# Comparative Study of Hardware Accelerated Convolution Neural Network on PYNQ Board

Alaa M. Salman[2,*], Ahmed S. Tulan[1,*], Rana Y. Mohamed[2,*,] Michael H. Zakhari[1,*], and Hassan Mostafa[1, 3]
*Electronics and Communication Department, Faculty of Engineering, Cairo University, Giza 12613, Egypt*
*Electronics and Communication Department, Faculty of Engineering ,Alexandria University, Egypt*
*University of Science and technology Nanotechnology Department Zewail, October Gardens, 6th of October, Giza 12578, Egypt City*

***Abstract*— In recent years convolutional neural networks (CNNs) have been remarkably used in many applications, and they are the heart of many intelligent systems. The advancements in both new electronic design automation (EDA) tools and in new hardware development boards such as Python Productivity for Zynq (PYNQ) have significantly decreased the development time of CNNs. However, the short time-to-market is at the cost of implementation area, performance and power consumption. Over the last period, CNNs' energy consumption needs have skyrocketed dramatically. Thus, In this work, the authors conduct a comprehensive study on the power consumption of hardware accelerated CNN whether implemented using new EDAs High Level Synthesis (HLS) or the basic design abstraction of Register Transfer Level (RTL). Both methods are implemented on modern development boards from Xilinx (as PYNQ). Modern EDAs flow such as HLS does not represent the best environment for a good power consumption. The power consumption of the HLS implementation is six times more power than the RTL one. It is concluded that the new EDAs method have a deficiency to deliver highly efficient CNNs but it has the ability to deliver sufficient results within a very short period of time.**

***Keywords* — CNN, LeNet, PYNQ, ZYNQ, HLS.**

## I. INTRODUCTION

Convolutional Neural Network is an artificial deep learning neural network with a history of research that can be traced back to the second half of the 20th century [1]. CNN consists of many layers that operates on the input image. The convolution layers identify the features of the image, an activation function increase the non-linearity of the image, pooling layers reduce the number of parameters of the input, fully connected layer that is a final straight line before the finish line where all the things are already here, output layer outputs an array of probabilities for the corresponding classes.

CNNs are now occupying a huge role in developing new Artificial Intelligence (AI) applications, in everything from industry to business. Within the increasingly extensively range of applications, industry is eager to find new ways to develop optimized CNN in a very short time-to-market. Now it is possible to develop CNN with C++ using High Level Synthesis. This has led to rapid decrease in the development time of hardware accelerated CNNs. Section II discusses the CNN architecture for the proposed case. Section III discusses the implementation of the CNN architecture using RTL. Section IV discusses PYNQ capabilities to run CNN architectures using Jupyter Notebook framework. Section V discusses the implementation of the CNN architecture using Vivado HLS tools. Section VI discusses the comparative case study between the RTL and HLS from power aspect. And finally in Section VII discusses the optimizations that can be done based on the results in Section VI.

## II. Proposed CNN Architecture

The proposed CNN architecture is LeNet-5 shown in figure 1. This section describes more details about the architecture, LeNet-5 consists of 7 layers each layer contains trainable parameters(weights). The input is a 32x32 pixel image.

Layer C1 is a convolutional layer with 6 feature maps. Each unit in each feature map is connected to 5x5 neighborhood in the input. The size of the feature maps is 28x28. C1 contains 156 trainable parameters and 122,304 connections.

Layer S2 is a sub-sampling with 6 feature maps of size 14x14. Each unit in each feature map is connected to a 2x2 neighborhood in the corresponding feature map in C1. The

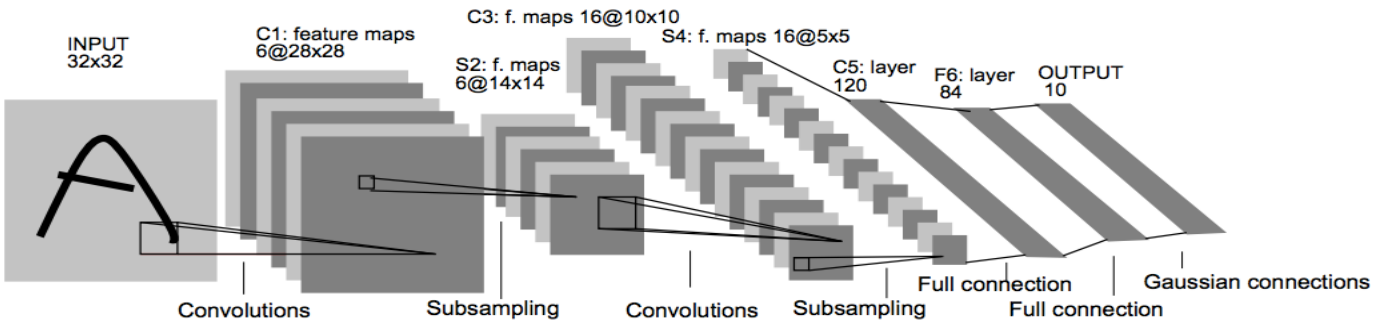


*Figure 1. Depicts LeNet-5 layers*

four inputs to a unit in S2 are added to a trainable bias. The result is passed through a sigmoid function.

Layer C3 is a convolutional layer with 16 feature maps. Each unit in each feature map in connected to several 5x5 neighborhoods at identical locations in a subset of S2's feature maps. Layer C3 has 1,516 trainable parameters and 151,600 connections.

Layer S4 is a sub-sampling layer with 16 feature maps of size 5x5. Each unit in each feature map is connected to a 2x2 neighborhood in the corresponding feature map in C3, in a way as C1 and S2. Layer S4 has 32 trainable parameters and 2,000 connections.

Layer C5 is a convolutional layer with 120 feature maps. Each unit is connected to a 5x5 neighborhood on all 16 of S4's feature maps. The size of C5's feature maps is 1x1 as the size of s4 is also 5x5. Layer C5 has 48,120 trainable connections.

Layer F6, contains 84 units this number is dependent on the design of the output layer and is fully connected to C5. It has 10,164 trainable parameters.

## III. Register Transfer Level

RTL is commonly used in Hardware Description languages (HDL) like Verilog and VHDL to create high level representations of circuits. This is an attempt to implement a hardware CNN architecture with RTL to compare with HLS one. The code is written with Verilog and synthesized on Xilinx FPGA using Vivado. The code is experimental for function and not fully optimized. However, it is sufficient to compare RTL and HLS.

The LeNet is constructed using elementary modules. The weights and biases are hard coded in an external read-only memory (ROM). Four elementary modules are implemented: conv, max-pool, rectified linear unit (Relu) and iterator. The conv performs the convolution computing and works also as the fully connected layer with kernel size equal to the input data size. The iterator is responsible for jogging around the input data and feeding the computing units.

The difficulties come out from the fact that the weights and intermediate results need to be stored in an external memory and the data iterator will become more and more complex with scaling the network.

## IV. PYNQ DEVELOPMENT BOARD

AI is changing the fundamental nature of hardware development. Xilinx developed PYNQ System on Chip (SoC) which is shown in Figure 2. PYNQ ease the use of FPGAs for embedded systems development. PYNQ architecture - ZYNQ architecture in general consists of two main parts: Programmable system (PS), which consists of ARM processors- which can run Linux OS. Programmable logic (PL) circuits are presented as hardware libraries called overlays. The Advanced Extensible Interface (AXI) interconnections provide high sharing of data between the PS and PL in designing the complete system; contains the data movement, consumes less time and is easier to program. PYNQ uses Python language and libraries to exploit the benefits of FPGA and microprocessors in ZYNQ to build more productive applications. PYNQ can be used as a software development tools to take advantage of the hardware capabilities, thanks to the integrated python environment. Also the PYNQ board can be used for rapid prototyping.

.Building and controlling the interface between the ARM processor and the FPGA on PYNQ is not as easy as it sounds. Xilinx produces an Intellectual Property (IP) that aids in the implementation of the ARM/FPGA interface as well as the FPGA/Memory interface. AXI with Direct Memory Access (DMA) [2] will be used in transmitting a stream of data between FPGA and DDR memory. AXI DMA focuses on boosting the throughput of transmitting a large stream of data and can support a throughput of one data word per clock cycle. Hardware. CNN requires transmission of large amount of data at very high speed between FPGA and DDR memory, so a combination of AXI4-streaming protocol and DMA can provide high data transmission throughput.

PYNQ is the first project to combine the following elements to simplify and improve All Programmable System-On-chip (APSoC) design:

1. A high-level productivity language (Python in this case)
2. FPGA overlays with extensive Application Programming Interfaces (APIs) exposed as Python libraries
3. A web-based architecture served from the embedded processors.
4. The Jupyter Notebook framework deployed in an embedded context

Training CNN is done by applying numerous tiny changes to different filter weights, and these tiny changes needs floating point precision [3] to obtain high accuracy, but running inference is different. One of the advantages of deep neural networks such as CNNs is that they tend to operate very well with high levels of noise in their inputs. For example during recognizing an object in a photo, the network has to ignore all the CCD noise [4], lighting changes, and other non-essential differences between the input photo and the training examples, and

*Table 1: the hardware resources usage using 32-bit floating point Convolution layer*

| Resource | Usage | Available | Percentage Usage (%) |
|---|---|---|---|
| BRAM | 54 | 140 | 38.6 |
| DSP | 76 | 220 | 34.5 |
| LUT | 1803 | 53200 | 3.39 |

focus on the important features instead. This ability means that they seem to treat low-precision calculations as just another source of noise, and still produce accurate results even with numerical formats that hold less information.

Neural networks can consume a lot of space on disk, for example AlexNet [16] consumes 200MB in float format, when implementing CNN layers using 32-bit floating-point data format. However, the hardware resources usage is found to be too large to implement on PYNQ's Zynq FPGA, in the convolution operation shown, with input feature map of dimension (1, 3, 32, and 32) and kernel of dimension (32, 3, 5, 5) the convolution can be converted into matrix multiplication of size (1024x75) x (75x32), and one forward-propagation of this layer contains 2.4576 MMACCs. Table 1 shows the resource usage for this layer.

Quantization can significantly reduce the resource usage, ensuring better scalability and more parallelism. It can be concluded that 8-bit and 16-bit fixed-point data formats [5], [6] provide the most optimal resource-accuracy trade-off. In order to reduce BRAM usage, 8-bit instead of 16-bit fixed point format will be used for LeNet [7], [8] deployment.

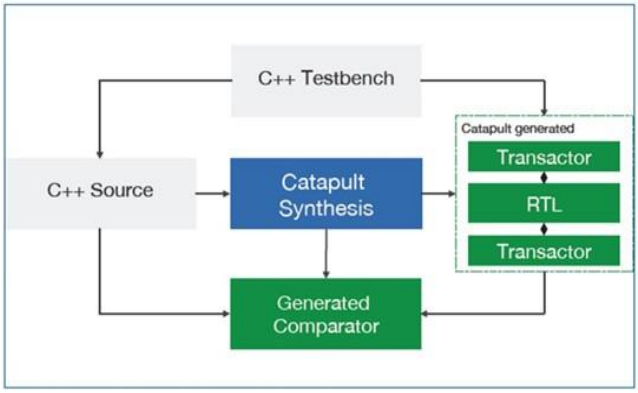


*Figure 2. High Level Synthesis design cycle*

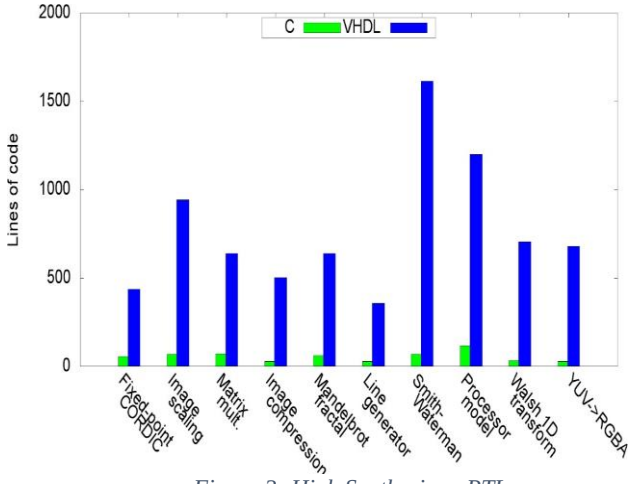


*Figure 3: High Synthesis vs RTL*

## V. High Level Synthesis

It is traditional to design FPGA IPs using component modules in RTL, despite of its efficiency and its optimum results has low readability, high difficulty to be modified and requires a lot of development time. HLS has been used to eliminate these drawbacks and make hardware development faster and easier. Vivado HLS [9] enabled Creation of IPs using C, C++ and System C codes to be directly converted to RTL without the need to create RTL manually. More and more are starting to use HLS as HLS IPs are spreading,

High-level Synthesis generates satisfying optimizations in a short-term design cycle as viewed in Figure 2. However, optimizations from HLS aren't as high as those obtained from other methods as RTL [10]. As it can imagined that by writing python vs writing assembly on a microcontroller. Of course, Assembly will take much time and effort, and as the level of abstraction get deeper in implementation the more time and effort will be consumed, but a more efficient design will be created. The fact that you are relying on a software tool to generate your design increases your productivity and cut the time to market significantly. Figure 3 shows a comparison between different implementation of different topics using VHDL and higher abstraction C vs the lines of codes which corresponds directly to the time spent on the

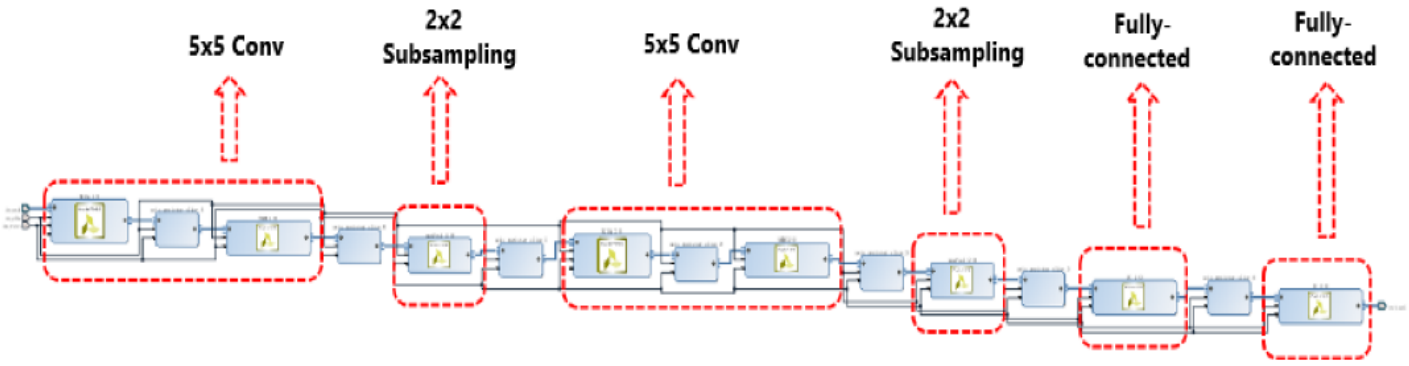


*Figure 4. SDF of the whole architecture*

implementation showing that developing RTL can take almost up to 16 times the effort vs the developing using HLS (smith-waterman)

RTL is a mature technology with a well-experienced developers and wide range of maintained IPs that you can use [11], [12]. In contrary, HLS is a new technology that is not very popular among Hardware Engineers, and Software Programming in C++/C is an extra headache for any Hardware Engineer. As there is a mad push toward accuracy of AI devices the world put much efforts in making bigger and more efficient accelerators for

AI. In adoption of more and more accelerators in big tech and business there is a lot of concerns about their power consumption and its effect on climate change.

The architecture is built using synchronous data flow (SDF) paradigm. A network is represented as a direct graph named synchronous data flow graph (SDFG). In SDFG shown in figure 4, nodes represent computations and lines represent data streams. Basically, the principle of SDF is that whenever input data are available for a node, the node will immediately start processing and generating the output. When the entire network is constructed using SDF, each component IP on the graph can independently drive the data streaming, forming a heterogeneous streaming architecture. With streaming IO, output data immediately streams out instead of being buffered in on-chip memory, hence saving memory footprint of the entire network [13]. The used FPGA Ips including convolution layer, pooling layer, fully-connected layer and Relu layer. These layers all employ the SDF model of computation as its bias.[13] Complete networks can be constructed as chains of these three layers Ips. With Vivado block diagram utility, engineers can build CNNs graphically.

## VI. Comparison on LeNet – PYNQ

Over the last period, CNNs' energy needs have skyrocketed dramatically. Thus, in this work, the authors conduct a comprehensive comparison on the power consumption of hardware accelerated CNN on the two presented methods on modern development boards from Xilinx e.g. PYNQ.

1. New EDAs as Vivado HLS in which its complete flow can be seen in Figure 5
2. The basic design abstraction as RTL.

LeNet architecture was first introduced by LeCun in 1998 [7], Gradient-Based Learning Applied to Document Recognition. The author implemented LeNet to be used primarily in recognition in documents. LeNet architecture is straightforward and small in terms of memory footprint making it a perfect match for a comparative study as it will be significantly easier to develop its RTL, to manage its data stream, and can be handled on different FPGAs scales. Also, the MNIST dataset is the most well-studied, understood dataset in machine learning history that makes it the most suitable dataset to be used in our work. In this experiment that took almost two month to develop, the authors ran hardware accelerated LeNet on PYNQ twice. In the first time, the LeNet was developed using RTL, and the second time, the LeNet was developed with HLS.

The results showed that the power consumption of RTL LeNet is 0.291 WATT and the HLS consumes six times more power than RTL LeNet with 1.981 WATT as shown in Figure 6. While the utilization of both HLS and RTL developed is shown in table 2.

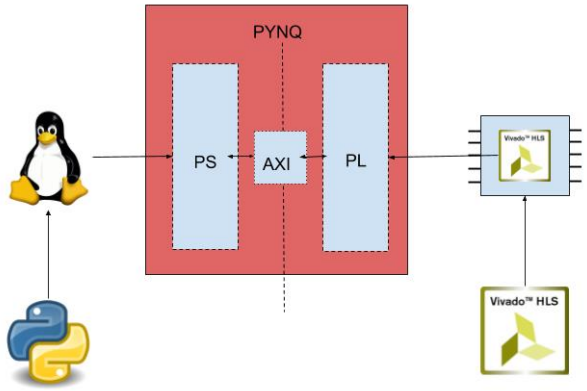


Fig. 5 PYNQ Board complete flow

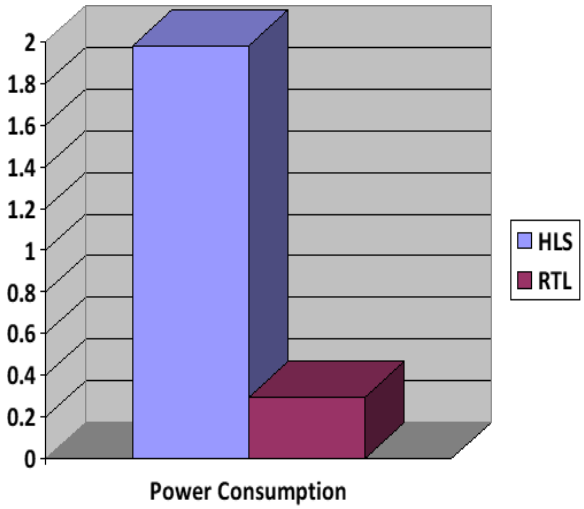


Fig. 6 High Level Synthesis vs RTL Power Consumption in Watt.

| Resource | HLS | RTL |
|---|---|---|
| **LUTs** | 38304 | 18467 |
| **Registers** | - | 7311 |
| **BRAMs** | 139 | 19 |

Table 2: the hardware resources utilization of both RTL and HLS development methods

## VII. CONCLUSIONS AND FUTURE WORK

The High-Level Synthesis model has unique features to decrease time-to-market and give developers a number of advantages and flexibility to maintain and edit their CNNs. In addition, it give an acceptable result [13] comparing to the developing efforts. At least for now HLS software cannot be depended on to get the optimum results. However, RTL is not the solution because of the very long time-to-market, efforts and expertise needed to finish the design.

In this paper, the authors comprehensively corroborate the deficiency of new EDAs method to deliver highly efficient CNNs with short time-to-market and emphasized that we still in need to develop more optimized way for fast CNN implementations.

## VIII. Acknowledgement

This work was partially funded by ONE Lab at Zewail City of Science and Technology and at Cairo University, NTRA, ITIDA, ASRT, and NSERC.

## REFERENCES

[1] Sanjay B Ankali, Dr. D V Ashoka, "Detection Architecture of Application Layer DDoS Attack for Internet," The International Journal of Advanced Networking and Applications (Int. J. Advanced Networking and Applications) 2011, vol. 03, no. 01, pp. 984-990

[2] Alaa M. Salman, “AXI Direct Memory Access” lauri.xn--vsandi-pxa.com. https://lauri.xn--vsandi-pxa.com/hdl/zynq/xilinx-dma.html (accessed Aug. 1, 2019).

[3] Daniele Bagni, A. Di Fresco, J. Noguera, F. M. Vallina, "A Zynq Accelerator for Floating Point Matrix Multiplication Designed with Vivado HLS", XAPP1170 (v2.0), January 21, 2016. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp1170-zynq-hls.pdf

[4] Ahmed S. Tulan, “How to Quantize Neural Networks with TensorFlow” petewarden.com/. https://petewarden.com/2016/05/03/how-to-quantize-neural-networks-with-tensorflow/ (accessed Oct. 1, 2019).

[5] Yao Fu, Ephrem Wu, Ashish Sirasao, Sedny Attia, Kamran Khan, and Ralph Wittig, "Deep Learning with INT8 Optimization on Xilinx Devices," Xilinx WP486 (v1.0.1) Apr. 24, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp486-deep-learning-int8.pdf

[6] Tim Dettmers, "8-BIT APPROXIMATIONS FOR PARALLELISM IN DEEP LEARNING," The International Conference on Learning Representations (ICLR). arXiv preprint arXiv:1511.04561v4, 2016

[7] Yann LeCun, Le´on Bottou, Yoshua Bengio, and Patrick Haffner, "GradientBased Learning Applied to Document Recognition," Proceedings of the IEEE, 86(11):2278–2324, 1998.

[8] Ahmed S. Tulan, “THE MNIST DATABASE.” yann.lecun.com/. http://yann.lecun.com/exdb/mnist/ (accessed Oct. 10, 2019).

[9] Rana Y. Mohamed, "Vivado High-Level Synthesis." xilinx.com/. https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html#:~:text=Vivado%C2%AE%20High%2DLevel%20Synthesis,need%20to%20manually%20create%20RTL (accessed Aug. 10, 2019).

[10] Sina Ghaffari, Saeed Sharifian, "FPGA-based convolutional neural network accelerator design using high level synthesize," 2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS), pp. 1–6.

[11] S. Hareth, H. Mostafa, and K. A. Shehata, “Low power CNN hardware FPGA implementation”, IEEE International Conference on Microelectronics (ICM 2019), Cairo, Egypt, pp. 162-165, 2019.

[12] Cl´ement Farabet, Cyril Poulet, Jefferson Y. Han, Yann LeCun, "CNP: AN FPGA-BASED PROCESSOR FOR CONVOLUTIONAL NETWORKS." FPL 09: 19th Inter- national Conference on Field Programmable Logic and Applications, 1:32 37, 2009.

[13] E. Adel, R. Magdy, S. Mohamed, M. Mamdouh, and H. Mostafa, “Accelerating Deep Neural Networks Using FPGA “, IEEE International Conference on Microelectronics (ICM 2018), Sousse, Tunisia, pp. 180-183, 2018.

[14] Magnus Halvorsen, "Hardware Acceleration of Convolutional Neural Networks." M.S. thesis, Dept. Computer and Information Science Univ. Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2015. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2353511/13656_FULLTEXT.pdf?sequence=1&isAllowed=y.

[15] R. Osama and Hassan Mostafa, ” Implementation of Deep Neural Networks on FPGA-CPU platform Using Xilinx SDSOC, ” Springer Analog Integrated Circuits and Signal Processing, In Press-Whitepaper-FPGA-Accelerated-CNN-003TR.pdf.

[16] Song Han, HuiziMao, and William J Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." The International Conference on Learning Representations (ICLR), arXiv preprint arXiv:1510.00149v5, 2016

[17] Chen Zhang, Yijin Guan, Peng Li, Bingjun Xiao, Guangyu Sun, and Jason Cong. "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks." Proceedings of the 2015 Association for Computing Machinery/Special Interest Group on Design Automation (ACM/SIGDA) International Symposium on Field-Programmable Gate Arrays, pp. 161-170

[18] M. Elgammal, O. A. Elkhouly, H. Elhosary, M. E. Sayed, A. Mohieldin, K. N. Salama, and H. Mostafa, “Linear and Nonlinear Feature Extraction for Neural Seizure Detection”, IEEE International Midwest Symposium on Circuits and Systems (MWSCAS 2018), Windsor, Ontario, Canada, pp. 795-798, 2018.