A Scalable and Adaptive Convolutional Neural Network Accelerator

Jan Pidanic
Department of Electrical Engineering
University of Pardubice
Pardubice, Czech Republic
jan.pidanic@upce.cz

Arpan Vyas
Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Guwahati, India
v.arpan@iitg.ernet.in

Rishav Karki
Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Guwahati, India
k.rishav@iitg.ernet.in

Prateek Vij
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati, India
v.prateek@iitg.ernet.in

Gaurav Trivedi
Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Guwahati, India
trivedi@iitg.ac.in

Zdenek Nemec
Department of Electrical Engineering
University of Pardubice
Pardubice, Czech Republic
zdenek.nemec@upce.cz

***Abstract*—Machine learning has become ubiquitous and penetrated every field of technology, medicine, and finance. Convolutional Neural Network (CNN) is one of the most commonly used class of machine learning algorithms that is being used in video and image processing, big data processing, natural language processing, robotics, and a variety of pattern matching and recognition tasks. Depending on the end application, CNNs are being employed on different scales ranging from tiny motion sensors and smartphones to automobiles and server farms.**

Although existing CNN accelerators are adaptive for different types of CNN models, they are generally suited for a particular scale of operation. In this paper, we describe a scalable and adaptive CNN accelerator. The same hardware-cum-software stack can be configured by a system-level parameter to be synthesized for different scales of operation. This makes the accelerator highly portable across systems of different scales. Furthermore, one single synthesized hardware can run inference for multiple CNN models because of the flexible software stack and hardware control unit making the system highly adaptive. We demonstrate the working of the system at different scales by implementing it on the Xilinx Virtex 7 FPGA and by running multiple CNN models at each scale.

***Keywords*—Convolutional Neural Networks, Hardware Accelerator, Scalable, Adaptive, FPGA**

I. Introduction

Numerous Artificial Intelligence (AI) applications have emerged in various sectors like automotive, healthcare, robotics, and personal electronics in the past few years. Machine Learning algorithms are increasingly replacing the traditional algorithms with hard-coded features designed by domain experts across various sectors. Among Machine Learning algorithms, Convolutional Neural Networks (CNN), which are brain-inspired algorithms, currently represent one of the most promising approaches for computer vision tasks. CNNs are applied in various domains like image understanding [1], speech recognition [2], and robotics [3]. In many of these domains, state-of-the-art CNNs have achieved far greater accuracy in performing various computer vision tasks than traditional algorithms [4]. However, this superior accuracy also incurs much computational complexity. With the advent of the Internet of Things (IoT), computing at the edge has also become necessary for many applications where security, low latency, and low power consumption are essential requirements. General Purpose compute engines, such as Graphics Processing Units (GPUs) consume a lot of power and hence are not suitable for edge applications. As a result, there is an interest in developing energy-efficient, specialized accelerators for CNNs.

Due to their reconfigurability and power efficiency, FPGAs are a viable hardware platform for accelerating compute-intensive operations used in CNNs. They consist of many configurable logic blocks, and programmable interconnects, enabling customizable accelerator architectures in hardware. This paper proposes a scalable and adaptive CNN inference accelerator, customizable for both server and edge computing, that can be quickly deployed for various workloads. The accelerator has been emulated on a Xilinx Virtex 7 FPGA.

The rest of the paper is organized as follows: Section II provides a brief review of existing neural network accelerators. In section III, we explore this paper's background, where we describe the central concepts in this work - Deep Neural

Networks and CNNs. Section IV describes the system architecture of our proposed implementation. Section V describes our experimental setup involving emulation of the design on the Xilinx Virtex 7 FPGA. Section VI presents the results and observations. Finally, Section VII concludes the paper.

II. Related Work

Due to the high computational complexity of CNNs, typically running into multiple Giga Operations Per Second (GOPS), traditional CPUs are not the best candidates for implementation. GPUs with a high degree of parallelism and specialized libraries like OpenCL are widely used in both training and classification. Specialized hardware built using FPGAs and ASICs are able to improve speed and power efficiency by roughly an order of magnitude as compared to GPUs. A lot of specialized accelerators using FPGAs and ASICs have come up in recent years.

Among the FPGA and ASIC based designs, one class of accelerators are adaptive and can run multiple CNN models on the same hardware, whereas the other class of accelerators are highly customized for a specific CNN model to provide improved inference speed. Eyeriss [5] based on ASIC optimizes for energy efficiency of the system by reducing overall data movement. This can be achieved via. data reuse and exploiting data statistics by using model compression and skipping insignificant computations. EyerissV2 [6] is a transformed model to handle compact and sparse networks making it suitable for edge devices. Angel-Eye [7] provides a design flow which can map an adaptive CNN accelerator onto embedded FPGAs and leverages its software backend to run multiple CNN models. ALAMO [8] is an RTL compiler which generates a customized hardware accelerator for a specified CNN model. HACO [9], tailored for VGG16, co-optimizes the hardware and algorithm by employing selective layer pruning which is state-of-the-art in terms of inference time. Table I compares these works with our implementation.

III. Background

A. Deep Neural Networks

Artificial Neural networks are machine learning models that are inspired by the biological network of neurons in our brain. Analogous to a biological neuron, an artificial neuron is the basic building block of artificial neural networks. A Deep Neural Network (DNN) consists of multiple hidden layers, which consist of multiple artificial neurons. Each neuron receives multiple input signals x_i, from other neurons, and each input signal x_i is multiplied by weights w_i. These weighted input responses are summed up, along with a fixed bias term w_b. This operation's output is then fed to an activation function, such as sigmoid, tanh, or ReLu, which introduces non-linearity in the network [10].

Finally, a loss function L is computed, which measures the gap between desired outputs and the neural network outputs. The weights w_i and w_b are parameters that can be updated during the training process, aiming to minimize the loss function. This training process requires large data sets and significant computational resources, and hence, it is usually performed in the cloud. Once the weights have been learned, inputs can be processed with these weights, and this process is known as inference.

B. Convolutional Neural Networks

Convolutional Neural Networks are a common form of DNNs. Unlike, fully connected networks, in CNNs, only a small neighborhood of input activations, called the receptive field, contributes to the output activation. The weights, stored in a weight matrix, are shared for every output, and this weight matrix, called the filter, is slid across the input activations. This effectively results in a 2-D convolution of the filter with the receptive field generating a single output activation.

A layer's input activations are arranged as a set of channels in a CNN, where each channel is a 2-D input feature map. There are multiple 2-D filters, and each input feature map is convolved with a distinct filter. The 2-D convolution results at each point, are added, along with a bias term to generate the output feature map. Each stack of 2-D filters is collectively referred to as a single 3-D filter. Multiple 3-D filters can be convolved with the input feature maps, and each 3-D convolution generates a distinct output feature map [11].

IV. System Architecture

The system is divided into two parts, a reconfigurable hardware accelerator, Fig. 2, and its software stack, Fig. 1. The software stack parses the trained model and converts it into a hardware readable format in the form of program instructions and memory dumps. The hardware accelerator executes the program instructions and computes the inference output layer by layer. The final output is stored back into the memory at a predefined location.

The hardware is described in Verilog HDL and is parameterized with N_PE. N_PE specifies the number of parallel processing elements, the number of convolvers in each processing element, and also the number of memory buffers in each of the two internal memory banks. Thus, the amount of hardware resources is proportional to the square of N_PE.

A. Software Stack

The software stack generates memory dump and machine instructions for the hardware. It requires a pre-trained model file in the 'h5' format, generated by Tensorflow. The software parses the model file for the network architecture and arrangement of convolutional, pooling, and fully connected layers along with their output activation functions. It, in turn, generates the hardware memory maps for input feature list, model weights, save and load operations, and inference outputs. It further generates the program instruction file which is executed on the hardware.

B. Hardware Blocks

1) Processing Elements and Processing Element Array: A processing element or PE can independently perform layer computations for convolution, pooling, and fully connected

TABLE I: Comparison

Work	Platform	Scalable	Adaptive	Frequency	Data Format	Inference Time on VGG16
Eyeriss	ASIC	–	✓	200MHz	16-bit fixed-point	1429 ms
Angel-Eye	Embedded FPGA	–	✓	150MHz	16-bit floating-point	96.5 ms
ALAMO	FPGA	–	✓	100MHz	8-16bit	–
HACO	FPGA	–	–	150MHz	8-bit	12 ms
This work	FPGA	✓	✓	100MHz	Configurable	302 ms

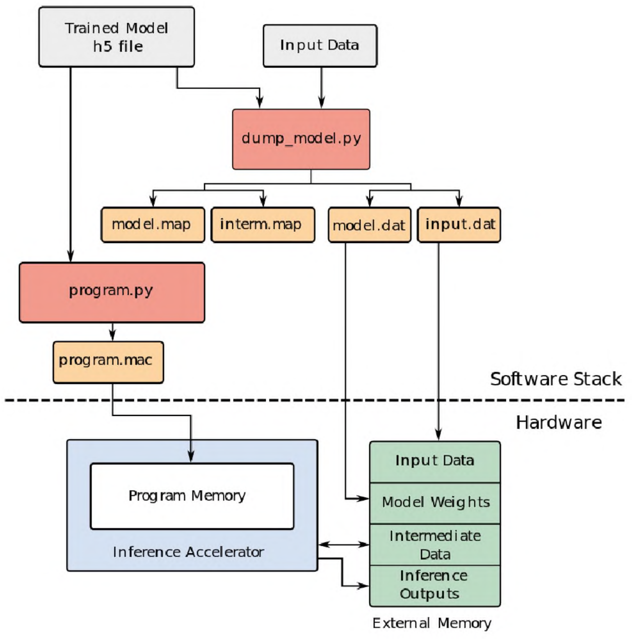

Fig. 1: Software Stack

layers. Each processing element contains N_PE parallel sliding window convolvers, bias adder, feedback adder, and activation compute block. Because convolution is compute-intensive, stacking multiple parallel convolvers in parallel PEs can significantly reduce the convolution time. Further, the same convolvers are used for computation of a fully connected layer. The PE also contains a single pooling module that does the min, max, and average pooling computations.

Processing Element Array contains a stack of N_PE parallel processing elements. This implies that at any point of time $N\_PE * N\_PE$ convolutions or N_PE pooling and dense operations can run simultaneously.

2) Memory Banks: There are two memory banks each with $N\_PE + 1$ parallel dual port RAM buffers. Only the ping mode of operation is supported where the input data and weights are loaded from external memory into the first memory buffer. The Processing Element Array then computes the output and stores it into the second memory bank. The output is saved back into the external memory from the second memory bank. In an enhanced ping-pong mode the intermediate output of one layer, once stored in the second memory bank, can be reused as the input for the next layer thus saving one burst of external memory save and load.

3) Controller: The controller consists of the datapath controller along with register files, SPI Interface, program memory, and program driver. It executes the program instructions and coordinates the data flow between External Memory, Memory Banks, and Processing Element Array. Depending on the type of layer like convolution, pooling, and fully connected, the controller coordinates the data flow to various sub-blocks of Processing Element to and from the memory banks.

V. Experiments

The hardware was implemented on Xilinx Vivado 19.1 for the XC7VX485T device which provides 485,760 logic cells and 2800 DSP slices. The design was synthesised for three different values of the scaling parameter at N_PE values of 4, 8 and 16. Due to interfacing issues with proprietary DDR memories and interfaces, the external memory was mimicked with on chip BRAMs. To run inference for a given CNN model, a trained network model and inputs were loaded into the external memory model and the program file was loaded in the accelerator's program memory. Three CNN models of varying complexity were run to demonstrate the flexibility of the inference system. The first model was a very small 3-layer digit classifier on the MNIST database of 60,000 images. This consists of one 32 channel convolution layer followed by a max-pooling and the final 10-node fully connected layer. With tuned batch size and multiple training iterations, it was able to achieve 97.33% classification accuracy was achieved. This model is suitable for low-end edge applications. The second model was a mid-sized MNIST classifier with a total of 9 layers and a very high accuracy of 99.25%. The third model was VGG-16 proposed by [12] which achieves 92.7% top-5 accuracy on ImageNet, a dataset of 14 million images and 1000 categories. This model has 16 layers and is best suited to be run on large-sized hardware with bigger buffers and multiple parallel compute elements.

VI. Results

A. Inference Timing

Inference time is measured for the above three models for all three configurations of N_PE. In Tables II, III and IV, we specify the total inference time for different values of N_PE. It is also observed that the ratio of a layer's compute time to its total time is higher for convolution layers and lower for dense layers. This happens because convolution is compute-intensive whereas matrix-vector multiplication is memory intensive. For the same reason, the maximum GOPS

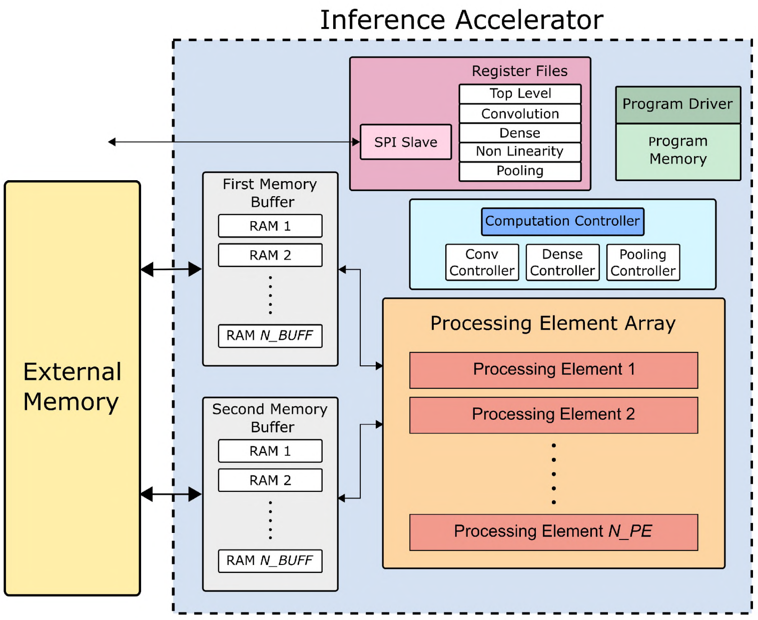

Fig. 2: Hardware Accelerator

TABLE II: VGG-16 Inference Time for different N_PE

Total Time N_PE 4	Total Time N_PE 8	Total Time N_PE 16
1840 ms	701.5 ms	302.2 ms

TABLE III: 9-Layer MNIST Inference Time for different N_PE

Total Time N_PE 4	Total Time N_PE 8	Total Time N_PE 16
1699 us	681 us	421 us

TABLE IV: 3-Layer MNIST Inference Time for different N_PE

Total Time N_PE 4	Total Time N_PE 8	Total Time N_PE 16
319 us	209 us	130 us

TABLE V: Resources Used

Hardware Configuration	LUT	LUTRAM	FF	DSP	BRAM*
N_PE 4, Word Size 16bits	9095	864	8006	380	40
N_PE 8, Word Size 16bits	23228	3264	22705	2800	72
N_PE 16, Word Size 12bits	106142	6336	87648	2800	68
Available	303600	130800	607200	2800	1030

* Can vary depending upon the size of the model targeted

TABLE VI: Peak Giga Operations Per Second

	Convolution	Pooling	Dense
N_PE 4	30	1.4	8
N_PE 8	117.6	3.2	16
N_PE 16	465.6	6.4	32

for convolution computation is much higher than that of Pooling and Dense layer computations. Table VI depicts the theoretical peak GOPS for different layers at varying N_PE values.

B. Hardware Resources Usage

Table V provides the resources used in implementation of the complete hardware for different N_PE and word sizes. For smaller MNIST networks with 4 and 8 N_PEs, this strikes a good balance between resources used and inference time. Larger networks with 16 N_PE is more appropriate, though now, the word size and hence accuracy may be reduced to some extent to keep the resource count low. The BRAMs used for on-chip memory banks store one layer's weights, inputs and outputs during computation. Thus the memory bank size and hence the number of BRAMs can vary depending on the model size targeted for a given implementation.

VII. Conclusion

In this paper, we have proposed an adaptive CNN inference accelerator based on FPGA, that can work with models of any size and complexity. The hardware can easily be scaled to run on devices with sizes varying from large cloud nodes to edge-computing nodes. The scalability and adaptability together make our setup very useful for a wide range of applications. In the future, we plan to increase the scope of our design to run more complex deep learning models with a combination of dense, RNN, and CNN layers. We also plan to investigate the energy requirements for our design and make it more power-efficient and suitable for commercial usage.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105.

[2] A. Graves, A. rahman Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," 2013.

[3] U. Côté Allard, F. Nougarou, C. L. Fall, P. Giguère, C. Gosselin, F. Laviolette, and B. Gosselin, "A convolutional neural network for robotic arm guidance using semg based frequency-features," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2464–2470.

[4] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[5] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[6] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[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, 2018.

[8] Y. Ma, N. Suda, Y. Cao, S. Vrudhula, and J. sun Seo, "Alamo: Fpga acceleration of deep learning algorithms with a modularized rtl compiler," *Integration*, vol. 62, pp. 14–23, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926017304777

[9] T. Yuan, W. Liu, J. Han, and F. Lombardi, "High performance cnn accelerators based on hardware and algorithm co-optimization," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 1, pp. 250–263, 2021.

[10] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating cnn inference on fpgas: A survey," 2018.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, 2012, p. 1097–1105.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.