Design of Computationally Efficient Convolutional Neural Network for Hardware Limited Operation

Jayakrishnan K S
Department of Electronics and Communication
College of Engineering Trivandrum
Thiruvananthapuram-16, India
jayakrishnanksk12@gmail.com

David Solomon George
Department of Electronics and Communication
College of Engineering Trivandrum
Thiruvananthapuram-16, India
david@cet.ac.in

***Abstract*—This paper deals with the design, simulation and synthesis of flexible custom modules that can be used to implement the forward pass arithmetic in a Convolutional Neural Network (CNN). By using different designs explored in this work a trained Convolutional Neural Network can be implemented. The various components essential for the functioning of the trained network are identified and implemented as separate reusable modules, with various versions of same components focusing on different hardware usage. Numerous design logics are used to reduce computation, hardware utilization and complexity without compromising the performance of the Network. One can choose the different designs according to the Convolutional Neural Network architecture, hardware availability and other functional requirements.This paper is implemented on Spartan 6 defence grade low energy board using Xilinx project navigator ISE and Verilog HDL.**

***Index Terms*—Convolutional Neural Network (CNN),Field Programmable Gate Array(FPGA).**

I. INTRODUCTION

The field of Machine learning (ML) has been dedicated to the implementation of operation of intelligent conscience in machines for decades now, and it has made much advance in this effort [1]. The achievement has been a result of careful observation and understanding of other intelligent organism's characteristics of intellectual growth and behavior. As a matter of fact the learning in ML is based on how living organisms learn through experience i.e., encountering a scenario reacting in a specific manner and analyzing the outcome to decide whether the reaction was acceptable or not and according to the analysis accept the reaction as proper procedure to tackle the scenario or retry scenario with different reactions till you find a favorable one. It is from this reasoning that those in ML decided to study the brain, the organ responsible for intelligence [2]. From monitoring the structure, constituents and functioning of the brain intelligence was further understood. It was observed that the network of neurons in the brain were responsible for learning and relearning different things. This process is implemented through different methods of learning in ML like supervised, regression or unsupervised learning. Each method has many types algorithms to tackle different tasks. From the attempt to implement the network of learning neurons in the brain we achieved Artificial neural networks [3] consisting of inter connected nodes, that can be trained to do different tasks. There are different types of neural networks such as deep neural networks, shallow neural networks, convolutional neural network (CNN) etc., these Neural Networks (NN) can be used to solve complex tasks once trained, but training, testing and using such NN can be complex, computationally costly, require lot of dedicated hardware resources and energy consuming. There have been a lot of attempts to solve these issues. Much research and modification have been done in the architectures, algorithms and hardware used for these NN. One such method gleamed from the working of the brain is that the learning and using of the learned information are done by different parts of the brain similarly, we can split the learning process and implementation of learned solutions in machines as well. This is found to be efficient as learning requires more resources, than the implementation of learned solutions. Also not every aspect of the learned solution is required at all times. Hence the relevant solutions are learned on highly resourceful hardware systems and are then transferred to and adaptively used on lower spec hardware in application specific situations as needed. This is of course done at the price of real time learning. But this method is more hardware, energy and cost efficient. This work will explore such a method.

The specific NN explored in this work is the CNN [4], which is a major neural network used in the field of computer vision. CNN are based on the working of the visual neural network of the brain [5]. They have been used widely and research in this field has yielded many types of CNN [6] architecture tailored for different uses such as image classification, object detection, natural language processing. These CNN are complex and highly energy consuming as well as computationally complex design and many work have been done to reduce these issues[7]. The complexity and hardware utilization is most for learning process involved where as the forward pass is very much straight forward and simple and there have been ideas where the learning and application of solution have been achieved on separate dedicated hardware [8][9]. This allows use of resource to be decentralized and used as needed. Here we aim to produce a modular design capable of implementing the forward pass of a CNN. Various functional units are observed and broken into sub units and they are implemented using different hardware logic to achieve

various versions that focus on speed, energy consumption and hardware use respectively. These are to be used according to application, CNN architecture and hardware availability.

II. METHODOLOGY

This paper will explore the design, simulation, synthesis, implementation and improvement of the forward pass hardware or learned solution executing hardware of a Convolutional neural network, on a Spartan 6 FPGA.Xilinx Navigator andVerilog HDL is used to design, simulate and synthesis the hardware modules. The training and validation of various CNN architecture are observed using TensorFlow 2 and python. The layer wise operation of each component is observed. The layer wise implementation of the CNN architecture is achieved by identifying the various major layers that are used with in it, such as Convolutional layer, pooling layer, fully connected layer. The operations in each of these layers are studied and expressed in arithmetic formulae form. They are then implemented in hardware using most minimal hardware use. The complex arithmetic is simplified or approximated or implemented using look up tables. The core repetitive units of each layer are identified and they are implemented as modules using the least amount of hardware, complexity and operational steps. This is possible because NN are made of repeating units of node responsible for the learning and implementation of learned solution. The overall design will be such that by using user-controlled parameters one can control the amount of hardware resources used in each implementation as per device specification. The design will allow easy addition of custom modules, flexibility in module selection as per functional requirement, allowance of pipelining and parallelism of the hardware. Through the implementation and study of CNN in TenserFlow 2 it was found out that the major operations in the forward pass are Tensor Multiplication, Addition, Comparation, and Division, all of this can be implemented using storage and shift Registers, sufficiently sized Accumulators, Mux and some basic gates.

III. RESULTS AND OBSERVATIONS

The most important component of CNN are its convolutional layer in which the neural network tries to recognize learned patterns in the input feature map. The idea of filters convolving over the input FM can be simplified as element wise multiplication of indexed matrices and subsequent summation of these results. This is what happens in each node of a convolutional layer. There are many ways of implementation of the convolutional layer. Here we will explore the case where the filter weights and image pixels are given as inputs along with other synchronization and control signals the output feature map is received as pixels array. The data type of the input and output are binary integers. They are given and received from the modules as array of 16-bit sign data. For real values we use fixed point arithmetic as it reduces the hardware requirements, energy consumption and computational cost. Each module designed has a set of parameters that can be adjusted to control size of the data related to inputs output and sub module counts used. The tensor convolution which is the most important part of this layer can be achieved as follows.

Segment input into filter size tensors then feed it to each node module. This process will require substantially more hardware but is less time consuming. Another way is to read the entire input and implement apply multiple filters to different segments simultaneously which is also hardware exhaustive. The low hardware utilizing method is to identify the basic repeating operation and use the module of it in a synchronized scheduled fashion where the overall time may over shoot. But the time over shoot can be controlled using CNN fastening algorithms and methods such as FFT or depth wise and point wise separable convolution. The fully connected layer can be achieved from a convolutional layer by setting filter size equal to input and setting shift parameter to zero. All the operation in the Convolutional layer was seen to be achieved using accumulation as the key base factor hence using Accumulator, shift registers, multiplexers and logic gate low hardware module was designed, simulated and synthesized. Other relatively higher hardware consuming

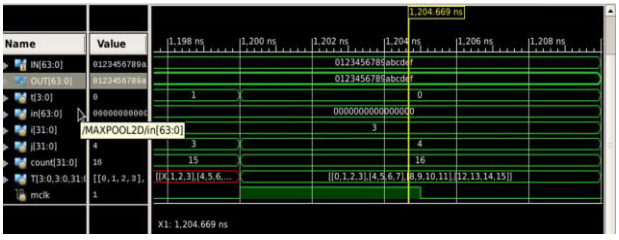


Fig. 1. input segmenting simulation result

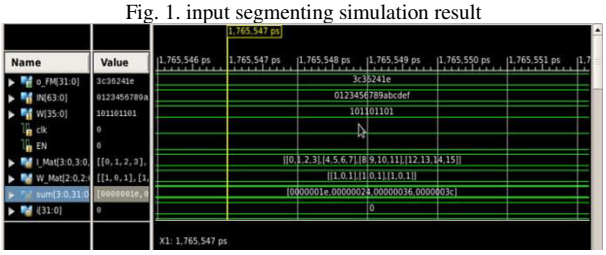
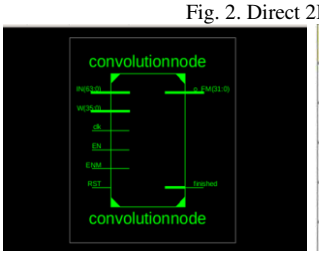


Fig. 2. Direct 2D convolution result



Logic Utilization	Used
Number of Slice Registers	220
Number of Slice LUTs	336
Number of fully used LUT-FF pairs	197
Number of bonded IOBs	136
Number of BUFG/BUFGCTRLs	1

Fig. 3. Direct 2D convolve module schematic top view and hardware use

modules were also designed, simulated and synthesized for various comparative purposes. From the study of various CNN architecture in TensorFlow 2 it was observed that weight approximation to 1/1000 of the decimal place will not affect the performance of the model.The simulation results for input segmenting module is shown in figure 1 where the given input is stored in a memory block T as a 2D tensor. figure

2 shows the simulation result of a 2D convolve module result of direct convolution between two such tensors are shown in an internal memory module and also as a stream output. Figure 3 shows the synthesis result for the above mentioned modules. The pooling layer was also implemented in such a way. The pooling aimed at reducing input data size involves a design where the filter inputs were removed and accumulators where the key operating elements. The simulation result of Maxpool module is shown in figure 4. followed by the synthesis result in figure 5.

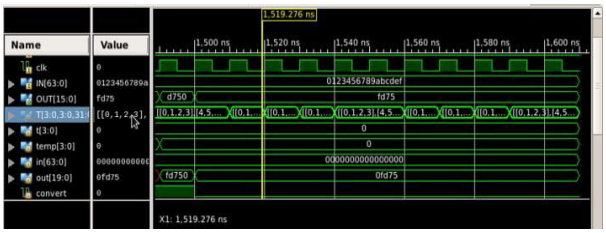
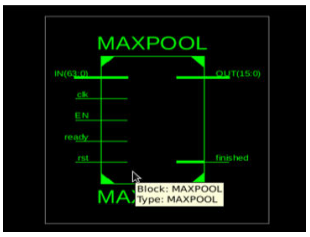


Fig. 4. Direct Maxpool simulation result.



Logic Utilization	Used
Number of Slice Registers	172
Number of Slice LUTs	251
Number of fully used LUT-FF pairs	169
Number of bonded IOBs	85
Number of BUFG/BUFGCTRLs	1

Fig. 5. Direct Maxpool module schematic top view and hardware use

The activation functions used here are Softmax and ReLU for classification cases. Softmax is a computationally demanding module implemented using accumulators or lookup tables in different modules as the need arises.The simulation result of Softmax lookup is shown in figure 6 and synthesis result is shown in figure 7. along with the looup the softmax also use divider module whose synthesis result is shown in figure 8. ReLU is achieved using Multiplexers and the sign bit of the input data. The simulation result of ReLU is shown in figure 9 and synthesis result is shown in figure 10. The input data to the higher-level modules are segmented and fed to sub modules as per submodule count. The results are approximated to fit the pre-set data size with addition of a scaling matrix that keep track of the quantitative size of approximated result in case of over flow. Values of low scale are approximated to null and null result operations are ignored using logic control. Thus reducing computation cost. The element wise matrix multiplication can be achieve using Verilog default multiplier, or custom multiplier consisting of a shift register and accumulator. the simulation result of both modules are shown in figure 11 and synthesis result is shown in figure 12.13. The indexed element accessing is achieved using counters also implemented with shift register and accumulation. Comparators used for flow control can also be implemented using accumulators and logic gates. The figure shows the result. The schematics, simulation result and hardware usage of accumulator are show

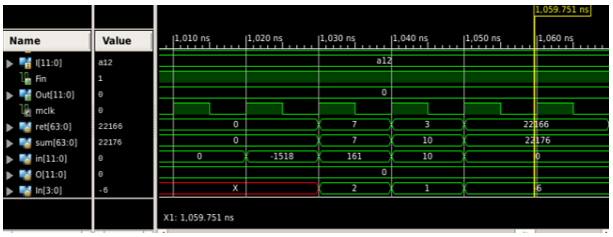
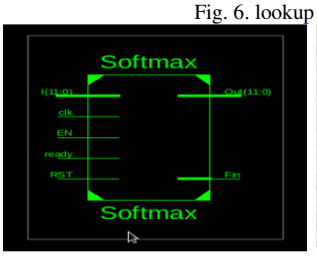
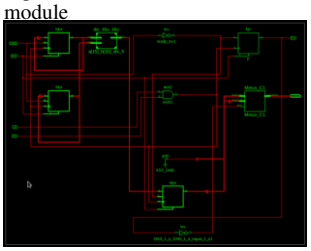


Fig. 6. lookup simulation result.



Logic Utilization	Used
Number of Slice Registers	99
Number of Slice LUTs	177
Number of fully used LUT-FF pairs	22
Number of bonded IOBs	29
Number of BUFG/BUFGCTRLs	1

Fig. 7. Softmax module schematic top view and hardware use of lookup module



Logic Utilization	Used
Number of Slice Registers	74
Number of Slice LUTs	443
Number of fully used LUT-FF pairs	13
Number of bonded IOBs	69
Number of BUFG/BUFGCTRLs	1

Fig. 8. Divider module schematic and hardware use

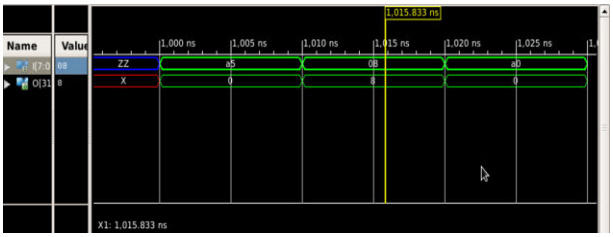
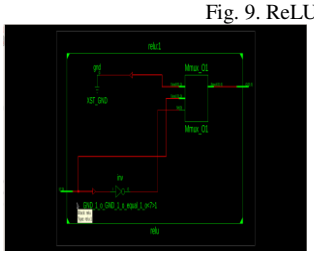


Fig. 9. ReLU simulation result.



Logic Utilization	Used
Number of Slice LUTs	7
Number of fully used LUT-FF pairs	0
Number of bonded IOBs	40

Fig. 10. ReLU module schematic top view and hardware use

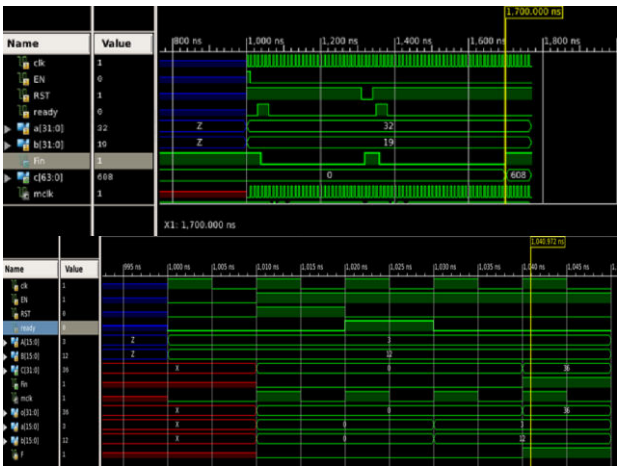
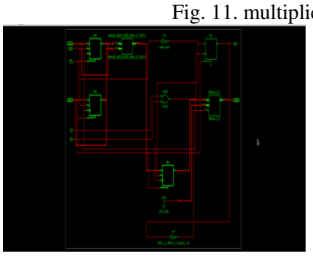
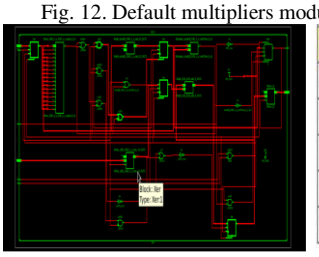


Fig. 11. multipliers simulation result.



Logic Utilization	Used
Number of Slice Registers	1
Number of Slice LUTs	34
Number of fully used LUT-FF pairs	1
Number of bonded IOBs	69
Number of DSP48A1s	1

Fig. 12. Default multipliers module schematic view and hardware use



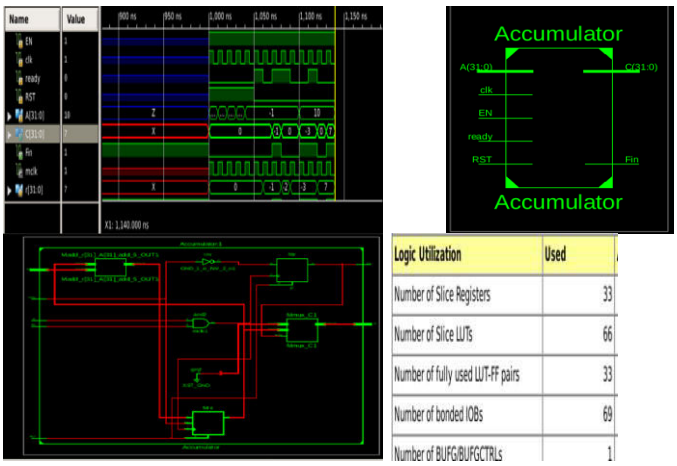
Logic Utilization	Used
Number of Slice Registers	85
Number of Slice LUTs	145
Number of fully used LUT-FF pairs	79
Number of bonded IOBs	69
Number of BUFG/BUFGCTRLs	1

Fig. 13. Custom multiplier module schematic and hardware use

in figure 14 . As one can see different modules have different hardware consumption and operation time. The two multiplier modules are one such example where the default one uses a DSP module where other resources are used sparingly and in the other DSP was not used while other modules were used more. Likewise each module has its on significance to enable implementation in various ways.

IV. CONCLUSION

In this work a flexible design model for forward pass implementation of a CNN is designed.These customizable modules require less hardware and are computationally efficient. These modules can be used to build an HDL library of neural networks. The future works include improvement of speed and performance without compromise hardware utilization using new algorithms. Design of new base modules and Updating of existing modules for better performance of said base modules. Development of better communication interface for the design.



Logic Utilization	Used
Number of Slice Registers	33
Number of Slice LUTs	66
Number of fully used LUT-FF pairs	33
Number of bonded IOBs	69
Number of BUFG/BUFGCTRLs	1

Fig. 14. Accumulator module schematic, simulation result and hardware use

REFERENCES

[1] Simeone, Osvaldo. "A brief introduction to machine learning for engineers." arXiv preprint arXiv:1709.02840 (2017).
[2] Martínez Selva, José Sánchez-Navarro, Juan Bechara, A Román, F. (2006). Brain mechanisms involved in decision-making. Revista de neurologia. 42. 411-8.
[3] Nielsen, Michael A. Neural networks and deep learning. Vol. 2018. San Francisco, CA: Determination press, 2015.
[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Proc. Adv. Neural Inf.Process. Syst., 2012, pp. 1097–1105.
[5] M. A. Ponti, L. S. F. Ribeiro, T. S. Nazare, T. Bui, J. Collomosse, Everything you wanted to know about deep learning for computer vision but wereafraid to ask, in: 2017 30th SIBGRAPI Conference on Graphics, Patternsand Images Tutorials (SIBGRAPI-T), IEEE, 2017, pp. 17–41.
[6] Alyamkin, Sergei, Matthew Ardi, Alexander C. Berg, Achille Brighton, Bo Chen, Yiran Chen, Hsin-Pai Cheng et al. "Low-Power Computer Vision: Status, Challenges, and Opportunities." IEEE Journal on Emerging and Selected Topics in Circuits and Systems 9, no. 2 (2019): 411-421.
[7] Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).
[8] Solovyev, Roman A., Alexandr A. Kalinin, Alexander G. Kustov, Dmitry V. Telpukhov, and Vladimir S. Ruhlov. "FPGA implementation of convolutional neural networks with fixed-point calculations." arXiv preprint arXiv:1808.09945 (2018).
[9] Abtahi, Tahmid, Colin Shea, Amey Kulkarni, and Tinoosh Mohsenin. "Accelerating convolutional neural network with fft on embedded hardware." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 26, no. 9 (2018): 1737-1749.