Accelerating Deep Neural Networks Using FPGA

Esraa Adel[1], Rana Magdy[1], Sara Mohamed[1], Mona Mamdouh[1], Eman El Mandouh[2], and Hassan Mostafa[1, 3]
[1]Electronics and Electrical Communications Engineering Department, Cairo University, Giza, Egypt
[2]Mentor Graphics Corporation
[3]Nanotechnolgy Department at Zewail City of Science and Technology, Giza, Egypt
Emails: esraaa_adel17@yahoo.com, Sara_AbdAlWahab@mentor.com, rana.omran95@eng-st.cu.edu.eg, mona.mamdouh@hotmail.com, eman_mandouh@mentor.com, hmostafa@uwaterloo.ca

Abstract— **Deep Convolutional Neural Networks (CNNs) are the state-of-the-art systems for image classification and scene understating. They are widely used for their superior accuracy but at the cost of high computational complexity. The target in this field nowadays is its acceleration to be used in real time applications. The solution is to use Graphics Processing Units (GPU) but many problems arise due to the GPU high-power consumption which prevents its utilization in daily-used equipment. The Field Programmable Gate Array (FPGA) is a new solution for CNN implementations due to its low power consumption and flexible architecture. This work discusses this problem and provides a solution that compromises between the speed of the CNN and the power consumption of the FPGA. This solution depends on two main techniques for speeding up: parallelism of layers resources and pipelining inside some layers**

***Index Terms*—Convolutional Neural Networks (CNNs); Alex-Net; Accelerating CNNs; FPGA; Virtex7.**

I. Introduction

In recent years, artificial intelligence and deep learning have shown their utility and effectiveness in solving many real-world problems. The main motivation is to eliminate the need of direct programming and create an intelligent system that can automatically adapt to new situations. Among various deep learning algorithms, CNNs are the state-of-the-art in computer vision problems such as face recognition [1], and autonomous driving [2].

While CNNs have significant higher accuracies than traditional algorithms, they require huge amounts of computational resources and memory access due to the convolution-operation large number of parameters, which represents a computational challenge for the General-Purpose Processors (CPUs) and consumes large amount of power. Recently many applications such as embedded systems in self-driving cars need high energy efficiency and real-time performance. As a result, hardware accelerators such as GPU, FPGA, and Application Specific Integrated Circuits (ASIC), have been utilized to improve the throughput of the CNN.

As CNNs offer significant potential for massive parallelization and extensive data reuse, therefore GPUs are the most widely used platforms to improve both training and classification processes of CNNs [3], thanks to their high throughput and memory bandwidth. However, GPUs consume a considerable amount of power which is another important performance evaluation metric in the modern digital systems.

ASIC design, on the other hand, has achieved high throughput with low power consumption [4], [5], but the development time and cost are significantly high compared to other solutions. Recently the new generation of FPGA increases the capacity of hardware resources continuously, which provides thousands floating-point computing units and low power consumption. Therefore, FPGA-based accelerators are efficient alternatives that provide high throughput, low power consumption, and configurability at a reasonable cost.

Many approaches have been used to implement CNN on FPGA. The authors of [6] provide a comparison between GPU and FPGA for DNNs. They propose a detailed case study on accelerating Ternary Res-Net, the results are very promising; Stratix 10 performance is 10%, 50% and 5.4x better in performance (TOP/sec) than Titan X Pascal GPU, results indicate that FPGAs may become the platform of choice for accelerating DNNs. The work in [4] propose an energy-efficient dataflow called row stationary, which aims to maximize the reuse and accumulation at the local memory level (RF or caches) for all types of data (weights, pixels and partial sums). The work in [7] transforms a convolutional layer into a regular matrix-multiplication (MM) in the Fully Connected (FC) layer, and implements an MM-like accelerator for both layers. The other work in [8] takes an opposite approach: it transforms a regular MM into a convolution, and implements a convolution accelerator for both convolutional and FC layers.

In this work, an architecture is proposed to accelerate CNN. This proposed architecture relays on parallelism for all kernels in the convolutional layer which is flexible for any network size and uses extensively local memories to store all the data, these proposed techniques correspondingly reduce the performance time and the power dissipated in external memory access. Results are provided for hardware utilization and power consumption, comparing the results with CPU performance. Reducing the power consumption by using fixed point operation instead of floating point. Applying the pipelining techniques to reduce the hardware resources and performance time.

This paper is organized as follows; Section II provides background on CNN. Section III discusses the hardware implementation. Section IV shows the hardware utilization results after performing the synthesis. Section V discusses the pipelining optimization technique and shows the reduction in resources. Section VI makes a comparison between the proposed hardware architecture and software results. Section VII concludes the proposed work and presents the future work.

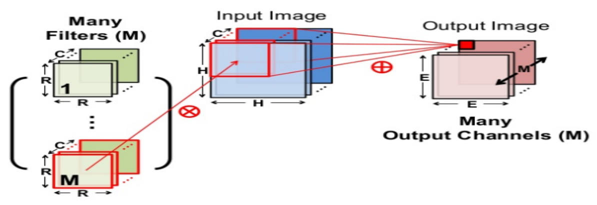

Fig. 1 Convolution operation [4].

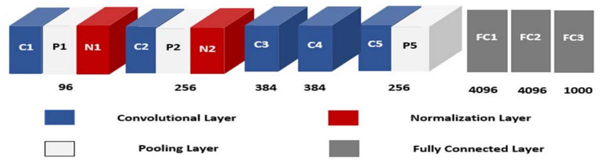

Fig. 2. Alex-Net neural network architecture.

II. BACKGROUND

CNNs are computational models inspired by the way of the human brain. CNNs operation has two phases, training phase and inference phase (feed-forward path). In the training phase, the CNN is trained on a known data set to learn its weights to minimize the error. This work focuses on the feed-forward path of a pre-trained CNN.

A typical CNN feed-forward path consists of a feature extractor and a classifier. The feature extractor extracts an input-image features across the CNN layers which are; Convolutional (Conv), Rectified Linear Unit (ReLU), Pooling (Pool) and Local Response Normalization (LRN). Then the feature extractor sends these extracted image-features to the classifier which is implemented using FC. In order to understand the proposed hardware implementation, the CNN detailed layers will be discussed in this section, also Alex-Net architecture will be discussed, which is one of the state-of-the-art CNN models will be presented.

A. Convolutional layer

The Conv layer is the core building block of a CNN that does most of the computational heavy lifting. It's always the first layer in a CNN. The convolution operation is done by applying element-wise multiplication and accumulation between input feature maps and the weight filters (called also kernels), which are the Conv operands obtained from the training phase, then sliding these filters over the input feature maps as shown in . Each of these weight filters can be thought of as feature identifiers.

The computation is given in (1), where M is the number of output feature maps (number of filters) of size E × E, C is the number of channels in Input feature maps, and R × R is the size of the Filter.

$$out[m][h_o][w_o] = b_i + \sum_{i=1}^{C} \sum_{K_h=0}^{R} \sum_{K_w=0}^{R} IN\ [i][h_o + k_h][w_o + k_w] * Kernel[m][i][k_h][k_w] \qquad (1)$$

B. Pooling layer

The Pool layers (also called sub-sampling) reduces the dimensionality of each feature map of its input feature maps, but retains the most important information. The number of output feature maps is identical to that of input feature maps, while the dimensions of each feature map scale down according to the size of the sub-sampling window (called also kernel).

C. Rectified Linear Unit (ReLU)

The ReLU layer introduces a non-linear operation. The ReLU performs after every Conv layer. Its output is given by; max (0, input). The purpose of ReLU is to introduce non-linearity in the CNN after linear operation of convolution, since most of the real-world data the network required to learn is non-linear to generalize or adapt with variety of data

D. Local Response Normalization (LRN)

The LRN reduces top-1 and top-5 error rates by 1.4% and 1.2%, respectively [9]. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real human neurons. The role of the LRN layer is to normalize around the local neighborhood of the excited neuron and make it even more sensitive as compared to its neighbors to avoid the saturation in network. The normalization functionality is given in (2).

$$out^i = in^i_{(x,y)} / (k + \propto \sum_{j=\max(0,i-\frac{n}{2})}^{\min(N-1,i+\frac{n}{2})} \left(in^j_{(x,y)}\right)^2)^\beta \qquad (2)$$

E. Alex-Net

Alex-Net is one of the state-of-the-art CNN, it won the 2012 ILSVRC (Image-Net Large-Scale Visual Recognition Challenge). It is the first model to achieve top-1 and top-5 error rates of 37.5% and 17.0% respectively on the test data of Image-Net dataset [10], which is an astounding improvement compared with the other top models in the context.

The CNN developed by Krizhevsky, Sutskever, and Hinton in 2012, consists of five Conv layers, some of them are followed by Pool layers, two LRN layers, and three fully-connected layers with a final 1000-way soft-max [11].

III. HARDWARE ARCHITECTURE

The main purpose is to accelerate the CNN, so the proposed architecture relays mainly on parallelism. The parallelism role is to reduce the time needed for image classification in Alex-Net architecture. In addition, some techniques as pipelining and on chip (local) memory usage are used for more acceleration which will be discussed in details showing the effect of the all techniques on the overall CNN speed.
In this section, each layer hardware-implementation perspective will be discussed in details.

A. Convolutional layer

The Conv layer operations mainly depend on MAC operations (multiplication and accumulation) within each filter with the input feature maps; these MAC operations are done using the smallest basic unit which is the Parallel Engine (PE) composed of multiplier and accumulator.
The parallel execution of the weight filters is done by duplicating these PEs, for example for executing the Conv1 layer, 96 PEs are used which is equal to the number of the weight filters of Conv1 layer. The number of parallel PEs can be customized based on the compromise between the number of available resources on FPGA and the required CNN speed, taking into consideration that by increasing the number of PEs the speed is much faster. Figure 3 shows the parallel structure of PEs used for Conv1 layer,

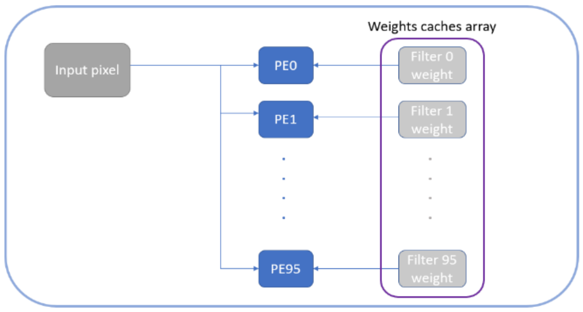

Fig. 3. Convolutional layer architecture

each PE is responsible for one weight filter operations as discussed before. This parallel structure speeds up the Conv1 execution by 96 times (number of filters). This will be further explained in the results sections showing the time reduction resulted from this parallelism.

B. *Pooling layer*

The Pool layer operations mainly depend on max-pooling operations as discussed in Section II, these max-pooling operations are done using the smallest basic unit which is a parallel engine composed of a comparator and a register.

The Pool layer hardware is similar to the Conv layer hardware from the parallelism technique perspective; the Pool kernels across the different input feature maps are executed in parallel. The parallel structure of the Pool layer is composed of number of parallel PEs equal to the depth of the input to the Pool layer where each filter is applied on one depth. The Pool layer execution time is negligible compared to the Conv layer execution time, subsequently the optimization techniques is mainly concerned with reducing the execution time of the Conv layer which will be discussed in details in the later sections.

C. *Local response normalization layer*

As discussed in the Section II, the LRN functionality is normalizing around the local neighborhood of the excited neuron and makes it even more sensitive as compared to its neighbors. This functionality is accomplished by equation (2).

The hardware implementation proposed as follows:

- Multiple input caches are needed for LRN Layer as it needs to access the whole input feature maps simultaneously to be able to implement the summation process. Subsequently the number of input caches in this design is equal input feature maps of this LRN layer.
- Getting the denominator result of (2), the summation of squares is done by a tree of adders that adds the input squares then multiply it by α and add k to the result.
- Implementing customized combinational fixed-point division, the proposed idea is to use two combinational integer divisions; the first one calculates the integer part and the second calculates the fraction part. In this CNN, the divider is always larger than one, so the fixed-point division is customized to work properly only when the divider is larger than one and any other values won't act properly. Sign bit is considered but actually no need for it because the Pool and ReLU output is always positive.

TABLE I
HARDWARE RESOURCES FOR FIRST SUPER LAYER

Layer\ Resources	**DSPs**	**LUTS**	**BRAMs**	**Registers**
Conv1	96	8049	48	5736
Max Pooling 1	2	8022	192	3226
Norm 1	7	16897	48	248

- Output storing technique: the output elements are stored in one cache if the following layer is group one Conv layer or two caches if the following layer is group two Conv layer.

D. *Fully connected layer*

The Alex-Net architecture consists of 2 main parts as mentioned in the Background section: feature extractor which is found to be enough for classification but adding a classifier improves the network accuracy. This classifier consists of a series of FC layers. The FC layer is mainly based on matrix-vector multiplication, the matrix consists of weights obtained from the training phase and the vector is a group of features resulting from the feature extractor part in the CNN. Each element in the output vector is a weighted sum of the input vector that`s why it`s called fully connected. The main idea of the proposed design to guarantee accelerating this layer more than the software is based on two main techniques:

- The main idea of parallelism in this layer is using parallel engines of the building block which consists of a multiplier and accumulator the same as Conv layer corresponding to the number of rows of the weights' matrix but due to the large number of rows only part of the rows is taken in parallel and after getting the outputs corresponding to these rows, other rows are taken.
- For more speeding up, pipelining is used. In the layer design there is a cache for each PE at its weight port and only one cache for all the PEs at the output. So, pipelining occurs in the weights cache as a weight of a certain row is used, it becomes useless so while computing the next weight, the weight of the next row corresponding to the previous weight is being stored in the cache so that the MAC operations of the rows are done one after the other without any waste of time between each row and the preceding one.

IV. SYNTHESIS

After performing the synthesis on the first super layer of the design; which consists of (Conv, Pool and LRN layers) on Virtex-7 VC709 FPGA using the Vivado 2015.2 synthesis tool, Table I shows the utilization of the resources.

Discussing some of the results shown in table I, for example, Conv1 Layer consists of 96 DSPs corresponding to the number of parallel engines which is equal to the depth of this layer as discussed before and the DSPs in the Pool and LRN layer are for the addressing equations.

V. PIPELINING

In order to utilize the un-utilized time between layers and reduce the hardware resources, the pipelining technique is applied between the Conv and Pool layers. Typically, in Alex-Net Pool1 parameters are (kernel size=3x3, stride=2). Therefore, the Pool layer can start its operation after the first three rows in

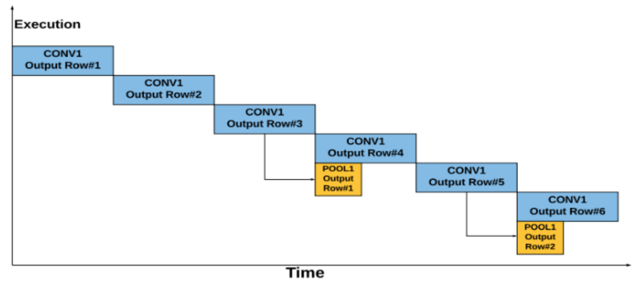

Fig. 4. Execution of convolution and pooling stages in pipelining.

TABLE II
COMPARISON BETWEEN HARDWARE RESOURCES AFTER PIPELINING

Point of comparison	**Original**	**Pipelined design**
Utilization: LUTs	11%	13%
Utilization: BRAMs	22%	9%
Power	1.141 W	1.071 W

Conv1 output are completed, that can produce the first row of Pool1 output. For the second row of the Pool1 can be executed after the fourth and fifth rows of Conv1 are completed. Fig.4 shows the pipelining flow in time. After Pool1 generates complete row output, Conv1 can over-write its output that would reduce the memory storage for the convolution output to store only four rows.

Table II shows that the utilization resources are reduced compared to the original design, especially in the BRAMs which are decreased from 22% to 9% which is considered a significant reduction that saves the area and power.

VI. RESULTS

The main purpose is to accelerate Alex-Net architecture for image classification, the functionality is verified on the RTL with reasonable accuracy and the timing results are obtained and compared to the MATLAB R2014a execution-time to show how much the CNN speed is enhanced.

Table III shows the execution time of some layers of the MATLAB R2014a (CPU) and the one of the accelerated CNN on FPGA. It's clear that the bottleneck of the CNN is the Conv layer, therefore it is the first design consideration to be accelerated as discussed in the previous section.

Comparative study to the GPU is summarized in table IV. The mentioned results show that the FPGA is faster than the CPU but slower than GPU, but it is still preferred over the GPU due to the FPGA lower power consumption which is proved by estimating the dynamic power consumption of the first three layers (Conv-Pool-LRN)using Virtex7 FPGA which is 0.785 watt and that estimation is reasonable compared to the power consumed in [12] which is 7.2 watt for the first ten layers which is lower than the power consumed by the GPU as mentioned in [13] that Titan X GPU throughput of 3.23 TOP/s comes at a relatively high-power cost which is 250 W.

III. CONCLUSION AND FUTURE WORK

In this paper, the acceleration of the forward path of a pre-trained Alex-Net on FPGA is demonstrated, introducing the parallelism and pipeline techniques used to accelerate the CNN.

The proposed architecture is discussed in details illustrating the achieved high speed and low power performance.

Future work concerns deeper analysis of particular mechanisms, new proposals to try different methods. The

TABLE III
SIMULATION TIME OF S/W AND RTL

Layer	**MATLAB simulation time (in Secs)**	**FPGA virtual simulation time**
Conv1	7.8	11 ms
Pool1	0.37	65.6 µs
Norm1	1.83	700 µs
Conv2	7.9	8.7 ms
Pool2	0.3	1690 ns
Norm2	1	432.64 µs

TABLE IV
SIMULATION TIME OF Alex-Net ON DIFFERENT GPUs

Hardware-Accelerator	**Execution time (ms)(Forward path)**
Proposed Architecture	40.94
Pascal Titan X	5.32
GTX 1080	7.00
Maxwell Titan X	7.09

next step is to compromise between the time required for the image prediction and the number of resources. In addition, some techniques can be used to reduce the power consumption as pruning and partial dynamic reconfiguration (PDR).

ACKNOWLEDGMENT

This work was partially funded by Mentor Graphics and ONE Lab at Zewail City of Science and Technology, Egypt and Cairo University, Egypt.

REFERENCES

[1] M. Coşkun, A. Uçar, Ö. Yildirim and Y. Demir, "Face recognition based on convolutional neural network," in *MEES*, 2017.

[2] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *IVS*, 2017.

[3] V. Sze et al., "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *arXiv preprint arXiv:1703.09039*, 2017.

[4] Y. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss : An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *ISSCC*, 2016.

[5] A. Elnabawy, H. Abdelmohsen, M. Moustafa, M. Elbediwy, A. Helmy, and H. Mostafa, "A Low Power CORDIC-Based Hardware Implementation of Izhikevich Neuron Model," in IEEE International NEWCAS, 2018.

[6] E. Nurvitadhi et al., "Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks," in *ISFPGA*, 2017.

[7] N. Suda et al., "Throughput-optimized opencl-based fpga accelerator for largescale convolutional neural networks," in *FPGA. ACM*, 2016. [Online]. Available: http://www.mit.edu/~wsshin/maxwellfdfd.html. [Accessed Nov. 14, 2017].

[8] K. J. Qiu et al., "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA. ACM*, 2016.

[9] A. Beam, "Deep Learning 101 - Part 1: History and Background, " Feburary , 2017. [Online]. Available: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html. [Accessed Sept. 12, 2017].

[10] A. Krizhevsky, "ImageNet Classification with Deep Convolutional Neural Networks, " 2015. [Online]. Available: https://www.coursehero.com/file/24481166/alexnet-tugcekyungheepdf/. [Accessed Nov. 14, 2017].

[11] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *NPIS*, 2012.

[12] Y. Kang, S. Kim, T. Shin and J. Chung, "Running Convolutional Layers of AlexNet in Neuromorphic Computing System," NRF-2014R1A1A2A16055253.

[13] P. Matthias Gysel, "Ristretto: Hardware-Oriented Approximation of Convolutional Neural Networks," B.S. thesis, Bern University of Applied Sciences ,Switzerland, 2012.