Efficient DNNs. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1379–1387, 2016.

Han, Song, Pool, Jeff, Tran, John, and Dally, William J. Learning both Weights and Connections for Efficient Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1135–1143, 2015.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the ACM International Conference on Multimedia*, pp. 675–678, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

Lavin, Andrew and Gray, Scott. Fast Algorithms for Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4013–4021, 2016.

Lebedev, Vadim and Lempitsky, Victor. Fast ConvNets Using Group-wise Brain Damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2554–2564, 2016.

LeCun, Yann, Denker, John S., and Solla, Sara A. Optimal brain damage. In *Proceedings of the 2Nd International Conference on Neural Information Processing Systems*, pp. 598–605, 1989.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998a.

LeCun, Yann, Cortes, Corinna, and Burges, Christopher JC. The MNIST database of handwritten digits. 1998b.

Liu, Baoyuan, Wang, Min, Foroosh, Hassan, Tappen, Marshall, and Penksy, Marianna. Sparse Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

Liu, Xingyu and Turakhia, Yatish. Pruning of Winograd and FFT Based Convolution Algorithm. 2016. `http://cs231n.stanford.edu/reports2016/117_Report.pdf`.

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

Park, Hyunsun, Kim, Dongyoung, Ahn, Junwhan, and Yoo, Sungjoo. Zero and data reuse-aware fast convolution for deep neural networks on GPU. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2016 International Conference on*, pp. 1–10. IEEE, 2016.

Park, Jongsoo, Li, Sheng, Wen, Wei, Tang, Ping Tak Peter, Li, Hai, Chen, Yiran, and Dubey, Pradeep. Faster CNNs with Direct Sparse Convolutions and Guided Pruning. In *International Conference on Learning Representations (ICLR)*, 2017.

Rippel, Oren, Snoek, Jasper, , and Adams, Ryan P. Spectral Representations for Convolutional Neural Networks.

In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 2449–2457, 2015.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going Deeper with Convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

Vasilache, Nicolas, Johnson, Jeff, Mathieu, Michael, Chintala, Soumith, Piantino, Serkan, and LeCun, Yann. Fast Convolutional Nets with fbfft: A GPU Performance Evaluation. In *International Conference on Learning Representations (ICLR)*, 2015.

Wen, Wei, Wu, Chunpeng, Wang, Yandan, Chen, Yiran, and Li, Hai. Learning Structured Sparsity in Deep Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 2074–2082, 2016.

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