Ciencia e Ingeniería Neogranadina 2019 Vol. 30(1) enero - julio 2020 ■ ISSN: 0124-8170 · ISSN-e: 1909-7735 ■ pp. 107-116

Editorial Neogranadina

DOI: https://doi.org/10.18359/rcin.4194

A Hardware Accelerator for The Inference of a Convolutional Neural Network

Edwin González[a] ■ Walter D. Villamizar Luna[b] ■ Carlos Augusto Fajardo Ariza[c]

Abstract: Convolutional Neural Networks (CNNs) are becoming increasingly popular in deep learning applications, e.g. image classification, speech recognition, medicine, to name a few. However, CNN inference is computationally intensive and demands a large number of memory resources. This work proposes a CNN inference hardware accelerator, which was implemented in a co-processing scheme. The aim is to reduce hardware resources and achieve the best possible throughput. The design is implemented in the Digilent Arty Z7-20 development board, which is based on the Xilinx Zynq-7000 System on Chip (SoC). Our implementation achieved a of accuracy for the MNIST database using only a 12-bits fixed-point format. Results show that the co-processing scheme operating at a conservative speed of 100 MHz can identify around 441 images per second, which is about 17 % times faster than a 650 MHz - software implementation. It is difficult to compare our results against other Field-Programmable Gate Array (FPGA)-based implementations because they are not exactly like ours. However, some comparisons, regarding logical resources used and accuracy, suggest that our work could be better than previous ones. Besides, the proposed scheme is compared with a hardware implementation in terms of power consumption and throughput.

Keywords: CNN; FPGA; hardware accelerator; MNIST; Zynq

Received: July 12th, 2019 **Accepted:** October 31st, 2019

Available online: July 15th, 2020

How to cite: E. González, W. D. Villamizar Luna, and C. A. Fajardo Ariza, "A Hardware Accelerator for the Inference of a Convolutional Neural network", *Cien.Ing.Neogranadina*, vol. 30, no. 1, pp. 107-116, Nov. 2019.

a Universidad Industrial de Santander. E-mail: edwin.gonzalez4@correo.uis.edu.co
ORCID: https://orcid.org/0000-0003-2217-9817

b Universidad Industrial de Santander. E-mail: walter.villamizar@correo.uis.edu.co
ORCID: https://orcid.org/0000-0003-4341-8020

c Universidad Industrial de Santander. E-mail: cafajar@uis.edu.co
ORCID: https://orcid.org/0000-0002-8995-4585

Acelerador en hardware para la inferencia de una red neuronal convolucional

Resumen: las redes neuronales convolucionales cada vez son más populares en aplicaciones de aprendizaje profundo, como por ejemplo en clasificación de imágenes, reconocimiento de voz, medicina, entre otras. Sin embargo, estas redes son computacionalmente costosas y requieren altos recursos de memoria. En este trabajo se propone un acelerador en hardware para el proceso de inferencia de la red Lenet-5, un esquema de coprocesamiento hardware/software. El objetivo de la implementación es reducir el uso de recursos de hardware y obtener el mejor rendimiento computacional posible durante el proceso de inferencia. El diseño fue implementado en la tarjeta de desarrollo Digilent Arty Z7-20, la cual está basada en el System on Chip (SoC) Zynq-7000 de Xilinx. Nuestra implementación logró una precisión del 97,59 % para la base de datos MNIST utilizando tan solo 12 bits en el formato de punto fijo. Los resultados muestran que el esquema de co-procesamiento, el cual opera a una velocidad de 100 MHz, puede identificar aproximadamente 441 imágenes por segundo, que equivale aproximadamente a un 17 % más rápido que una implementación de software a 650 MHz. Es difícil comparar nuestra implementación con otras implementaciones similares, porque las implementaciones encontradas en la literatura no son exactamente como la que realizó en este trabajo. Sin embargo, algunas comparaciones, en relación con el uso de recursos lógicos y la precisión, sugieren que nuestro trabajo supera a trabajos previos.

Palabras clave: CNN; FPGA; acelerador en hardware; MNIST, Zynq

1 Introduction

Image classification has been widely used in many areas of the industry. Convolutional neural networks (CNNs) have achieved high accuracy and robustness for image classification (e.g. Lenet-5 [1], GoogLenet [2]). CNNs have numerous potential applications in object recognition [3, 4], face detection [5], and medical applications [6], among others. In contrast, modern models of CNNs demand a high computational cost and large memory resources. This high demand is due to the multiple arithmetic operations to be solved and the huge amount of parameters to be saved. Thus, large computing systems and high energy dissipation are required.

Therefore, several hardware accelerators have been proposed in recent years to achieve low power usage and high performance [7-10]. In [11] an FPGA-based accelerator is proposed, which is implemented in Xilinx Zynq-7020 FPGA. They implement three different architectures: 32-bit floating, 20-bits fixed point format and a binarized (one bit) version, which achieved an accuracy of 96.33 %, 94.67 %, and 88 %, respectively. In [12] an automated design methodology was proposed to perform a different subset of CNN convolutional layers into multiple processors by partitioning available FPGA resources. Results achieved a 3.8x, 2.2x and 2.0x higher throughput than previous works in AlexNet, SqueezeNet, GoogLenet, respectively. In [13] a CNN for a low-power embedded system was proposed. Results achieved 2x energy efficiency compared with GPU implementation. In [14] some methods are proposed to optimize CNNs regarding energy efficiency and high throughput. In this work, an FPGA-based CNN for Lenet-5 was implemented on the Zynq-7000 platform. It achieved a 37 % higher throughput and 93.7 % less energy dissipation than GPU implementation, and the same error rate of 0.99 % in software implementation. In [15] a six-layer accelerator is implemented for MNIST digit recognition, which uses 25 bits and achieves an accuracy of 98.62 %. In [16] a 5-layer accelerator is implemented for MNIST digit recognition, which uses 11 bits and achieves an accuracy of 96.8 %.

It is important to note that most of the works mentioned above have used High-Level Synthesis (HLS) software. This tool allows creating a software accelerator directly, without the need to manually create a Register Transfer Level (RTL). However, HLS software generally causes higher hardware resource utilization, which explains why our implementation required less logical resources than previous works.

In this work, we implemented an RTL architecture for Lenet-5 inference, which was described by using directly Hardware Description Language (HDL) (eg. Verilog). It aims to achieve low hardware resource utilization and high throughput. We have designed a Software/Hardware (SW/HW) co-processing to reduce hardware resources. The work established the number of bits in a fixed-point representation that achieves the best ratio between accuracy/number of bits. The implementation was done using 12-bits fixed-point on the Zynq platform. Our results show that there is not a significant decrease in accuracy besides low hardware resource usage.

This paper is organized as follows: Section 2 describes CNNs; Section 3 explains the proposed scheme; Section 4 presents details of the implementation and the results of our work; Section 5 discusses results and some ideas for further research; And finally, Section 6 concludes this paper.

2 Convolutional neural networks

CNNs allow the extraction of features from input data to classify them into a set of pre-established categories. To classify data CNNs should be trained. The training process aims at fitting the parameters to classify data with the desired accuracy. During the training process, many input data are presented to the CNN with the respective labels. Then a gradient-based learning algorithm is executed to minimize a loss function by updating CNN parameters (weights and biases) [1]. The loss function evaluates the inconsistency between the predicted label and the current label.

A CNN consists of a series of layers that run sequentially. The output of a specific layer is the input of the subsequent layer. The CNN typically

uses three types of layers: convolutional, sub-sampling and fully connected layers. Convolutional layers extract features from the input image. Sub-sampling layers reduce both spatial size and computational complexity. Finally, fully connected layers classify data.

2.1 Lenet-5 architecture

The Lenet-5 architecture [2] includes seven layers (Figure 1): three convolutional layers (C1, C3, and C5), two sub-sampling layers, (S2 and S4) and two fully connected layers (F6 and OUTPUT).

In our implementation, the size of the input image is 28×28 pixels. Table 1 shows the dimensions of inputs, feature maps, weights, and biases for each layer.

Convolutional layers consist of kernels (matrix of weights), biases and an activation function (eg. Rectifier Linear Unit (ReLu), Sigmoid). The convolutional layer takes the feature maps in the previous layer with depth and convolves them with kernels of the same depth. Then, the bias is added to the convolution output and this result passes through an activation function, in this case, ReLu [17]. Kernels shift into the input with a stride of one to obtain an output feature map. Convolutional layers have feature maps of n-k+1×n-k+1 pixels.

Note that weights and bias parameters are not found in the S2 and S4 layers. The output of a sub-sampling layer is obtained by taking the maximum value from a batch of the feature map in the preceding layer. A fully connected layer has feature maps. Each feature map results from the dot product between the input vector of the layer and the weight vector. The input and weight vectors have elements.

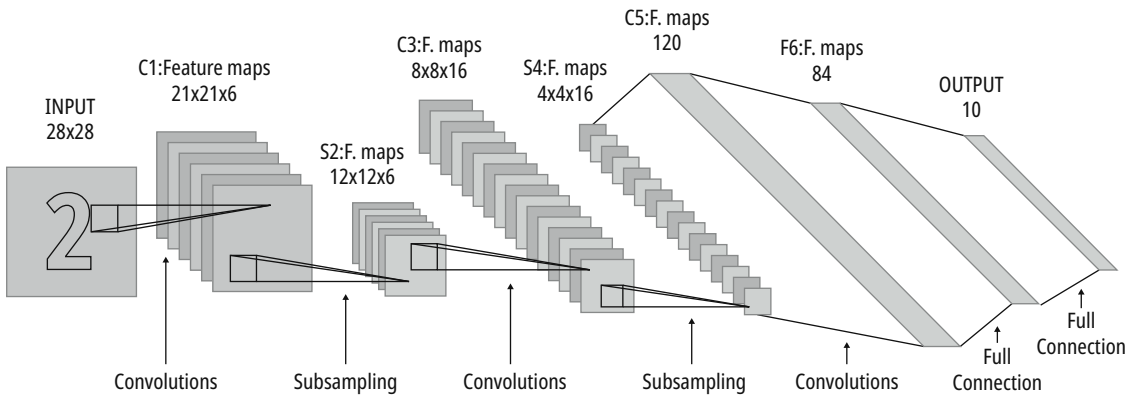

Fig. 1. Lenet-5 architecture.
Source: Adapted from [1].

Table 1. Dimensions of CNN layers

Layer	Size			
	Input n×n×p	Weight k×k×p×d	Bias d	Feature Map m×m×d
C1	28×28×1	5×5×1×6	6	24×24×6
S2	24×24×6	N/A	N/A	12×12×6
C3	12×12×6	5×5×6×16	16	8×8×16
S4	8×8×16	N/A	NA	4×4×16
C5	4×4×16	4×4×16×120	120	1×1×120
F6	1×1×120	1×1×120×84	84	1×1×84
Output	1×1×84	1×1×84×10	10	1×1×10

Source: The authors.

3 Software/hardware co-processing scheme

Fig. 2 presents the proposed SW/HW co-processing scheme to perform the Lenet-5 inference on Zynq-7000 SoC. It consists of two main parts: an Advanced RISC Machine (ARM) processor and an FPGA, which are connected by the Advanced eXtensible Interface 4 (AXI4) bus [18]. A C application runs into the ARM processor, which is responsible for the data transfer between software and hardware. Furthermore, a hardware accelerator is implemented on the FPGA. This accelerator consists of a custom computational architecture that performs the CNN inference process.

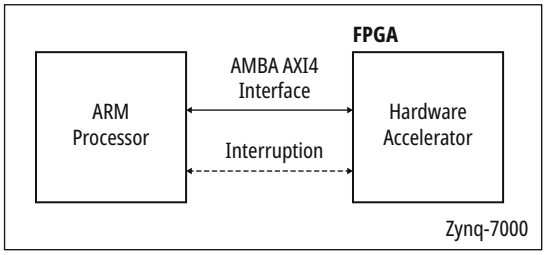

Fig. 2. General software/hardware co-processing scheme.

Source: The authors.

3.1 Hardware accelerator

Fig. 3 shows the hardware accelerator that develops the inference process of CNN. The FSM controls the hardware resources into the accelerator. REGISTER_BANK contains the dimensions of the entire architecture (Table 1). MEMORY_SYSTEM consists of four Block RAMs (BRAMs). The first one stores weights, and the second one stores biases. RAM_3 and RAM_4 store the intermediate values of the process. These memories are overwritten in each layer. All four memories are addressed by the ADDRESS_RAM module.

MATH_ENGINE is the mainstream module in our design because it performs all the necessary arithmetic operations in the three types of layers mentioned in Section 2. All feature maps of the layers are calculated by reusing this module. It is necessary to highlight the saving of hardware resources with the implementation of this module.

This module is used to:

- Perform a convolution process in a parallel fashion
- Calculate a dot product between vectors
- Add bias
- Evaluate the ReLu activation function
- Perform the sub-sampling process

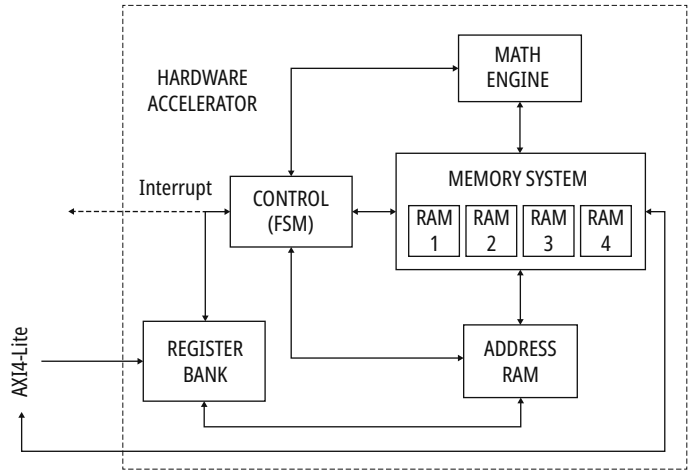

Fig. 3. Hardware accelerator scheme.

Source: The authors.

To perform all the processes mentioned above, the module MATH_ENGINE is composed of CONV_DOT, BIAS_&_ReLu, and SUB_SAMPLING. The overall architecture of the MATH_ENGINE is shown in Fig. 4.

CONV_DOT consists of six blocks. Each block performs either a 5×5 convolution or a dot product between vectors with a length of 25. The BIAS_&_ReLu submodule adds a bias to the output of the CONV_DOT submodule and performs the ReLU function. The SUB_SAMPLING submodule performs a max-pooling process (layers S2 and S4).

3.2 Description of the SW/HW scheme process

Before the accelerator classifies an image, it is necessary to store all the required parameters (weights, biases, and dimensions of CNN layers). The processor sends the input image and the parameters to the MEMORY_SYSTEM and the BANK_REGISTER, respectively. Initially, RAM3 stores the image. Then, the processor sets the start signal of FSM, which initiates the inference process. The FSM configures MATH_ENGINE, ADDRESS_RAM, and MEMORY_SYSTEM according to the data in the BANK_REGISTER.

As mentioned, for convolutional and fully connected layers, the CONV_DOT submodule performs the convolution and dot product operations, respectively. The Bias_&_ReLu submodule adds the bias and performs the ReLu activation function. Moreover, for subsampling layers, the input is passed through the SUB_SAMPLING submodule. According to the current layer, the MATH_ENGINE output could be either the result of Bias_&_ReLu or SUB_SAMPLING.

To reduce the amount of memory required, we implemented a re-use memory strategy. The strategy is based on the use of just two memories, which are switched to store the input and output of each layer. For example, in Layer_i input is taken from RAM3 and output is stored into RAM4. Then, in Layer_(i+1) input is taken from RAM4 and output is stored into RAM3 and so on.

Lenet-5 has ten outputs, one for each digit. Image classification is the digit represented by the output that has the maximum value. The inference process is carried out by executing all the layers. Once this process is finished, the FSM sends an interruption to the ARM processor, which enables the transfer of the result from the hardware accelerator to the processor.

4 Results

This section describes the performance of the SW/HW co-processing scheme implementation. The implementation was carried out into a Digilent Arty Z7-20 development board using two complement fixed-points. This board is designed around the Zynq-7000 SoC with 512 MB DDR3 memory. The SoC integrates a dual-core ARM Cortex-A9 processor with Xilinx Artix-7 FPGA. The ARM processor runs at 650 MHz and the FPGA is clocked at 100 MHz. The project was synthesized using Xilinx Vivado Design Suite 2018.4 software.

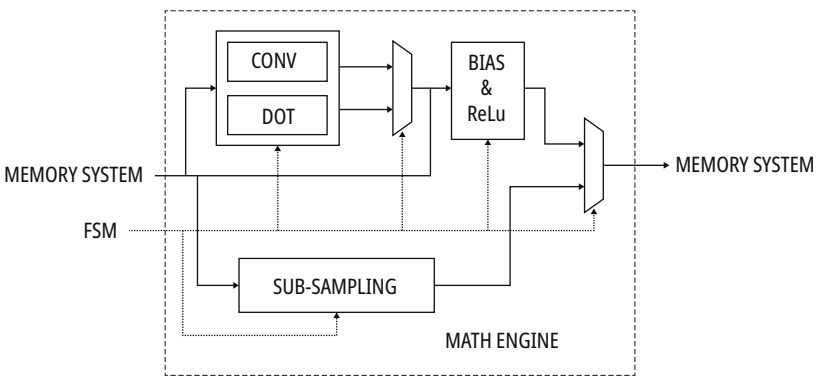

Fig. 4. Math engine.
Source: The authors.

4.1 CNN training and validation

CNN was trained and validated on a set of handwritten digits images from the MNIST database [1]. The MNIST database contains handwritten digits from 0 to 9. The digits are centered on a 28 x 28-pixel grayscale image. Each pixel is represented by 8 bits, obtaining values in a range from 0 to 255, where 0 means white background, and 255 means black foreground. The MNIST database has a training set of 60,000 images and a validation set of 10m000 images. API Keras with TensorFlow backend was used to train and validate the CNN in python. Python was set up to use a floating-point representation to achieve high precision.

Furthermore, the inference process was iterated eight times on the proposed scheme by changing word length and fractional length. CNN parameters and the validation set were quantized in second's complement fixed-point format using MATLAB. Table 2 shows the percent of accuracy obtained by each validation.

Table 2. Accuracy for different data representations

	Word Length [bits]	Integer, Fraction	Accuracy %
Floating Point	32	N/A	98.85
Fixed Point	17	«7,10»	98.85
	15	6,9	98.70
	16	8,8	98.70
	15	7,8	98.70
	14	6,8	98.68
	12	6,6	97.59
	11	6,5	91.46
	16	5,11	61.31

Source: The authors.

The W, F notation indicates integer length (W) and fractional length (F). A starting point was set by finding the number of bits that represents the maximum value in the integer part.

4.2 Execution time

The number of clock cycles spent in data transfer between software and hardware is counted by a global timer. A counter was implemented on the FPGA, which counts the number of clock cycles required by the hardware accelerator to perform the inference process. Table 3 shows the execution time of the implementation.

Table 3. Execution times per image

Process	Time [ms]
Load parameters	7.559*
Load image input	0.122
Inference process	2.143
Extract output data	0.002
Total Time	2.268

Source: The authors.

The parameters are sent to the hardware only once when the accelerator is configured by the processor. Therefore, the time to upload the parameters* was not considered in the total execution time.

The execution time per image is 2,268 ms. Thus, our implementation achieves a throughput of 440,917 images per second.

4.3 Hardware resource utilization

Table 4 shows hardware resource utilization for different word lengths. As the MATH ENGINE performs 150 multiplications in parallel, the amount of Digital Signal Processors (DSPs) used in all cases is constant. In this case, 150 DSPs in the MATH ENGINE and 3 DSPs are used in the rest of the design.

Note that a shorter word length reduces hardware resources. However, it is important to consider how accuracy will be affected by the integer and fractional lengths (Table 2).

Table 4. Comparison of hardware resource utilization for different word lengths

Precision	Word Length [bits]	Hardware resources			Max. accuracy %
		LUTs	FFs	BRAM [KB]	
Fixed point	17	4738	2922	173.25	98.85
	16	4634	2892	164.25	98.70
	15	4549	2862	155.25	98.70
	14	4443	2832	146.25	98.68
	12	4254	2772	119.25	97.59
	11	4151	2742	114.75	91.46

Source: The authors.

4.4 Power estimation

The power estimation for the proposed scheme was made by using Xilinx Vivado software (Fig. 5). This estimation only reports the consumption of Zynq-7000 SoC (a DDR3 RAM is considered part of the ARM processor).

The FPGA consumes power in the software implementation due to the architecture of the Zynq-7000 platform. Note that the ARM processor consumes most of the total power even when the hardware accelerator performs the inference process, which takes most of the execution time.

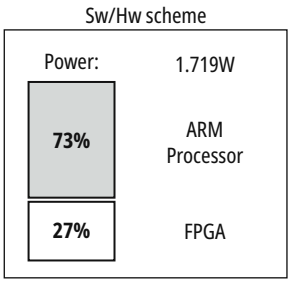

Fig. 5. Power estimation for the proposed scheme.
Source: The authors.

5 Discussion

We compared the implementation of our co-processing scheme with two different implementations on the Zynq-7000 platform: software-only and hardware-only solutions. In the software-only solution, the input image and CNN parameters are taken to the DDR3 RAM. The hardware-only solution uses a serial communication (Universal Asynchronous Receiver-Transmitter (UART)) module to replace the ARM processor in the co-processing scheme.

Table 5 shows the results of the three implementations on the Arty Z7 board. Although the proposed scheme implementation uses less than twice the word length of the software-only solution, the accuracy only fell by 1.27 %. Also, our co-processing scheme achieved the highest throughput. The hardware-only implementation is a low power version of the proposed scheme. Future research will focus on improving the performance of the hardware-only solution. Note that the hardware-only implementation has the lowest throughput because of the bottleneck imposed by the UART (to transfer the input image, CNN parameters are stored in BRAMs). However, this implementation could be the best option for applications in which power consumption is critical and not throughput.

As mentioned, it is difficult to compare our results against other FPGA-based implementations because they are not exactly like ours. However, some comparisons can be made regarding the use of logical resources and accuracy. Table 6 presents a comparison with some predecessors.

Table 5. Power estimation for different implementations

	Co-processing scheme	Hardware-only	Software-only
Data format	12-bit fixed-point	12-bit fixed-point	32-bit floating-point
Frequency (MHz)	100	100	650
Accuracy (%)	97.59	97.59	98.85
Throughput (images/second)	441	7	365
Power estimated (W)	1.719	0.123	1.403

Source: The authors.

Table 6. Comparison with some predecessors

Metric	Our Design	[11]	[17]	[18]
Model	3 Conv 2 Pooling 2 FC	2 Conv 2 Pooling 1 FC	2 Conv 2 Pooling 2 FC	2 Conv 2 Pooling
Fixed Point	12 bits	20 bits	25 bits	11 bits
Operation Frequency	100 MHz	100 MHz	100 MHz	150 MHz
BRAM	45	3	27	0
DSP48E	158	9	20	83
FF	2772	40534	54075	40140
LUT	4254	38899	14832	80175
Accuracy	97.59 %	94.67%	98.62 %	96.8 %

Source: The authors.

Note that our design uses less Look-Up Tables (LUTs) and Flip-Flops (FFs) than these previous works. Only [15] achieves better accuracy because this implementation uses more bits; however, this number of bits increases the number of logical resources.

6 Conclusion

In this paper, an SW/HW co-processing scheme for Lenet-5 inference was proposed and implemented on a Digilent Arty Z7-20 board. Results show an accuracy of 97.59 % using a 12-bit fixed-point format. The implementation of the proposed scheme achieved a higher throughput than a software implementation on the Zynq-7000 platform. Our results suggest that the usage of a fixed-point data format allows the reduction of hardware resources without compromising accuracy. Furthermore, the co-processing scheme makes it possible to improve the inference processing time. This encourages future advances in energy efficiency on embedded devices for deep learning applications.

References

[1] Y. LeCun et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998. https://doi.org/10.1109/5.726791

[2] C. Szegedy et al., "Going deeper with convolutions," in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015. pp. 1-9. https://doi.org/10.1109/CVPR.2015.7298594

[3] A. Dundar, J. Jin, B. Martini, and E. Culurciello, "Embedded streaming deep neural networks accelerator with applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 7, pp. 1572-1583, July 2017. https://doi.org/10.1109/TNNLS.2016.2545298

[4] B. Ahn, "Real-time video object recognition using convolutional neural network," in 2015 International Joint

Conference on Neural Networks (IJCNN), July 2015, pp. 1-7. https://doi.org/10.1109/IJCNN.2015.7280718

[5] B. Yu, Y. Tsao, S. Yang, Y. Chen, and S. Chien, "Architecture design of convolutional neural networks for face detection on an fpga platform," in 2018 IEEE International Workshop on Signal Processing Systems (SiPS), Oct. 2018, pp. 88-93. https://doi.org/10.1109/SiPS.2018.8598428

[6] Z. Xiong, M. K. Stiles, and J. Zhao, "Robust ecg signal classification for detection of atrial fibrillation using a novel neural network," in 2017 Computing in Cardiology (CinC), Sep. 2017, pp. 1-4. https://doi.org/10.22489/CinC.2017.066-138

[7] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35-47, Jan 2018. https://doi.org/10.1109/TCAD.2017.2705069

[8] N. Suda et al., "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 16-25. http://doi.acm.org/10.1145/2847263.2847276

[9] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 161-170. https://doi.org/10.1145/2684746.2689060

[10] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper*, vol. 2, no. 11, pp. 1-4, 2015.

[11] T. Tsai, Y. Ho, and M. Sheu, "Implementation of fpga-based accelerator for deep neural networks," in 2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), April 2019, pp. 1-4. https://doi.org/10.1109/DDECS.2019.8724665

[12] Y. Shen, M. Ferdman, and P. Milder, "Maximizing cnn accelerator efficiency through resource partitioning," in 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), June 2017, pp. 535-547. https://doi.org/10.1145/3140659.3080221

[13] Y. Wang et al., "Low power convolutional neural networks on a chip," in 2016 IEEE International Symposium on Circuits and Systems (ISCAS), May 2016pp. 129-132,. https://doi.org/10.1109/ISCAS.2016.7527187

[14] G. Feng, Z. Hu, S. Chen, and F. Wu, "Energy-efficient and high-throughput fpga-based accelerator for convolutional neural networks," in 2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), Oct. 2016, pp. 624-626. https://doi.org/10.1109/ICSICT.2016.7998996

[15] S. Ghaffari and S. Sharifian, "FPGA-based convolutional neural network accelerator design using high level synthesize," in Proceedings - 2016 2nd International Conference of Signal Processing and Intelligent Systems, ICSPIS 2016, 2016, pp. 1-6. https://doi.org/10.1109/ICSPIS.2016.7869873

[16] Y. Zhou and J. Jiang, "An FPGA-based accelerator implementation for deep convolutional neural networks," in Proceedings of 2015 4th International Conference on Computer Science and Network Technology, ICCSNT 2015, 2015, vol. 01, no. Iccsnt, pp. 829-832. https://doi.org/10.1109/ICCSNT.2015.7490869

[17] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzman machines," in Proceedings of the 27th International Conference on International Conference on Machine Learning, ser. ICML'10. USA: Omnipress, 2010, pp. 807-814. [Online]. http://dl.acm.org/citation.cfm?id=3104322.3104425

[18] Xilinx. *Axi reference guide*. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf