Pruning of Winograd and FFT Based Convolution Algorithm

Xingyu Liu
xyl@stanford.edu

Yatish Turakhia
yatisht@stanford.edu

Abstract

Winograd- and FFT-based convolution are two efficient convolution algorithms targeting high-performance inference. Their efficiency comes from the reduction of the number of multiplication operations due to linear and Fourier transforms. However, the two existing approaches cannot handle efficient compression of the neural network, which might contribute significant improvement in computation and memory footprint.

We propose to investigate the potential of pruning Winograd- and FFT-accelerated CONV layer computation. We used the heuristics of pruning the model in the based on the absolute value of the elements in the transformed domain. Our results shows that using Winograd-based convolution on LeNet-5, the number of parameters of the first and second layer can be both pruned to 10% of its original number, and using FFT-based convolution, can be both pruned to 25%, with only 0.18% loss of accuracy.

1. Introduction

Convolutional neural networks have become highly pervasive for various applications in visual recognition. In fact, the most recent and most successful neural networks, such as GoogleNet [7] and ResNet [3], only consist of convolutional layers. However, the testing time performance of several successful convolutional neural networks is still far from real-time. Compared to fully-connected layers, it is known that convolutional layer is far more intensive computationally. As a result, if we could accelerate CONV layer inference by reducing the computation workload, we could possibly achieve real-time neural network.

Recent work [5] has demonstrated a fast convolutional algorithm to improve the performance of convolutional layers by up to 4X in terms of reduction of number of multiplication operations. The key idea is to perform convolution in transformed domains using Winograd algorithm that reduces the number of operations required to perform the convolution. Similarly, prior work has investigated the speedup potential of using FFT for convolution, which converts the convolution to simple element-wise multiplication of matrices in the transformed domain.

To further pursue high efficiency in inference, a recent research work [2] described a method of reducing the storage and computation required by neural networks by pruning and compressing the network based on certain heuristics. Experiment results showed that the parameters of FC and CONV layers in several successful neural networks could be reduced by an order of magnitude and 2/3 without affecting their accuracy [2], which is a huge improvement in terms of memory footprint as well as computation workload. However, the above two algorithms , Winograd- and FFT-based convolution cannot exploit the potential of pruning and compressing the network.

In this work, we propose to investigate the potential of pruning Winograd- and FFT-based CONV layer computation. We used similar pruning heuristics as the previous work, but in the two transformed domain. To the best of the authors' knowledge, this is the first research work on the pruning of efficient convolution algorithm. The experiment results showed that the two convolution layers of LeNet-5 on MNIST dataset can be pruned to 10% of its original size, and using FFT-based convolution, can be both pruned to 25%, with only 0.18% loss of accuracy.

2. Related Work

Pruning has been shown to be very effective on convolutional neural network. This could have two advantages: (i) reducing the number of parameters required to be stored for CNN; and (ii) reducing the computation workload of the CNN. Both have huge impact on performance and energy. A recent research work [2, 1] has shown that for reducing its parameters by up to 13X. The pruning heuristics they used for FC layers are as follows: first, the network is trained give information on which connections are important based on the value of the weights in the connection. Next, after setting a threshold value, unimportant connections whose weight values are below the threshold is pruned. Finally, the network is re-train to fine-tune the weights of the remaining connections. Similar approaches can be applied to CONV layer, where the parameters can be reduced by up to 13X.

A more recent paper [1] further proposed a method to

address a problem in the aforementioned pruning approach that such pruning usually results in irregular network connections which cannot fit well on parallel computation. The proposed method uses structured sparsity at various scales including channel wise, kernel wise and intra kernel strided sparsity. The pruned network is then re-trained to prevent lose of accuracy.

3. Pruning Method of Winograd- and FFT-based 2D Convolution

In this section, we describe the theoretical fundamental of the two efficient convolution algorithms and the mathematical support for the implementation of pruning and retraining. Figure 1 shows the overview of this procedure.

3.1. Pruning

It's known that convolution can be implemented using Fourier Transform. Letting $\mathcal{F}$ denote the Fourier transform and $\mathcal{F}^{-1}$ denote its inverse transform, the convolutions between feature map f and kernel g can be computed as follows:

$$f * g = \mathcal{F}^{-1}(\mathcal{F}(f) \cdot \mathcal{F}(g))$$

The complexity of the FFT-based method requires $6Cn^2 \log n + 4n^2$ operations, with each FFT requires $O(n^2 \log n^2)$ multiplications and elementwise product requires $4n^2$ multiplications [6].

Pruning further reduces the number of multiplications. Mathematically, the convolution operation after pruning is only approximate and is given by equation (1):

$$f * g \approx \mathcal{F}^{-1}(\mathcal{F}(f) \cdot \mathcal{F}(g) \cdot \mathbf{M}) \tag{1}$$

where M is a mask (with 0 and 1 as its elements) applied to FFT of kernel g, i.e. $\mathcal{F}(g)$. Since M is known at test time, the multiplications which are masked can be completely avoided. The ratio of 0's in M to the 1's is M is known as *pruning rate*. One could also avoid computation of corresponding elements in the FFT of feature map i.e. $\mathcal{F}(g)$.

Although FFT shows great speed-up for large kernels, recent study has shown that for small kernels, Winograd-based convolution algorithm outperforms [5]. Since the most recent and most successful neural networks typically adopt deep layers with small kernels (3×3 or 1×3), it's necessary to look at both algorithms.

A recent paper [5] proposed a fast algorithm for 2D convolution using Winograd's minimal filtering algorithms[8]. To compute an $m \times n$ output feature map with $r \times s$ filters, which we denote as $F(m \times n, r \times s)$, naive convolution requires $m \times n \times r \times s$ multiplications, while the proposed algorithm requires $(m + r - 1)(n + s - 1)$ multiplications. Since floating-point multiplications are expensive on GPUs, saving on multiplication operations means gaining in performance.

For 1D convolution with data size of 4 and kernel size of 3, we denote the input data to be $d = (d_0, d_1, d_2, d_3)$ and kernel to be $g = (g_0, g_1, g_2)$. We first transform the input data linearly to be $(d_0 - d_2, d_1 + d_2, d_2 - d_1, d_1 - d_3)$ and kernel to be $(g_0, \frac{g_0 + g_1 + g_2}{2}, \frac{g_0 - g_1 + g_2}{2}, g_2)$. By doing elementwise multiplication, we have intermediate results of

$$m_1 = (d_0 - d_2)g_0$$

$$m_2 = (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2}$$

$$m_3 = (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2}$$

$$m_4 = (d_1 - d_3)g_2$$

Then the the first and second element of the output $d_0g_0 + d_1g_1 + d_2g_2$ and $d_1g_0 + d_2g_1 + d_3g_2$ can be written as $m_1 + m_2 + m_3$ and $m_2 - m_3 - m_4$ respectively. The whole algorithm only uses 4 multiplications while naive convolution requires $2 \times 3 = 6$ multiplications.

When d and g are written as column vectors, the above ideas could be generalized in matrix form:

$$S = A^T[(Gg) \odot (C^T d)]$$

For $F(2, 3)$, the matrices are:

$$C = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

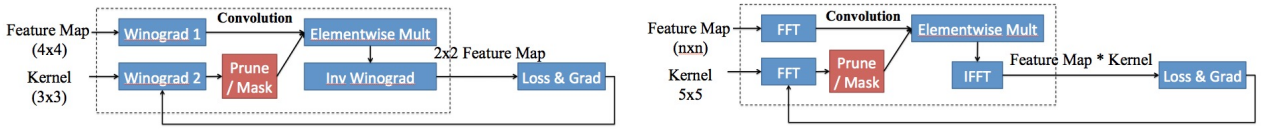
$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

For $F(2 \times 2, 3 \times 3)$, d and g are 4×4 and 3×3 matrices and the convolution becomes

$$S = A^T[(GgG^T) \odot (C^T dC)]A$$

$F(2 \times 2, 3 \times 3)$ uses 16 multiplications, however the naive algorithm uses $2 \times 2 \times 3 \times 3 = 36$. This is an arithmetic complexity reduction of $\frac{36}{16} = 2.25X$.

Similar to FFT-based convolution, pruning can also be applied to Winograd-based convolution. A mask could be



(a) Winograd convolution and pruning

(b) FFT convolution and pruning

Figure 1: Overview of Winograd and FFT based convolution and pruning.

applied to the transformed kernel before element-wise multiplication, as illustrated in equation (2) so that the number of multiplication could be further reduced.

$$S \approx A^T[((GgG^T) \odot M) \odot (C^T dC)]A \qquad (2)$$

Similar to the FFT case, one could also avoid computation of corresponding elements in the Winograd transformed domain of feature map i.e. $C^T dC$.

The heuristics of pruning we used in this work are based on the absolute value of transformed kernel in the Winograd domain and modulus of transformed kernel in the FFT domain. The heuristics were also used by [2] and is intuitive, since the absolute value described the importance of the each parameter element in the transformed domain.

3.2. Retraining

Aforementioned pruning improves computation efficiency at the sacrifice of validation and testing accuracy. We address this problem by doing re-training on the kernel in Winograd- and FFT-domain after pruning, i.e. train the network given the pruned element permanently set to zero.

Conventional training includes both forward pass of activation and backward pass of gradient. Compared to conventional training and conventional pruning-and-retraining proposed in [2], the retraining in the Winograd- and FFT-domain faces new challenges. Let's take Winograd-based convolution as an example. If only the parameters in the transformed domain are cached, the gradient in the transformed domain needs to be knew explicitly for the kernel update. However, when the 2×2 gradient of the output back-propagated from the next layer, it's impossible to reconstruct the gradient of the 4×4 element-wise product simply from the 2×2 gradient. Moreover, it's also impossible to re-construct the gradient of 4×4 data given the the gradient of the 4×4 element-wise product, since multiple gradient entries are forced to zero so the inverse linear transform will no longer stand.

Thus, we proposed an equivalent method of retraining to avoid the above mathematical dilemma. Equivalent of caching the transformed kernel, we cache the original kernel and mask instead. During forward pass, it will cost additional computation of the transformed kernel and applying mask. In the backward pass, however, the computation of gradient and update of original kernel are identical as the conventional training. In this way, we convert the retraining problem mathematically to its equivalent.

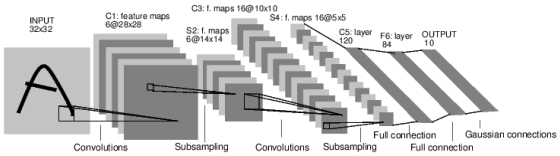


Figure 2: LeNet-5 architecture. Traditional Lenet-5 uses a kernel size of 5×5 and a padding of 2. For the research Winograd algorithm, however, we used 3×3 kernel and a padding of 1.

4. Experiment Methodology

In this section, we describe our experiment methodology of investigating the pruning and retraining of the two efficient convolution algorithms. We used LeNet-5 architecture for our implementation, as shown in Figure 2 with training on MNIST dataset. The network has two convolution layers where our techniques have been applied.

4.1. Winograd-based Convolution

We used Caffe as the framework for Winograd-based convolution pruning and retraining. The network we are investigating is LeNet-5. The dataset we are working on is MNIST. As describe in the previous section, we defined a new custom layer class of 'WinogradConv' where the forward pass and backward pass is performed. The backward pass is the same as normal convolution, so we reused the code of 'Convolution' layer class. However, we should define our own forward pass layer for the problem.

Due to complexity of the algorithm itself, only the CPU version of the forward pass is implemented. The forward pass consists of multiple $F(2 \times 2, 3 \times 3)$ operations which are finally combined to form the output feature map. Since the kernel size is 3×3, consecutive $F(2 \times 2, 3 \times 3)$ work on 4×4 patches of data with an overlap of two pixels vertically/horizontally to each other so that no overlap of original kernel exists. To generate the 4×4 patches correctly and efficiently, we used the *im2col* method provided by Caffe

with the parameters of dimension of 4 and stride of 2. The 4×4 patches and 3×3 kernels are transformed using gemm and static array storing the content of C and G in equation (2). The element-wise multiplications are then performed in a for loop before transformed back to the domain of output feature map using gemm and static array storing the content of A in equation (2). The output 2×2 output is then written to the corresponding position in the output feature map. Despite the use of gemm, the whole algorithm was implemented in a 6-layer for loop. Because of this, the forward pass is extremely slow and training small network like LeNet-5 for ten epochs on small dataset like MNIST takes more than an hour. That's why we chose small network of LeNet-5 and small dataset of MNIST.

The new 'WinogradConv' class has a member variable of 'mask', an dynamic array with the same size of transformed kernel. It keeps the mask used in one pruning-and-retraining iteration. The pruning is performed once by a member function 'winograd_prune()' at the beginning of retraining. The function receives the pruning rate as a parameter and sort the existing transformed kernel based on the absolute value of each element, according to aforementioned heuristics. The pruning threshold is set according to the pruning rate and the value of each element in the sorted array. For those elements whose absolute values are less than the threshold, the element in the corresponding position of the mask array is set to zero permanently.

During retraining, the mask is used in the for loop of the element-wise multiplication. This is equivalent to using pruned transformed kernel directly.

4.2. FFT-based Convolution

We used Theano as the framework for implementing FFT-based convolution pruning and retraining. First, we implemented LeNet-5 architecture using Theano and trained it on MNIST dataset. Then, similar to Winograd, we defined new operator for performing convolution in Theano, called 'FFTConvOp'. Most of the methods of this operator were kept same as the original convolution operator in Theano, called 'ConvOp'. However, we changed the 'perform' method, which is used during the forward pass to compute the function associated with the convolution operator. The input to this function is (i) a mini-batch of input images, (ii) kernel against which the input feature map is convolved and (iii)a mask which defines which elements of the kernel in the FFT-domain should be pruned (set to zero).

The dimensions of the input images is given by $B \times C \times H \times W$, where B is the mini-batch size, C is the number of input channels, H is image height and W is image weight. The dimension of the kernel parameter is given by $O \times C \times h \times w$, where O is the number of output channels, C is the number of input channels, h is filter height and w is filter width. The 'perform' method performs a $B \times O \times C$ 2-D convolutions between images in the mini-batch and the input kernels. We replaced this 2-D convolution using FFT and inverse FFT given by equation 1. This was implemented using *rfftn* and *irfftn* methods in numpy. For a $H \times W$ feature map convolved with hxw kernel, the FFT of the kernel has dimensions given by $(H+h-1) \times ((W+w)/2+1)$, and therefore, the mask has dimensions given by $O \times C \times (H+h-1) \times ((W+w)/2+1)$. This corresponds to $20 \times 1 \times 32 \times 17$ for convolution layer 1 and $50 \times 20 \times 16 \times 9$ for convolution layer 2 as 20 and 50 channels were used by the two layers, respectively.

Output Classes	10
Training samples	50,000
Test samples	10,000
Learning rate	0.1
Training epochs	25
Training error	1.12%
Test error	1.25%

Table 1: Parameters and error rates obtained for training LeNet-5.

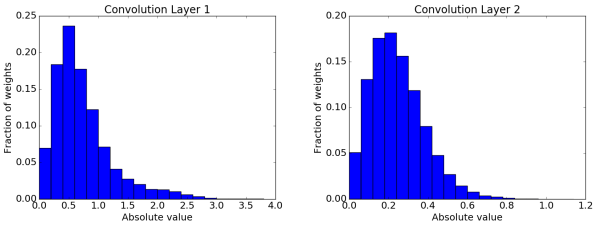
The 'grad' function remained unchanged and was used to update the weights in the time domain during back-propagation while retraining. The FFT corresponding to the kernel was recomputed every time in the forward pass. We confirmed that the output of the new convolution layer matched the output of the original layer when no pruning was performed. Since the weights of the kernel in the FFT domain are complex numbers, we used the absolute values of the weights for pruning, setting lowest $p\%$ of the weights to 0 for pruning rate p.

We note that our current implementation of FFT-based convolution with pruning on Theano framework is very slow. It takes about 12 hours for 1 epoch retraining on the entire MNIST dataset and an additional 30 minutes to perform testing. Therefore, for FFT, we show results for fewer pruning rates.

5. Experiment Results and Discussion

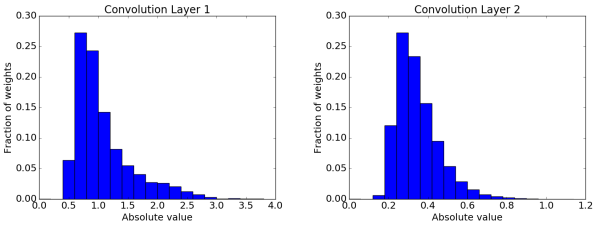
Table 1 shows the parameters used during training LeNet-5 architecture and the corresponding error rates. LeNet-5 has two convolution layers.

Figure 3 show the distribution of absolute value of weights of the FFT of kernel in fully trained LeNet-5. Figure 4 and 5 show the corresponding weight distribution at 50% and 75% pruning rate respectively. It is important to note that the pruning is applied per channel for each layer and therefore, the weight distribution after pruning in Figure 4 and 5 may not be equal to the distribution obtained after simply removing weights corresponding to the lowest values. However, it is clear that the pruning still retains a



(a) Convolution Layer 1 (b) Convolution Layer 2

Figure 3: Distribution of FFT weights of kernel for fully-trained LeNet-5.



(a) Convolution Layer 1 (b) Convolution Layer 2

Figure 4: Distribution of FFT weights of kernel after pruning 50% of the weights.

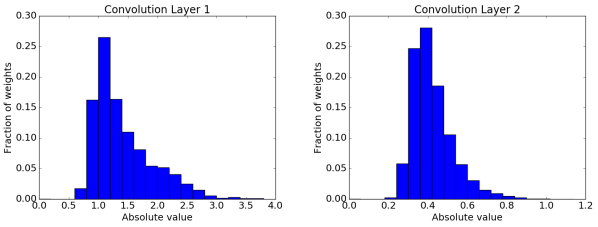
Pruning rate	Accuracy loss (w/o retraining)	Accuracy loss (1 epoch retraining)
25%	0.00%	0.00%
50%	0.21%	0.00%
75%	1.19%	0.18%
100%	89.93%	89.93%

Table 2: Loss of accuracy with pruning and retraining of FFT-based convolution.

large fraction of the weights which have high absolute values.

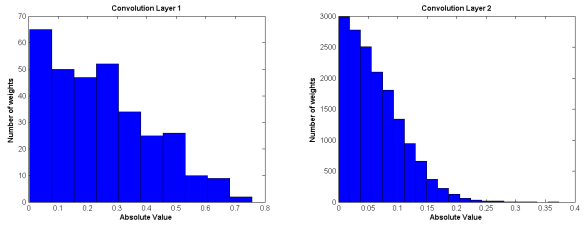
Table 2 shows the accuracy loss for different pruning rates for FFT-based convolution, with and without retraining. At 25% pruning, there is no loss of accuracy, even without retraining. This shows that in a fully-trained network, 25% of the weights are already 0 or have values so negligible that they never affect the classifier outcome. Even at 50% pruning, only 0.21% accuracy is lost but the accuracy is completely regained on retraining the network for only 1 epoch. At 75% pruning, 0.18% accuracy is lost after retraining for 1 epoch. It might be possible to regain that lost accuracy with more retraining. As a sanity check, we noted the loss of accuracy with 100% pruning. This should correspond to a random network that has no affect with retraining - both of which could be confirmed by the table (a random model would have about 90% error rate).

Figure 6 shows the distribution of absolute value of



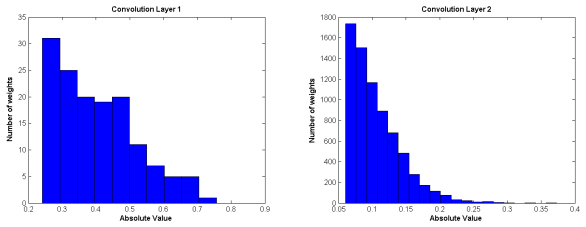
(a) Convolution Layer 1 (b) Convolution Layer 2

Figure 5: Distribution of FFT weights of kernel after pruning 75% of the weights.



(a) Convolution Layer 1 (b) Convolution Layer 2

Figure 6: Distribution of Winograd weights of kernel for fully-trained LeNet-5.



(a) Convolution Layer 1 (b) Convolution Layer 2

Figure 7: Distribution of Winograd weights of kernel fully-trained LeNet-5 after re-training with a pruning rate of 0.55

weights of the Winograd of kernel in fully trained LeNet-5. It is interesting to see that the pruning still retains a large fraction of the weights which have high absolute values. The result coincide with our heuristics that only the weights with large absolute values are kept.

Figure 7 shows the corresponding weight distribution after retraining with a pruning rate of .0.55. Although the total number of weights are inevitably reduced, it's interesting to see that after pruning the weights tends to have similar overall distribution as the weights without pruning. This may reveal some internal mechanism of how convolutional layer works. The weight distribution after pruning and retraining shed light on the possibility that we could do pruning and retraining for multiple rounds to further reduce the overall weight size.

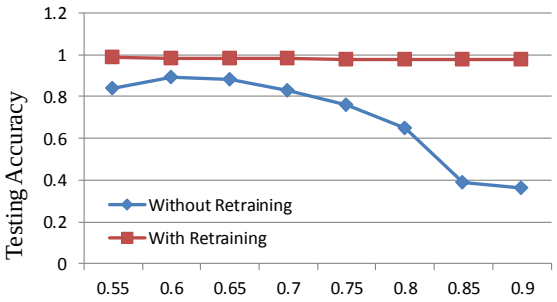


Figure 8: Test accuracy of pruned Winograd algorithm before and after retraining with different pruning rate. The test accuracy of original well-train LeNet-5 is 98.2%

Figure 8 shows the test accuracy of the LeNet-5 before pruning and after pruning, which is what we care about most. Regardless of inevitable noise, the overall trend of the test accuracy before pruning is that it drops with the increase of pruning rate. It is intuitive since the more we prune, the more information we lose from the original well-trained convolution layer. An interesting thing is that after retraining, the test accuracy can again be restored to the nearly or even higher than the test accuracy of the original network. The 'without loss' pruning rate can be extended to 90%, meaning we only need to keep 10% of the original weights. Since the pruned weights has similar distribution, we could possibly prune-and-retrain for several rounds to keep the accuracy from dropping.

As we can see, the pruning and retraining work well with both FFT- and Winograd-based convolution. With reasonable pruning rate, FFT-based convolution only have very small accuracy loss. With retraining, Winograd-based convolution can sustain its original test accuracy even with pruning rate of up to 90%. It's an exciting result that we should pay attention to, since it not only showed the potential of pruning of fast convolution algorithms, but also showed some internal structure of neural network.

6. Conclusion and Future Work

In this paper, we proposed a pruning and retraining mechanism for two efficient convolution algorithm. We derived the equivalent approach to handle the mathematical dilemma of back-propagation with the existance of pruning. The heuristics of pruning in terms of absolute value of weights are also proposed and tested. Our results showed that it is possible to prune a large number of parameters in the transformed domain of Winograd- and FFT-based convolution algorithms. In particular, using Winograd algorithm, it is possible to prune the number of parameters to 10% of the original without any loss of accuracy and with FFT algorithm, parameters could be pruned to 25% with only 0.18% loss of accuracy on LeNet-5 network. The loss of accuracy was measured with only 1 epoch of retraining since the retraining was slow on our current implementation.

We suggest some direction future work. First, it might be possible to prune even more with more rounds of retraining. Second, it would also be interesting to prune parameters of fast convolution algorithms on state-of-the-art networks much larger than LeNet-5, such as AlexNet [4], ResNet [3] and GoogleNet [7], all of which having large number of convolution layers. Third, besides compression, a major benefit of pruning is better performance. Benchmarking performance of fast convolution algorithms after pruning has been left for future work. Fourth, combining network pruning and quantization would yield further compression rate, which means even less memory footprint and potential higher throughput. Fifth, the current CPU implementation of pruning and re-training is way to slow for research on large network and large datasets. Our next step would be implementing the prune-and-training mechanism efficiently on GPU. Finally, we would like to explore the potential of customized accelerator (ASIC) designed for such fast and pruned convolution algorithm. Since the pruning method we introduced results in huge irregularity in terms of the pruned location, using customized accelerator would be able to exploit the potential that general purpose computing platform cannot handle.

References

[1] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *CoRR*, abs/1512.08571, 2015.

[2] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] Andrew Lavin. Fast algorithms for convolutional neural networks. *CoRR*, abs/1509.09308, 2015.

[6] Michaël Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *CoRR*, abs/1312.5851, 2013.

[7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[8] Shmuel Winograd. *Arithmetic complexity of computations*, volume 33. SIAM, 1980.