EFFICIENT SPARSE-WINOGRAD CONVOLUTIONAL NEURAL NETWORKS

Xingyu Liu, Song Han, Huizi Mao & William J. Dally
Stanford University
{xyl, songhan, huizi, dally}@stanford.edu

ABSTRACT

Convolutional Neural Networks (CNNs) are compute intensive which limits their application on mobile devices. Their energy is dominated by the number of multiplies needed to perform the convolutions. Winograd's minimal filtering algorithm (Lavin (2015)) and network pruning (Han et al. (2015)) reduce the operation count. Unfortunately, these two methods cannot be combined — because applying the Winograd transform fills in the sparsity in both the weights and the activations. We propose two modifications to Winograd-based CNNs to enable these methods to exploit sparsity. First, we prune the weights in the "Winograd domain" (after the transform) to exploit static weight sparsity. Second, we move the ReLU operation into the "Winograd domain" to improve the sparsity of the transformed activations. On CIFAR-10, our method reduces the number of multiplications in the VGG-nagadomi model by $10.2\times$ with no loss of accuracy.

1 INTRODUCTION

Deep Convolutional Neural Networks (CNNs) are compute-limited. Their energy is dominated by the number of multiplies needed to perform the convolutions. Winograd's minimal filtering algorithm (Lavin (2015))(Winograd (1980)) reduces the number of multiplies required by $2.25\times$ to $4\times$, depending on the output patch size m. Pruning the model and exploiting the dynamic sparsity of activations due to ReLU non-linearity also reduces the required computation. (Han et al. (2015)) and (Han et al. (2016)) have shown that for typical CNNs, weights can be pruned to $30-50\%$ density and after ReLU non-linearity is applied, only $30-50\%$ of activations are non-zero. Thus exploiting sparsity of both weights and activations reduces the number of multiplies by $4-11\times$.

Unfortunately, the Winograd transformation fills in the zeros in both the weights and the activations (Figure 1a) — eliminating the gain from exploiting sparsity. Thus, on a pruned network, Winograd's algorithm actually increases the number of multiplies. The loss of sparsity more than offsets the reduced operation count from operating in the transform domain.

In this paper, we introduce two modifications to the original Winograd-based convolution algorithm to eliminate this problem. First, we prune the weights after (rather than before) they are transformed (Figure 1b). Thus the weights are sparse when the element-wise multiply is performed — reducing operation count. Second, we move the ReLU operation after the transform (Figure 1c) to also make the activations sparse at the point where the multiplies are performed. Together, these two transforms enable the gains of Winograd's algorithm and of exploiting sparsity to be combined. On the VGG-nagadomi network with $m = 2$, this combined approach gives a $10.2\times$ reduction in operation count.

2 PRUNING WINOGRAD-TRANSFORMED WEIGHTS

Conventional pruning (Han et al. (2015)) zeros weights in the "spatial domain", before the Winograd transform. Transforming the sparse kernel fills in the zeros resulting in a dense kernel in the "transform domain" (Figure 1a). We enable a Winograd transform to be used with sparse weights by pruning the weights in the transform domain (Figure 1b). We eliminate the spatial-domain kernel entirely. Our method operates in three phases: dense training, pruning, and retraining.

During the dense training phase, we train a dense 4×4 kernel (for $m = 2$) directly in the transform domain. The transformed kernel is initialized and trained directly by back-propagation through the

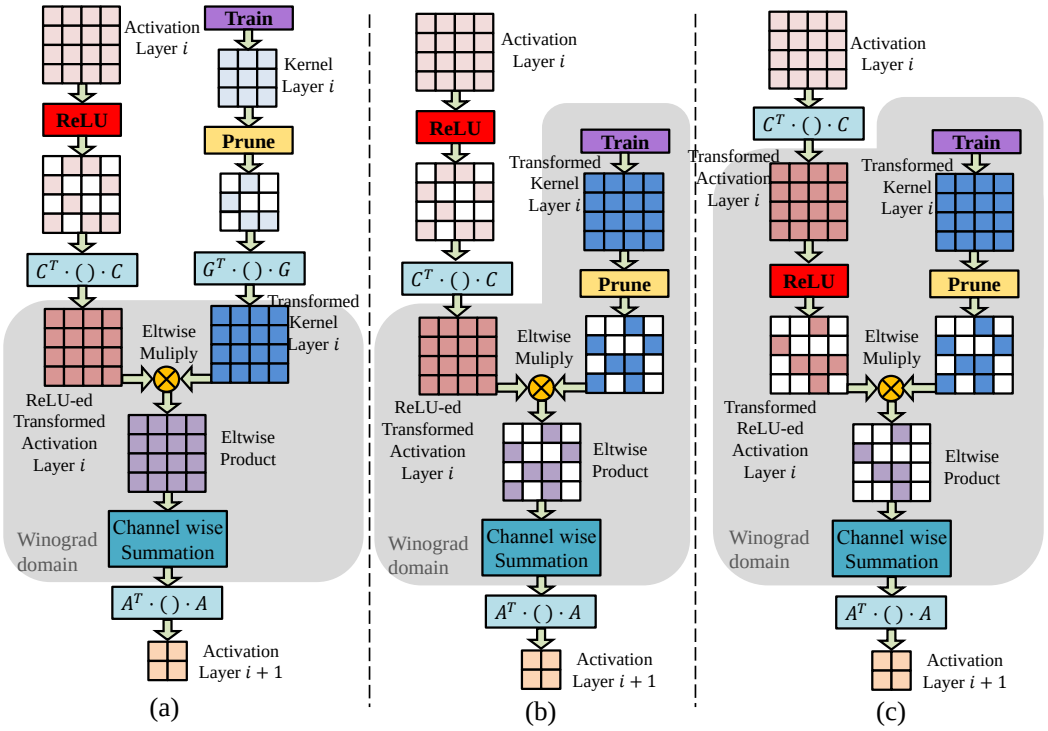

Figure 1: Combining Winograd convolution with sparse weights and activations. (a) Original Winograd-based convolution proposed by Lavin (2015) fills in the non-zeros in both the weights and activations. (b) Pruning the 4×4 transformed kernel restores sparsity to the weights. (c) Moving the ReLU layer after Winograd transformation also restores sparsity to the activations.

inverse transform — eliminating the need to maintain a kernel in the spatial domain or to transform a spatial kernel.

During the pruning phase, we prune the transformed kernel by computing the threshold T required to achieve a desired pruning rate R and setting all weights less than T to zero. In our experiments we used the same R for all network layers. Because sensitivity varies from layer to layer, we expect that better performance could be achieved by varying the pruning rate R_i for each layer i.

During retraining, we retrain the model using a "sparsity mask" to force the weights that were pruned to remain zero.

3 Moving ReLU to the Winograd Domain

In conventional CNNs, the ReLU non-linearity is applied to the output activations of the previous layer to produce the input activations of the current layer (Figure 1a & b). The ReLU operation zeros all negative activations resulting in significant sparsity in the spatial input activations. Unfortunately, the Winograd transform fills in this sparsity, resulting in dense transformed activations and no savings in the number of multiplies.

To give sparse activations in the transformed domain, where the multiplies are performed, we move the ReLU operation after the Winograd transform (Figure 1c). The ReLU zeros all negative transformed activations, reducing the number of multiplies. Because this ReLU is really associated with the previous layer, we perform this transformed ReLU starting with layer 2.

4 Results

We used Tensorflow (Abadi et al. (2016)) and Tensorpack (Wu (2016)) to train VGG-nagadomi (Nagadomi (2014))(Simonyan & Zisserman (2014)) on the CIFAR-10 dataset (Krizhevsky & Hinton (2009)). We tested pruning on three architectures (Figure 1 (a)(b)(c)) by pruning weights and

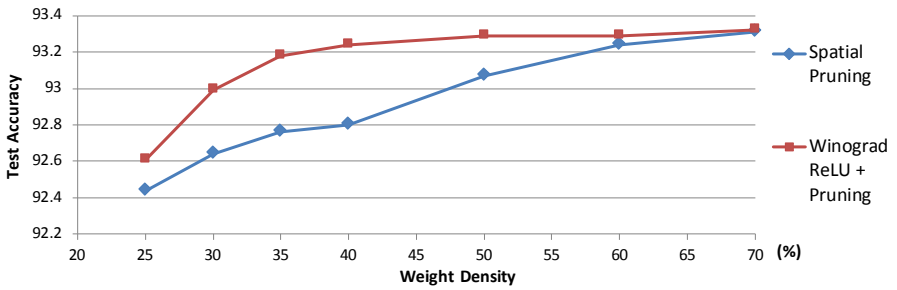

Figure 2: Test accuracy vs density for the three architectures of Figure 1 on VGG-nagadomi.

re-training until accuracy converges. We varied the pruning rate R from 20% to 70%. The first convolution layer is not included in pruning but is included in re-training.

Figure 2 shows accuracy as a function of density for the three architectures of Figure 1. The network of Figure 1c (which moves pruning and ReLU to the transform domain) can be pruned to 40% density without significant ($> 0.1\%$) loss of accuracy. The conventional network of Figure 1a can only be pruned to 60% density before accuracy falls.

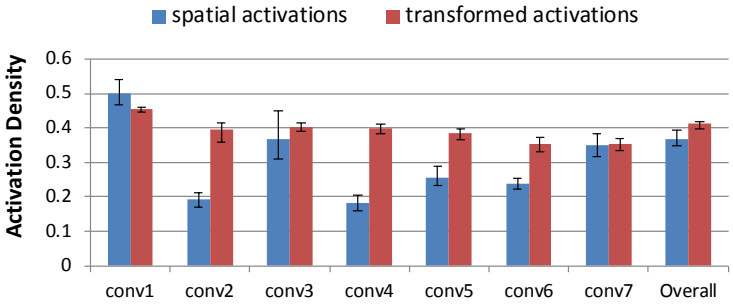

Figure 3: Activation density of convolution layers of VGG-nagadomi. Whiskers show one standard deviation above and below the mean.

Figure 3 shows the activation density for each network layer for the architectures of Figure 1a and 1c. Moving ReLU into the Winograd domain is effective in achieving activation sparsity with an overall activation density of 41.1% compared to 36.9% density for the spatial activations.

The original VGG-nagadomi network (no pruning, no Winograd) requires 2.3×10^8 multiplies per forward pass. Pruning this network and exploiting sparse activations reduces this by $4.6\times$ to 5.0×10^7. Using the Winograd transformation (Figure 1a) requires 1.1×10^8 multiplies, a reduction of $2.2\times$ compared to the original network, but an increase of $2.1\times$ compared to the pruned network. Moving pruning and ReLU into the Winograd domain requires 2.3×10^7 multiplies. It combines the $2.2\times$ savings from Winograd with the $4.6\times$ savings from sparsity to give a net reduction of $10.2\times$ compared to the original network.

5 Conclusion

We have shown that we can combine the $\approx 5\times$ computation savings of sparse weights and activations with the $2-4\times$ savings of the Winograd transform by making two modifcations to conventional CNNs. To make the weights sparse at the point of multiplication, we train and prune the weights in the transform domain. We move the ReLU non-linear operation after the Winograd transform to make the activations sparse at the point of multiplication. The net result is a $10.2\times$ reduction in computation for a 2×2 output patch ($m = 2$) with no loss of accuracy.

We expect that even greater savings on computation can be realized by using larger patch sizes (e.g., $m = 4$) and by using different pruning rates R_i for each network layer. To determine the scope of these techniques, they need to be evaluated on larger networks and data sets and on networks with residual bypassing layers (He et al. (2016)).

REFERENCES

Martín Abadi et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pp. 265–283, Berkeley, CA, USA, 2016. USENIX Association. ISBN 978-1-931971-33-1. URL http://dl.acm.org/citation.cfm?id=3026877.3026899.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *NIPS*, 2015.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Andrew Lavin. Fast algorithms for convolutional neural networks. *CoRR*, abs/1509.09308, 2015. URL http://arxiv.org/abs/1509.09308.

Nagadomi. Code for kaggle-cifar10 competition. 5th place. https://github.com/nagadomi/kaggle-cifar10-torch7, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

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

Yuxin Wu. Neural network toolbox on tensorflow. https://github.com/ppwwyyxx/tensorpack, 2016.