1D-FALCON: Accelerating Deep Convolutional Neural Network Inference by Co-optimization of Models and Underlying Arithmetic Implementation

Partha Maji and Robert Mullins

Faculty of Computer Science and Technology, University of Cambridge,
15 JJ Thomson Avenue, Cambridge, CB3 0FD, United Kingdom
{partha.maji,robert.mullins}@cl.cam.ac.uk

Abstract. Deep convolutional neural networks (CNNs), which are at the heart of many new emerging applications, achieve remarkable performance in audio and visual recognition tasks, at the expense of high computational complexity, limiting their deployability. In modern CNNs it is typical for the convolution layers to consume the vast majority of the compute resources during inference. This has made the acceleration of these layers an important research and industrial goal. In this paper, we examine the effects of co-optimizing the internal structures of the convolutional layers and underlying implementation of fundamental convolution operation. We demonstrate that a combination of these methods can have a big impact on the overall speed-up of a CNN, achieving a tenfold increase over baseline. We also introduce a new class of fast 1-D convolutions for CNNs using the Toom-Cook algorithm. We show that our proposed scheme is mathematically well grounded, robust, does not require any time-consuming retraining, and still achieves speed-ups solely from convolutional layers with no loss in baseline accuracy.

Keywords: Convolutional Neural Network, Deep Learning, Computational Optimization, Hardware Implementation

1 Introduction

Convolutional neural networks (CNNs) are becoming a mainstream technology for an array of new embedded applications including speech recognition, language translation, image classification and numerous other complex tasks. This breakthrough has been made possible by recent progress in deep learning. But, these deep models typically require millions of parameters and billions of operations to produce human level accuracy ([1],[8], [18]). The memory and compute requirements especially complicate the deployment of deep neural networks on low power embedded platforms as they have a very limited compute and power budget. To avoid running end-to-end inference on embedded systems, the current state-of-the-art solutions enable this type of application by off-loading the computation to cloud-based infrastructures where server-grade machines (GPUs and other manycore processors) perform the heavy number crunching. Unfortunately, the cloud assisted approach is limited due to the implications on the privacy, latency, and scalability of mobile applications [18].

In this paper, we propose a robust and easy-to-implement acceleration scheme, named 1-D FALCON (Fast Approximate Low-rank CONvolution), which can be applied on readily available state-of-the-art pre-trained models. Our proposed scheme exploits the inherent redundancy present in the convolution layers in order to reduce the compute complexity of deep networks. Additionally, we decompose each convolution layer into low-rank vectors to exploit row stationary computing [18]. We then apply a modified version of the Toom-Cook algorithm to compute the convolution on 1-D vectors to further reduce the number of multiplications in discrete convolution.

Although many earlier studies have focused on reducing overall memory footprint by compression, only a few have aimed at speeding up convolutional layers. Unlike many previously proposed pruning and regularization techniques, our scheme does not involve any time-consuming iterative retraining cycle. Furthermore, since rank selection and decomposition are only dependent on individual layer's inherent property, each convolution layer can be approximated in parallel. Our approximation scheme is mathematically well grounded, robust and thus easily tunable using numerical formulation and without sacrificing baseline accuracy. To the best of our knowledge, this paper is the first to study a co-optimization scheme that combines both the one-shot low-rank model approximation technique and a fast arithmetic scheme that exploits convolutions by separability.

2 Related Work

Model pruning has been used both to reduce over-fitting and memory footprint. Optimal brain damage [3] and optimal brain surgeon [9] are early examples of pruning which aimed at reducing the number of connections within a network. Recently, Han *et al.* proposed a pruning scheme for CNNs which aims at reducing the total number of parameters in the entire network ([8], [7]). However, the authors in this paper mentioned that it is challenging to achieve any significant runtime speed-up of convolutional network with conventional direct implementation. In addition, pruning involves a very long iterative prune and retraining cycle. For example, it took seven days to retrain the pruned five (convolution) layer AlexNet [8], which is not practical for fast time-to-market products.

Liu *et al.* [14] proposed a Sparse Convolutional Neural Networks (SCNN) model that exploits both inter-channel and intra-channel redundancy to maximize sparsity in a model. This method is very effective for number of parameter reduction in the fully-connected layers. The retraining stage with a modified cost function is very time-consuming.

Denton *et al.* showed in a recent research that the generalized eigendecomposition based truncation can help to reduce parameters from the fully-connected layers [4]. Although, the authors didn't consider the compute heavy convolutional layers. Jaderberg *et al.* proposed a singular value decomposition based technique for layer-by-layer approximation [10]. Their methodology uses iterative steps where a layer can only be approximated after the previous layer has been compressed. The author used an updated loss function to learn the low-rank filters which is again a time consuming process. The author also reported that simultaneous approximation of all the layers in parallel is not efficient. Mamalet *et al.* design the model to use low-rank filters from scratch and combine them with pooling layer [15]. However, their technique cannot be applied to general network design. Sironi *et al.* showed that learning-based strategies can be used to obtain seperable (rank-1) filters from multiple filters, allowing large speedups with minimal loss in accuracy [17]. We build our methodology on this fundamental idea. Instead of learning separable filters, we use a one-shot apporach which can be applied statically.

Gupta *et al.* [5] studied the effect of limited precision data representation in the context of training CNNs. They observed that CNNs can be trained using only 16-bit wide fixed-point number representation with little to no degradation in the classification accuracy. A number of optimization schemes have been proposed recently that recommend use of fewer bits to represent the parameters and datapaths ([2],[6],[7]). Our proposed scheme is orthogonal to these techniques and can be combined with quantization to further reduce the compute complexity and storage requirement.

Cong *et al.* showed that by using Strassen's algorithm computation complexity in convolutional layers can be reduced by up to 47% [1]. Vasilache *et al.* used a FFT based scheme to speed up convolutions, which are not very effective for small filters [19]. Recently, both nVidia's cuDNN and Intel's MKL library added support for Winograd's algorithm to speed up convolutions, which was originally proposed by Lavin *et al.* [12]. Although combining sparse methods and Winograd's convolution holds the potential to achieve significant speed up, pruning Winograd kernels to induce sparsity poses challenges.

3 Methodology

Sze *et al.* in their Eyeriss research project showed that a row stationary (RS) 1-D convolution is optimal for throughput and energy efficiency than traditional tiled 2-D implementation [18]. Our methodology follows the principles of 1-D row-stationary convolution. To achieve this we first approximate each layer to the necessary level to reduce compute complexity and then decompose each convolutional filter bank into two rank-1 filter banks by introducing an intermediate layer in between. If the classification accuracy drops at this stage we fine-tune the model using the training dataset. Then we apply a modified version of the Toom-Cook algorithm, which computes each 1-D convolution for a chosen set of distinct data points, to further reduce the number of strong operations (in this case multiplications). We will show that the combined application of these two schemes results into significant reduction in compute complexity.

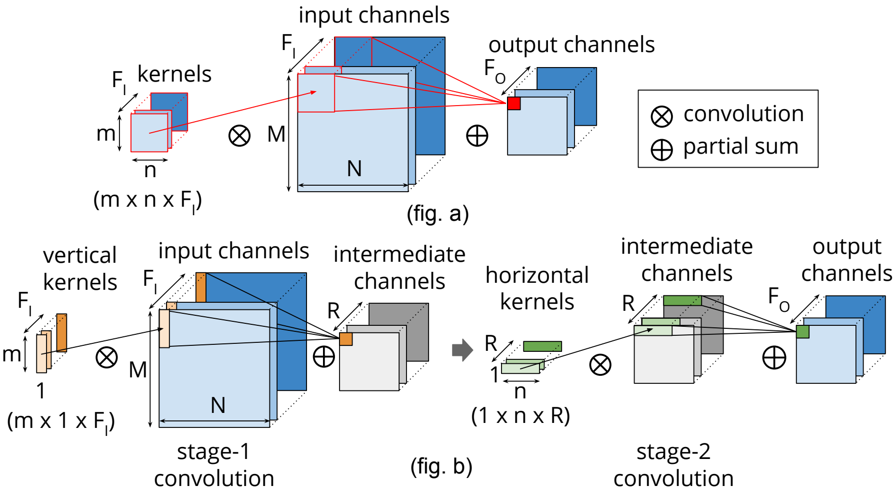

Fig. 1: (a) The original convolution with a $(m \times n)$ kernel. (b) The two-stage approximate convolution using a $(m \times 1)$ column kernel in stage-1 followed by a $(1 \times n)$ row kernel in stage-2. There are R channels in the intermediate virtual layer.

3.1 Layerwise Approximation and Convolution by Separability

In CNNs, multiple layers of convolutional filter (also known as kernel) banks are stacked on top of each other followed by a non-linear activation function. A significant redundancy exists between those spatial filter dimensions and also along cross-channel feature maps. Most of the previous research has focussed on either exploiting approximation along spatial filter dimensions or along one of the feature channel dimension. In our approach, we aim at approximating the redundancy across both the input and output feature maps.

Let us assume, in a convolutional neural network, a 4-dimensional kernel can be represented as $\mathcal{W} \in \mathbb{R}^{F_I\text{x}(m\text{x}n)\text{x}F_O}$, where spatial 2-dimensional kernels are of size $(m\text{x}n)$ and F_I, F_O are the input and output channels within a layer, respectively. We can also represent an input feature map as $\mathcal{X} \in \mathbb{R}^{M\text{x}N\text{x}F_I}$ and corresponding kernels as $\mathcal{W}_i \in \mathbb{R}^{m\text{x}n\text{x}F_I}$ for i-th set of weights, where each input feature map is of size $(M\text{x}N)$. The original convolution for the i-th set of weights in a given layer now becomes

$$\mathcal{W}_i * \mathcal{X} = \sum_{f=1}^{F_I} \mathcal{W}_i^f * x^f \tag{1}$$

Our goal is to find an approximation of kernel $\mathcal{W}_i$, such that $\mathcal{W}_i = \widetilde{\mathcal{W}_i} + \mathcal{E}$. Using the concept of separable filters [17], let us assume for a small error $\mathcal{E}$, the chosen rank is R. How the rank R is chosen will be explained in the next section. The modified kernel now can be represented by the Equation (2), where

$\mathcal{V} \in \mathbb{R}^{R\times(m\times1\times F_I)}$ is the approximate column kernel, and $\mathcal{H} \in \mathbb{R}^{F_O\times(1\times n\times R)}$ is the approximate row kernel.

$$\mathcal{W}_i*\mathcal{X} \simeq \sum_{r=1}^{R} \mathcal{H}_i^r(\mathcal{V}^r)^T = \sum_{r=1}^{R} h_i^r*\mathcal{V}^r = \sum_{r=1}^{R} h_i^r*(v_r*x) = \sum_{r=1}^{R} h_i^r*(\sum_{f=1}^{F_I} v_r^f*x^f) \tag{2}$$

Fig 1 depicts the idea of re-constructing the convolution layer using the newly constructed column and row low-rank kernels and compares them with the original 2-D direct convolution. We compute the column and row kernels $(\mathcal{V}, \mathcal{H})$ statically using generalized eigenvalue decomposition by minimizing the error $\mathcal{E}$. Since we decide the magnitude of the approximation statically, we avoid long running time from learning based techniques. Additionally, as the approximation is an inherent property of each layer, we can restructure all the convolutional layers in a CNN in parallel, which also saves time. If the accuracy of a model drops at this stage after approximating all the layers, we fine-tune the complete model for once using the training dataset.

3.2 Rank Search and Layer Restructuring Algorithm

The rank R is chosen by the one-shot minimization criterion described before. We apply singular value decomposition on the 2-D tensor $\mathbb{R}^{(F_I m)\times(nF_O)}$, which we obtain from the original 4-D tensor $\mathbb{R}^{F_I\times m\times n\times F_O}$. Unlike other minimization criterion such as Mahalanobis distance metric or data covariance distance metric [4], our simple criterion gives us an exact decomposition. Algorithm 1 describes the main steps of our low-rank approximation and CNN layer restructuring scheme.

Algorithm 1: Rank Approximation and Layer Restructuring Algorithm

```
function LayerwiseReduce $(M, C, W)$;
Input  : Target ConvNet model: M, Kernel Dimension: $p_i$,
         Compression factor of each layer: $[c_1, c_2, .., c_n]$,
         Pre-trained weights of individual layer:$[w_1, w_2, .., w_n]$
Output: Reduced ConvNet Model: $M^*$,
         Reduced weights of each layer: $[v_1, v_2, .., v_n]$, $[h_1, h_2, .., h_n]$
for $i \leftarrow 1$ to $Layers$ do
    if $layerType == Conv$ then
        $targetRank \leftarrow \frac{p_i F_I F_O}{c_i(F_I+F_O)}$;
        $U\Lambda V^T \leftarrow SVD(w_i)$;
        $disconnectLayers(w_i)$;
        $v_i \leftarrow U\sqrt{\Lambda}$;
        $h_i \leftarrow V\sqrt{\Lambda}$;
        $addNewLayer(targetRank)$;
        $M^* \leftarrow reconstructModel(M, v_i, h_i)$;
    end
end
```

3.3 The modified Toom-Cook's Fast 1-D Convolution

Once we have obtained newly constructed multi-stage 1-D convolution layers, we then apply a modified version of the Toom-Cook algorithm to reduce number of multiplication further. In the Toom-Cook method, a linear convolution can be written as product of two polynomials in the real field [20].

$$s(p) = w(p)x(p), \quad where \;\; deg[x(p)] = N-1 \;, \quad deg[w(p)] = L-1$$

The output polynomial $s(p)$ has degree $L+N-2$ and has $L+N-1$ different coefficients. Instead of explicitly multiplying the polynomials $w(p).x(p)$ using the discrete convolution, the Toom-Cook algorithm evaluates the polynomials $w(p)$ and $x(p)$ for a set of data points β_i and then multiplies their values $s(\beta_i) = w(\beta_i)x(\beta_i)$. Afterwards the product polynomials $s(p)$ is constructed using Lagrange interpolation. The algorithm consists of four steps:

1. Choose $L+N-1$ distinct data points β_0, β_1,...,β_{L+N-2}.
2. Evaluate $w(\beta_i)$ and $x(\beta_i)$ for all the data points.
3. Compute $s(\beta_i) = w(\beta_i)x(\beta_i)$.
4. Finally, compute $s(p)$ by Lagrange interpolation as follows

$$s(p) = \sum_{i=0}^{L+N-2} s(\beta_i) \frac{\prod_{j \neq i}(x-\beta_j)}{\prod_{j \neq i}(\beta_i - \beta_j)} \tag{3}$$

Since, $(L+N-1)$ distinct data points are chosen in step 1, total $(L+N-1)$ multiplications are required in step 3. The Toom-Cook algorithm can also be viewed as a method of factoring matrices and can be expressed as the following form ($\odot$ denotes element-wise multiplication)

$$s(p) = S[\{Ww(p)\} \odot \{Xx(p)\}] \tag{4}$$

where W, X and S are the transform matrix for kernels, input, and output respectively. The cost of computing $\{Ww(p)\}$ gets amortized over reuse of the result for many input slices. The matrices X and S consist of small integers $(0, \pm1, \pm2, ..)$, making it possible to realize them by a number of pre- and post-additions. The only dominant cost over here are $(L+N-1)$ elementwise multiplications.

4 Results and Analysis

In order to evaluate the effectiveness of our scheme we compared it against several popular networks targeting MNIST, CIFAR-10 and ImageNet dataset. In this paper, we demonstrate our result for VGG-16 model, which won the ImageNet challenge in 2014. VGG-16 is a deep architecture and consists of 13 convolutional layers out of a total 16 layers. To make a comparison with wide variety of speed-up techniques we chose a direct 2-D convolutional scheme [18], a low-rank scheme based on Tucker decomposition [11], two popular pruning techniques ([8],[16]), a

Table 1: A comparison of speed-up of VGG-16 using different schemes

Optimization Scheme	#MULs	Speed-up	Fine-Tuning Time
2-D convolution [18]	15.3G	1.0x	None
Groupwise Sparsification [13]	7.6G	2.0x	>10 epochs
Iterative Pruning [16]	4.5G	3.4x	60 epochs
Winograd [F(4x4,3x3)], [12]	3.8G	4.0x	None
Pruning+Retraining [8]	3.0G	5.0x	20-40 epochs
Tucker Decomposition [11]	3.0G	5.0x	5-10 epochs
1-D FALCON [Ours]	1.3G	11.4x	1-2 epochs

sparsification scheme [13], and Winograd's filtering scheme [12]. We used three main metrics for comparison: *(i)* **MULs:** Total number of strong operations in the convolutional layers, *(ii)* **Speed-up:** Total speed-up achieved compared to baseline 2-D convolution, and *(iii)* **Fine-Tuning Time:** Average fine-tuning time in number of epochs. As can be seen from the Table-1, our FALCON scheme achieves significant speed-up compared to any other scheme and does not require long fine-tuning time. The overall speed-up comes from combined application of both low-rank approximation scheme and fast 1-D convolution technique.

Speed-up from Low-rank Approximation: The computational cost of the baseline 2-D direct convolution is $\mathcal{O}(F_I MNmnF_O)$. But, using our 1-D FALCON approximation scheme, the computational cost for vertical-stage and horizontal-stage are $\mathcal{O}(F_I MNmR)$, $\mathcal{O}(RMNnF_O)$, respectively, resulting a total computational cost of $\mathcal{O}((mF_I + nF_O)MNR)$. If we choose R such that $R(mF_I + nF_O) \; << \; mn(F_I F_O)$, then computational cost can be reduced. Our evaluation on VGG-16 showed an average speed-up of 3-5x in all layers and a maximum 8-9x speed-up on many individual layers.

Speed-up from Toom-Cook Algorithm: The 1-D Toom-Cook algorithm requires a $(N + L - 1)$ number of multiplications compared to a direct implementation which will require $N{\times}L$ number of multiplications, where N, L are the dimensions of input feature slice and 1-D filter, respectively. In case of VGG-16 model, we chose $N = 4$ and $L = 3$, resulting a 2x savings in each 1-D stage. As our modified VGG-16 model has vertical and horizontal stages, it achieves a total 4x saving in multiplication.

Efficient Use of Memory Bandwidth and Improved Local Reuse: The 1-D convolution by separability in our FALCON scheme also aims to maximize the reuse and accumulation at the local storage level for all types of data including weights, activations and partial sums. In case of padded convolution unnecessary data loads are also avoided due to the fact that halo regions are now needed only either at the vertical or horizontal edges.

5 Conclusions

In this work we demonstrated that co-optimization of internal structure of models and underlying detailed implementation together can help to achieve significant speed-up in convolutional neural network based inference tasks. We have introduced an easy-to-implement and mathematically well grounded scheme to aim at row stationary 1-D convolution, which can be applied on any pre-trained model statically. Unlike many pruning and regularization techniques, our scheme does not require any time consuming fine-tuning. Our evaluation showed that using our 1-D FALCON scheme, a significant speed-up can be achieved in the convolutional layers without sacrificing baseline accuracy.

References

1. Cong, J., Xiao, B.: Minimizing Computation in Convolutional Neural Networks, pp. 281–290. Springer International Publishing, Cham (2014)
2. Courbariaux, M., Bengio, Y.: Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. CoRR abs/1602.02830 (2016)
3. Cun, Y.L., Denker, J.S., Solla, S.A.: Advances in neural information processing systems 2. chap. Optimal Brain Damage, pp. 598–605 (1990)
4. Denton, E., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. NIPS (2014)
5. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. CoRR abs/1502.02551 (2015)
6. Gysel, P., Motamedi, M., Ghiasi, S.: Hardware-oriented approximation of convolutional neural networks. CoRR abs/1604.03168 (2016)
7. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. ICLR (2016)
8. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. NIPS (2015)
9. Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: Optimal brain surgeon. NIPS (1993)
10. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. CoRR abs/1405.3866 (2014)
11. Kim, Y., Park, E., Yoo, S., Choi, T., Yang, L., Shin, D.: Compression of deep conv. neural networks for fast and low power mobile applications. EMDNN (2016)
12. Lavin, A.: Fast algorithms for convolutional neural networks. CVPR (2016)
13. Lebedev, V., Lempitsky, V.: Fast convnets using group-wise brain damage. CVPR (2016)
14. Liu, B., Wang, M., Foroosh, H., Tappen, M., Pensky, M.: Sparse convolutional neural networks. CVPR (June 2015)
15. Mamalet, F., Garcia, C.: Simplifying convnets for fast learning. ICANN (2012)
16. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient transfer learning. EMDNN (2016)
17. Rigamonti, R., Sironi, A., Lepetit, V., Fua, P.: Learning separable filters (2013)
18. Sze, V., Chen, Y., Emer, J.S., Suleiman, A., Zhang, Z.: Hardware for machine learning: Challenges and opportunities. CoRR abs/1612.07625 (2016)
19. Vasilache, N., Johnson, J., Mathieu, M., Chintala, S., Piantino, S., LeCun, Y.: Fast convolutional nets with fbfft: A GPU performance evaluation. ICLR (2015)
20. Wang, Y., Parhi, K.: Explicit cook-toom algo. for lin. convolution. ICASSP (2000)