International Journal of Engineering Research And Management (IJERM)
ISSN: 2349- 2058, Volume-09, Issue-11, November 2022

Extremely Efficient Convolutional Neural Network for Mobile Devices using FFT

Weizhi Cui

Abstract—**Building deeper and larger convolutional neural network (CNN) is a primary trend for a wide range of applications such as image recognition, nature language processing. However, the most accurate CNNs usually have hundreds of layers and thousands of channels, thus requiring large computation and power consumption. The deployment of deep CNN in power-constrained and performance-limited scenarios remains challenging due to substantial requirements for computing resources and energy needed. In this paper, we propose a novel approach designed efficient CNN using FFT. The implementationand optimization on low-power zynq platform has been presented. Our empirical results show our method reduces computational time by a factor 2 times.**

Index Terms—**Overlap-and-Add, Convolutional Neural Network; Mobile Devices; FFT.**

I. Introduction

In recent years, Convolutional Neural Network (CNN) has been demonstrated as an effective method for various applications including image[9], video classification[10], object tracking[18]. CNN as a variant of the standard Deep Neural Network (DNN), is enable learning data-driven, highly representative, hierarchical image features from sufficient training data. It offers significant improvements in performance over deep neural network[13]. Deep CNN usually includes convolutional layer, ReLU layer, pooling layer and fully connected (FC) layer. The convolutional layers which are high computational complexity serve as a feature extractor to detect the specific features or patterns from the two-dimensional input data.

The most accurate CNNs usually have hundreds of layers and thousands of channels, thus requiring millions of parameters and billions of operations to process real applications. For example, AlexNet[9] which is a well-known CNN architecture has millions of parameters and billions of operations through the 5-layer convolution of the backward and forward propagation. It is worth noting that mobile platforms often have limited performance and power restrictions. It is a challenging to apply accurate CNN on mobile platforms which based on modern embedded SoC such as FPGAs, DSPs, GPUs([2], [3], [7]). Due to power and performance constraints, these computationally intensive networks are difficult to implement in mobile platforms such as smart cameras, drone-based image processing, medical patient monitoring automotive navigational intelligence, among many others.

Fast Fourier Transform (FFT) is a well-known approach, that reduces the computational complexity of 2D convolution from $O(N^2M^2)$ to $O(N^2log_2M)$, where the size of input image is $N\times N$ and the size of filter is $M\times M$. This method takes the FFT of the filter and input feature map, and then performs the multiplication in the frequency domain. The inverse FFT is applied to the resulting product to recover the output feature map in the spatial domain.

In this paper, we propose to use the Fast Fourier Transform (FFT) and Overlap-and-Add (OaA) to reduce the computational requirements of the convolutional layer. We use The Fastest Fourier Transform in the West (FFTW) to compute the discrete Fourier transform. The method implements on the Zynq platform which is an ARM-based and FPGA-based platform. The rest of the paper is organized as follows: Section 2 describes the OaA technique and our convolution implementation. Section 3 shows the related work. The section 4 is main method we proposed in this paper. Section 5 illustrates the experimental results. The paper ends with a conclusion in Section 6.

II. Background

Throughout this paper, we use the following notations:

1) Input feature maps of size $N_1\times N_2$.
2) The kernels size is $M_1\times M_2$.
3) The FFT block size is $L_1\times L_2$.
4) Output feature maps of size $N_1\times N_2$.

Convolutions are used widely in computer vision as a method of feature extraction, and when used in CNN, it generates feature maps from input images. For a 2D input data x, the output feature map y is generated by convolving x with filters f such that $y=CONV2(x,f)$. Using a direct implementation of convolution, the complexity of convolving a input image with size $N_1\times N_2$ and filter size $L_1\times L_2$ is $O(N_1N_2L_1L_2)$.

The 2D convolution can be computed in the frequency domain as a Hadamard product:

$$y=CONV2(x,f)=IFFT2(FFT2(x).*FFT2(f)) \quad (1)$$

The computational bottleneck is the Fourier transform between the space and the frequency domain.The complexity of 2D convolution by Hadamard product in the frequency domain is $O(N_1N_2\log_2 N_1N_2)$.

Overlap and Add method (OaA) is an efficient way to

Manuscript received November 27, 2022.

Weizhi Cui, School of Computer Science and Technology, Tiangong University, Tianjin, China

further reduce the complexity of 2D convolution. In OaA, the input is broken into $(N_1N_2)/(L_1L_2)$ (rounded up) blocks. A convolution between each block(the size is L_1L_2) and the kernel is computed. The results are overlapped and added to obtain the complete output feature map. Each convolution in OaA can be efficiently computed in the frequency domain, where the bottleneck is the complexity of each 2-D fast Fourier transfor $O((L_1L_2)log_2(L_1L_2))$. The total complexity for the entire input and kernel is the number of blocks times the complexity of each block convolution, i.e.,

$$\begin{aligned}&(N_1N_2)/(L_1L_2)*O((L_1L_2)log_2(L_1L_2))\\&\quad=O(N_1N_2log_2(L_1L_2)).\end{aligned} \tag{2}$$

since $N_1 \geq L_1 \geq M_1$ and $N_2 \geq L_2 \geq M_2$, the mini complexity of OaA method is $O(N_1N_2\log_2 M_1M_2)$, when $L_1 = M_1$ and $L_2 = M_2$.

In this paper, 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.

III. Related Work

Approaches to apply accurate CNNs on mobile platforms have been studied intensively at many different applications, and devices. Currently, there are two main approaches to accelerate CNNs including algorithm and hardware. From algorithm perspective, the target is to reduce thecomplexity of convents by quantizing or otherwise approximating the convolutional layer. In papers [5], [6], the redundant connections are reduced by pruning while maintaining performance. Some work ([8], [11], [19]) proposes quantization to reduce redundancy in calculations to speed up inference.

From the hardware perspective, specific architecture and modules are designed to reuse data, enhance "locality" of data,and accelerate convolution operations. Prior work has also been demonstrated on various embedded System-on-Chips (SoC) such as Nvidia DGX-1 and ARM-based cpu as well as FPGAs([14], [20]).

The weights in convolutional layers of CNN are used for multiple times in computation, and thus the overall performance can be significantly degraded by accelerate convolutional layers. Due to the complexity of the convolutional layer, some work has addressed the computation by unrolling the 2D convolution to matrix multiplication [16] or computing the 2D convolution in the frequency domain as a Hadamard product [15]. The computational bottleneck of frequency domain method is the Fourier transform. The 2D Fourier transform can be efficiently computed using Fast Fourier Transforms (FFTs) with complexity $O(N^2\log_2 N)$, where the output of convolution layer is $N \times N$.

Several authors proposed to use Fast Fourier Transform (FFT) based convolution method on GPU to speed up computation of the CNN. The advantages of FFT method compared to other convolution methods are rapid element-wise products and re-usability of transformed feature/filter maps([12], [15], [17]).

As far as we know, using FFT to reduce the convolutional layer computation complexity is studied first by Mathieu et al. [15]. Then, this method is refined by Vasilache et al. [17] and implemented in the NVIDIA cuDNN library [4]. The Strassen algorithm for fast matrix multiplication Cong and Xiao [1] is to reduce the number of convolutions in a computational layer, thereby reducing its total arithmetic complexity. The authors also suggested that more techniques from arithmetic complexity theory might be applicable to computation.

IV. Overlap-and-add Algorithm

Algorithm 1 is 1-D overlap-and-add method for spacial convolution. The input of overlap-and-add method is x of size N_x and h of size M. The input array x is first split into smaller blocks that are the size of L ($L \geq M$). Smaller convolutions are computed between the kernel and the block inputs. The resulting convolutions are overlapped by L, and added together to create the same results as a traditional spacial convolution.

The total complexity for the entire input and kernel is the number of blocks times the complexity of each block convolution, i.e.,$O(N^2log_2L)$. Since $L \geq M$, then the total complexity is $O(N^2log_2M)$ when $L = M$.

1-D overlap-and-add method can easily generalize to 2-D convolution. The inputs of x are $N_1 \times N_2$, and the kernels of h are $M_1 \times M_2$. x can be broken up into smaller blocks of dimensions $L_1 \times L_2$ resulting in smaller FFTs. Let $P_1 = N_1 / L_1$ and $P_2 = N_2 / L_2$. So

$$CONV2(x,h) = \sum_{i=1}^{P_1}\sum_{j=1}^{P_2}(CONV2(x_{ij},h)). \tag{3}$$

Algorithm 1 Overlap and Add (One Dimension)

```
1:  state
2:  $N_x = length(x)$;
3:  $M = length(h)$;
4:  $N = N_x + M$
5:  $H = FFT(h, L)$;
6:  i=1;
7:  while $i <= N_x$ do
8:      $i_1 = min(i + L - 1, N_x)$
9:      $y_t = IFFT(FFT(X(i : i_1), L) * H, L)$;
10:     $k = min(i + N - 1, M + N_x - 1)$;
11:     $y(i : k) = y(i : k) + y_t(1 : k - i + 1)$;
12:     $i = i + L$;
13: end while
```

Each convolution $CONV2(x_{ij},h)$ can be efficiently computed in the frequency domain, where:

$$CONV2(x_{ij},h) = IFFT2(FFT2(x_{ij}) \cdot * FFT2(h)) \tag{4}$$

The complexity of each 2-D fast Fourier transform $O(L_1L_2log_2L_1L_2)$. The total complexity for the entire input and kernel is the number of blocks times the complexity of each block convolution, i.e., $O(N_1N_2log_2L_1L_2)$. Then, the total complexity will be $O(N_1N_2log_2M_1M_2)$ when $M_1 = L_1$ and

$M_2 = L_2$. Although the overhead of FFT and additional operations due to zero-padding and stride, the performance increase of the OA technique can outweigh these overhead costs.

```
Algorithm 2 Overlap and Add (Two Dimension)
1: start
2: Let $M_1 \times M_2$ be the size of $h$ ;
3: Let $N_1 \times N_2$ be the size of $x$ ;
4: Set $x$ break into non-overlapping blocks of dimensions $L_1 \times L_2$;
5: Set zero pad $h$, it has dimensions $(L_1 + M_1 - 1) \times (L_2 + M_2 - 1)$;
6: Do $H = FFT2(h)$;
7: i=1,j=1
8: while $i <= P_1$ and $j <= P_2$ do
9:     Set zero pad $x_{ij}(n_1, n_2)$to be of dimensions $(L_1 + M_1 - 1) \times (L_2 + M_2 - 1)$;
10:    Do $X_{ij} = FFT2(x_{ij})$;
11:    $Y_{ij} = X_{ij}H$;
12:    $y_{ij} = IFFT2(Y_{ij})$;
13: end while
14: Find $y(n_1, n_2)$ by overlap and adding the last $(M_1 - 1) \times (M_2 - 1)$ samples of
15: $y_{ij}(n_1, n_2)$ with the first $(M_1 - 1) \times (M_2 - 1)$ samples of $y_{i+1,j+1}(n_1, n_2)$ to get the result.
```

V. EXPERIMENTS AND RESULTS

A. *Experimental Setup*

The target hardware for the proposed implementation is ilinx Zynq-7000 platform. Note that Zynq comprises two main parts: a Processing System (PS) formed around a dual-core ARM Cortex-A9 processor, and Programmable Logic (PL), which is equivalent to that of an FPGA. It also features integrated memory, a variety of peripherals, and high-speed communications interfaces.

In this paper, we only use the dual-core ARM Cortex-A9 processor on Xilinx Zynq-7000 platform. To compute the FFT, ARM-based fftw-library is used. From version 3.3.1 the fftw-library includes support for the NEON hardware accelerator SIMD engine featured in the ARM cores.

B. *Time vs. number of kernels*

In this experiment we compare the required time to compute one convolutional layer as the number of kernels in the layer increases. The input data is of size 64×64, and each kernel is of size 5×5. Each number of kernels experiment is repeated 20 times and the results are averaged. The number of kernels is varied from 1 to 550 with a discrete step of 25.

Figure 1 shows the speed-up factor of OaAconv compared to FFTconv the convolutional as the number of kernels varies. We can see that OoAconv outperforms FFTconv at every step. With the number of kernels increases, the acceleration trend tends to be stable, keeps up to 2 times.

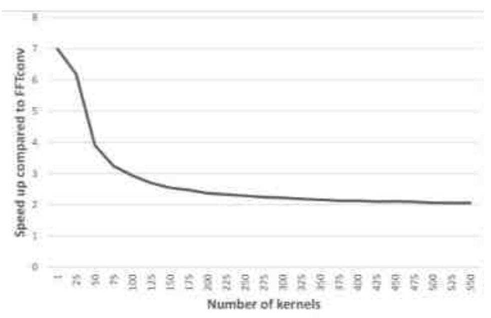

Fig.1 Speed-up OaAconv vs FFTconv with input size 64×64.

In order to further verify the acceleration effect of Ooaconv, we have done a set of experiments with the input size of 1024×1024 and 256×256, each kernel also is of size 5×5. This produced the results seen in Figure 2 and Figure 3. From the Figure 2, we can see that this increases the performance by only 1.15x speed-up factor with the input size of 256×256.

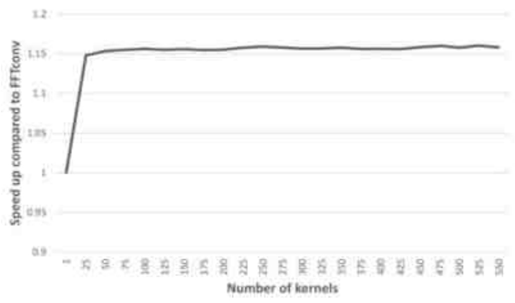

Fig.2 Speed-up OaAconv vs FFTconv with input size 256×256.

Figure 3 shows the speed-up over OaAconv vs. FFTconv with the input data size of 1024×1024. Compared with the previous test results to verify whether the different input block size can lead to large difference on OoAconv algorithm acceleration effects.

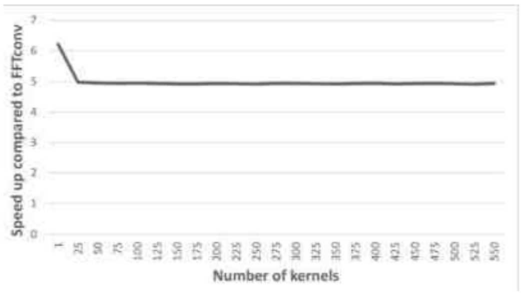

Fig.3 Speed-up OaAconv vs FFTconv with input size 1024×1024.

As can be seen from the results, for different input block size, through the accelerated operation OoAconv algorithm computes on different number and size of small pieces, resulting in differences in computing time, and with the number of kernels increases, the acceleration effect will gradually stabilize.

C. *Time vs. kernel size*

In this experiment, we vary the size of the kernel while keeping the input size constant. The number of kernels used is also held constant at 125.

The kernel size vary from 1x1 to 63x63 with a discrete step of 2x2. The input data of size is 1024×1024, 256×256 and 64×64. Each "kernel size" experiment is repeated 20 times and the results are averaged. Figure 4 to Figure 6 show the speed-up over OaAconv vs. FFTconv with the input data size of 1024×1024, 256×256 and 64×64.

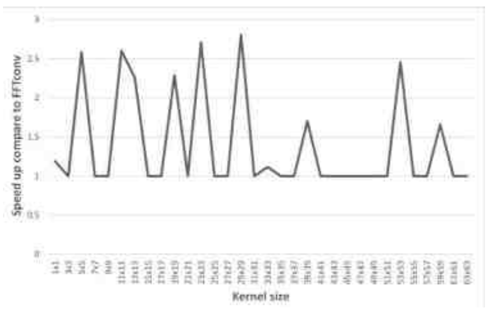

Fig.4 input size 64x64, number of kernel 125.

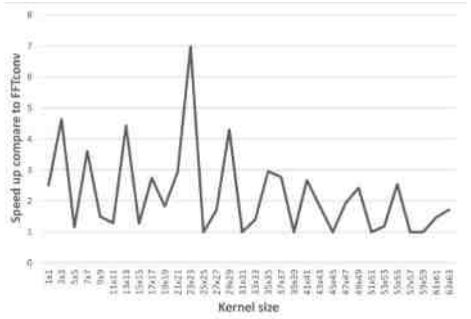

Fig.5 input size 256x256, number of kernel 125.

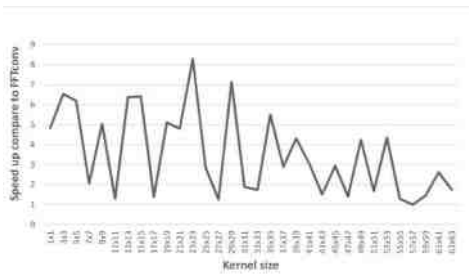

Fig.6 input size 1024x1024, number of kernel 125.

It can be seen from the above three figures, for different kernel size and input size, the corresponding acceleration will be different. And for the input size of 64×64, OoAconv algorithm doesn't show a clear acceleration effect, but as for the overall trend or it has a very good acceleration effect.

VI. Discussion and Future Work

In this paper, we exploited Overlap-and-Add method to reduce the computation complexity of the convolutional layer.The OaA method is implemented on zynq platform only using a dual-core ARM Cortex-A9 processor to accelerate convolutional layers. Experimental results show the speed-up over OaAconv vs. FFTconv about 2x. In the future we plan to explore concurrent processing the CPU and the FPGA on Xilinx Zynq-7000 platform.

References

[1] Cong, Jason, and Bingjun Xiao. "Minimizing computation in convolutional neural networks." International conference on artificial neural networks. Springer, Cham, 2014.

[2] Conti, Francesco, and Luca Benini. "A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters." 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2015.

[3] Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations." Advances in neural information processing systems 28 (2015).

[4] cuDNN. https://developer.nvidia.com/ cudnn. Accessed: 2015-11-01.

[5] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149 (2015).

[6] Han, Song, et al. "Learning both weights and connections for efficient neural network." Advances in neural information processing systems 28 (2015).

[7] Sim, Jaehyeong, et al. "14.6 a 1.42 tops/w deep convolutional neural network recognition processor for intelligent ioe systems." 2016 IEEE International Solid-State Circuits Conference (ISSCC). IEEE, 2016.

[8] Jaderberg, Max, Andrea Vedaldi, and Andrew Zisserman. "Speeding up convolutional neural networks with low rank expansions." arXiv preprint arXiv:1405.3866 (2014).

[9] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Communications of the ACM 60.6 (2017): 84-90.

[10] Karpathy, Andrej, et al. "Large-scale video classification with convolutional neural networks." Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 2014.

[11] Lebedev, Vadim, et al. "Speeding-up convolutional neural networks using fine-tuned cp-decomposition." arXiv preprint arXiv:1412.6553 (2014).

[12] Lavin, Andrew, and Scott Gray. "Fast algorithms for convolutional neural networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[13] LeCun, Yann, Fu Jie Huang, and Leon Bottou. "Learning methods for generic object recognition with invariance to pose and lighting." Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.. Vol. 2. IEEE, 2004.

[14] Levine, Sergey, et al. "End-to-end training of deep visuomotor policies." The Journal of Machine Learning Research 17.1 (2016): 1334-1373.

[15] Mathieu, Michael, Mikael Henaff, and Yann LeCun. "Fast training of convolutional networks through ffts." arXiv preprint arXiv:1312.5851 (2013).

[16] Scherer, Dominik, Hannes Schulz, and Sven Behnke. "Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors." International conference on Artificial neural networks. Springer, Berlin, Heidelberg, 2010.

[17] Vasilache, Nicolas, et al. "Fast convolutional nets with fbfft: A GPU performance evaluation." arXiv preprint arXiv:1412.7580 (2014).

[18] Wang, Naiyan, and Dit-Yan Yeung. "Learning a deep compact image representation for visual tracking." Advances in neural information processing systems 26 (2013).

[19] Wang, Min, Baoyuan Liu, and Hassan Foroosh. "Design of efficient convolutional layers using single intra-channel convolution, topological subdivisioning and spatial" bottleneck" structure." arXiv preprint arXiv:1608.04337 (2016).

[20] Zhang, Chen, et al. "Optimizing FPGA-based accelerator design for deep convolutional neural networks." Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays. 2015.