Very Efficient Training of Convolutional Neural Networks using Fast Fourier Transform and Overlap-and-Add

Tyler Highlander*
highlander.2@wright.edu

Andres Rodriguez*,⋆
andres.rodriguez.8@us.af.mil

*College of Engineering and Computer Science
Wright State University
Dayton, Ohio, USA

⋆Air Force Research Laboratory
Dayton, Ohio, USA

Convolutional neural networks (CNNs) are currently state-of-the-art for various classification tasks, but are computationally expensive. Propagating through the convolutional layers is very slow, as each kernel in each layer must sequentially calculate many dot products for a single forward and backward propagation which equates to $\mathcal{O}(N^2n^2)$ per kernel per layer where the inputs are $N \times N$ arrays and the kernels are $n \times n$ arrays. Convolution can be efficiently performed as a Hadamard product in the frequency domain. The bottleneck is the transformation which has a cost of $\mathcal{O}(N^2 \log_2 N)$ using the fast Fourier transform (FFT). However, the increase in efficiency is less significant when $N \gg n$ as is the case in CNNs. We mitigate this by using the "overlap-and-add" technique reducing the computational complexity to $\mathcal{O}(N^2 \log_2 n)$ per kernel. This method increases the algorithm's efficiency in both the forward and backward propagation, reducing the training and testing time for CNNs. Our empirical results show our method reduces computational time by a factor of 16.3 times the traditional convolution implementation for a 8×8 kernel and a 224×224 input array.

Mathieu *et al.* [2] demonstrated that doing full convolutions as Hadamard products in the frequency domain significantly reduces the training and testing time of CNNs. In their work they efficiently calculated FFTs on a GPU and used these transforms to perform convolutions via a Hadamard product in the frequency domain.

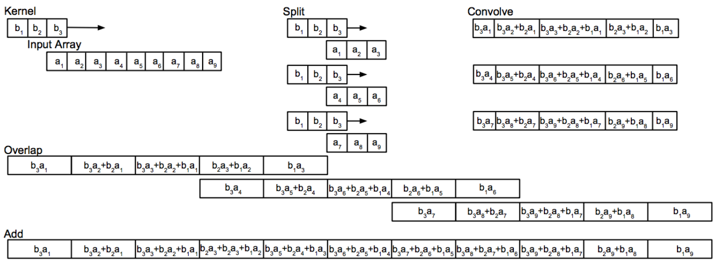
In this paper, we propose to use the overlap-and-add (OaA) technique [4] to further reduce the training and testing complexity to $\mathcal{O}(N^2 \log_2 n)$ per kernel. Note that the overlap-and-save [4] is a similar technique that may be marginally faster but has the same complexity. Table 1 compares the complexity of each method, where spaceConv refers to the traditional convolution in the space domain, FFTconv refers to convolution via a Hadamard product in the frequency domain without overlap-and-add, and OaAconv refers to convolution using overlap-and-save where each smaller convolution is efficiently computed in the frequency domain.

Method	Computational Complexity
spaceConv	$\mathcal{O}(N^2n^2)$
FFTconv	$\mathcal{O}(N^2 log_2 N)$
OaAconv	$\mathcal{O}(N^2 log_2 n)$

Table 1: Computational complexity comparison

In OaAconv, the input is broken into N^2/n^2 blocks equal to the kernel size $n \times n$. A convolution between each block and the kernel is computed and the results are overlapped and added. Figure 1 illustrates a simple 1-D overlap-and-add method for spacial convolution (that can easily generalize to 2-D). The input is first split into smaller blocks that are the size of the kernel. Smaller convolutions are computed between the kernel and the block inputs. The resulting convolutions are then added together to create the same results as a traditional spacial convolution. In practice each convolution is computed as a Hadamard product in the frequency domain. Note that even though the OaAconv method calculates more transforms than the FFTconv method, the fact that the transforms are smaller greatly outweighs the cost of having to calculate more.

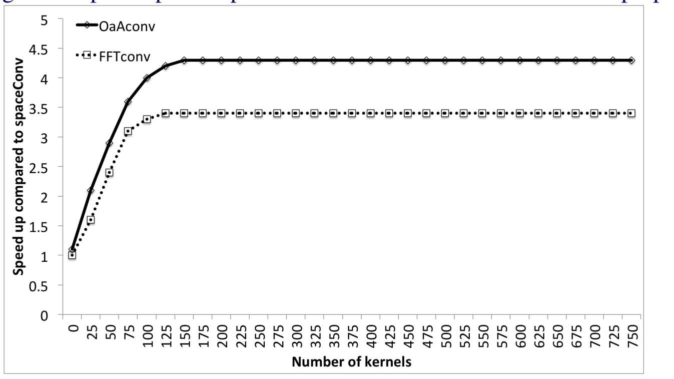
Figure 1: 1-D Overlap-and-Add convolutions



The overall time complexity of OaAconv can be further reduced by noting that all the block convolutions can be computed in parallel. If N^2 threads are available (a fair assumption for modern GPUs), the complexity on each thread is $n^2 \log_2 n$, and the overall time complexity is $\mathcal{O}(\max(N^2, n^2 \log_2 n))$ which is usually $\mathcal{O}(N^2)$. We can get additional speed up by taking advantage of the NVIDIA CUDA Fast Fourier Transform library (cuFFT) that computes each individual FFTs up to 10 times faster [3] (see also [5]). However, in order to have a fair comparison, our experiments in this paper run on single threads. As part of this work, we created a Caffe [1] fork[1] that uses a multi-thread GPU implementation of OaA for efficient convolutions.

We computed the increased in performance of FFTconv and OaAconv as we varied the number of kernels, the size of kernels, and the size of the input array for both forward and backward propagations. Figure 2 shows the speed-up factor of FFTconv and OaAconv compared to spaceConv in the forward propagation as the number of kernels is varied from 25 to 750 with a discrete step of 25. Each "number of kernels" experiment is repeated 10 times and the results are averaged. The input array is of size 32×32 and each kernel is of size 5×5. In this and most other experiments described in the full-paper, FFTconv and OaAconv outperform spaceConv, and OaAconv outperforms FFTconv at every step.

Figure 2: Speed-up over spaceConv vs. num of kernels in forward prop.



In future work we plan to optimize the FFT implementations for small size transforms, and design a CNN entirely in the frequency domain eliminating the bottleneck of the transforms. By creating a CNN that is in the frequency domain, the convolutional layers would only be the Hadamard products. The current challenge to this is to efficiently map an approximation of the non-linear transforms of CNNs (e.g., rectified linear units, sigmoid function, and/or hyperbolic tangent) to the frequency domain.

[1] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678, 2014.

[2] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through FFTs. *International Conference on Learning Representations*, 2014.

[3] CUDA Nvidia. Cufft library, 2010.

[4] Alan V Oppenheim, Ronald W Schafer, John R Buck, et al. *Discrete-time signal processing*, volume 2. Prentice-hall Englewood Cliffs, 1989.

[5] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *International Conference on Learning Representations*, 2015.

[1] github.com/THighlander