ISSN 2278-3091
Volume 8, No.4, July – August 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse04842019.pdf
https://doi.org/10.30534/ijatcse/2019/04842019

FPGA Library Based Design of a Hardware Model for Convolutional Neural Network with Automated Weight Compression using K-Means Clustering

Roderick Yap[1], Goldwin Giron[2], Leonard Miguel Lanto[3], Lorenzo Garcia[4], David Sta Maria[5], Lawrence Materum[6]
[1]De La Salle University, Manila, Philippines, roderick.yap@dlsu.edu.ph
[2] De La Salle University, Manila, Philippines, goldwin_giron@dlsu.edu.ph
[3] De La Salle University, Manila, Philippines, miguel_lanto@dlsu.edu.ph
[4]De La Salle University, Manila, Philippines, lorenzo_garcia@dlsu.edu.ph
[5]De La Salle University, Manila, Philippines, lorenzo_garcia@dlsu.edu.ph
[6]De La Salle University, Manila, Philippines, lawrence.materum@dlsu.edu.ph

ABSTRACT

In this paper, a design of a synthesizable hardware model for a Convolutional Neural Network (CNN) is presented. The hardware model is capable of self-training i.e. without the use of any external processors. It is trained to recognize four numerical digit images. Another hardware model is also designed for the K-means clustering algorithm. This second hardware model is used to for compressing the weights of the CNN through quantization. Weight compression is carried out through weight sharing. With weight sharing, the system is able to save component usage. The two hardware models designed are then subsequently integrated to automate the compression of the CNN weights after the CNN completes its training. The entire design is based on fixed point arithmetic operation using VHDL as design entry tool and XILINX Virtex 5 FPGA as the target library for synthesis. After completing the design, it is evaluated in terms of hardware consumption with respect to rate of compression. When evaluating the recognition performance ability of the hardware model, digit images experimentation results have shown that the weight compression can reach as high as 60% without any negative effect on the performance of the CNN. Based on data gathered, the compression with the least hardware consumption occurs at 80 %. For the various digits trained, the CNN outputs after the training, range from 89% to 97%.

Key words: Convolutional Neural Network; K-Means Clustering; Hardware Model; VHDL; Weight Compression; Field Programmable Gate Array

1. INTRODUCTION

CNN is a popular tool used for many image recognition applications. A lot has been done on its various forms of implementation and application. In many cases, computer-based or processor-based CNN are used. There is a CNN focused on recognizing places or locations visited before [1]. Another CNN application focused on different human pose with detection of arm, torso and limb [2]. Granite tiles classification was used in [3]. In [4], CNN was applied to learning of the human eye. CNN was applied to recognize certain infrared images in [5]. In [6], CNN was used for detecting rice disease. On the other hand, a lot of researches have also focused on implementing CNN on Field Programmable Gate Arrays (FPGAs). FPGAs offer the promise of better portability due to its smaller size when compared to processors. In addition, FPGAs provides better opportunity for parallelism for some of the CNN's operations. This is in contrast with a normal processor where all tasks are executed sequentially. An FPGA based accelerator for deep CNN is proposed in [7]. Another CNN accelerator was implemented on FPGA in [8] with highlight on the use of Open Computer Language. In [9], FPGA was used for implementing CNN. A small size and a large size networks were presented. The paper also focused on memory based hyperbolic tangent implementation for faster computation. Design space exploration of FPGA Based accelerators was studied in [10]. Emphasis was given to design space and how to optimize the block resources in FPGA. Automatic Verilog Code generation for FPGA Based CNN was proposed in [11]. Alexnet and Lenet were implemented on FPGA in the said research. Similarly, [12] proposed a caffe framework-based CNN on an FPGA platform. The design provided choices for preferred CNN model including features on pipelined structures.

Aside from FPGAs, custom-made integrated circuits also known as Application Specific Integrated Circuits (ASICs) were used for CNN implementation. In [13], deep learning was implemented using on-chip silicon to take advantage of parallelism. A CNN accelerator was implemented using 65nm UMC library in [14]. A combinatorial optimization and deep learning hardware accelerator based on memristive Boltzmann machines is proposed in [15]. This

implementation highlighted better speed performance and lower energy consumption.

Like the CNN, clustering has found many applications too as evidenced by various published researches. Many of these researches were implemented using computers or processors. [16] focused on density-based clustering where normal distribution was used for analyzing data and for eventual determination of the cluster centers. Various types of clustering were analyzed and compared in [17]. Cluster evolution analysis was the subject in [18]. Clustering involving colors was done in [19]. Several clusters in parallel was highlighted in [20]. Fuzzy based clustering was the subject in [21].

Though vastly different from each other in terms of operation and purpose, CNN and Clustering have found a common application in compression. In [22], the concept of deep compression was introduced. The weights of a neural network can be quantized using K-means clustering. The goal of this quantization was to impose weight sharing. Through weight sharing, memory resources are saved.

2. SIGNIFICANCE OF THE STUDY

The highlight of this research is the implementation of a CNN and K-Means clustering in an FPGA using hardware modeling approach. FPGA offers the advantage of small size, lower power consumption and better portability. With CNN and K-means clustering integrated, the CNN can commence with K-means clustering based weight compression routine automatically, as soon as the CNN's training for image recognition is finished. This system eliminates the use of a computer to do the CNN task and clustering task. For applications like embedded systems, where image recognition is needed but the required prototype is small, this research will become beneficial because its size is much smaller than that of a computer.

3. THEORETICAL CONSIDERATION

3.1 Some Mathematical Operations in CNN

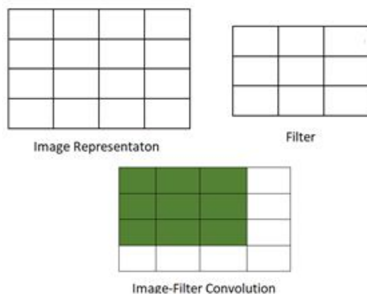

Figure 1: Image Filter Convolution

In CNN, the first calculation would usually be the convolution. Shown in Figure 1 is a sample image named image A. The image is represented as 4x4 pixel. To its right is a 3x3 Filter initially set by the designer. For both image and filter, every cell is identified by its row index, i and column index, j. Let A represent the 4x4 image and B represent the 3x3 filter. Every cell of A is denoted by A(i, j) and every cell in B is denoted by B(i, j). Convolution involves positioning the filter over the image and subsequently sliding the filter until the entire image is covered. The first position is shown in Figure 1 as represented by the area shaded in green. Convolution involves getting the sum of the product of every pair of cells that coincide. Let this sum be represented by C_1. C_1 is mathematically represented by Equation 1. Similar equations can be obtained as the filter slides through the image by a single stride.

$$C_1 = \sum_{i=1}^{3} \sum_{j=1}^{3} A(i, j) B(i, j) \tag{1}$$

Each result obtained from convolution are subsequently fed to an activation function to introduce non linearity. One popular activation function is the sigmoid function. This is shown in Equation 2 [23]. In the equation, C_i represents a result coming from the convolution operation. Aside from Sigmoid, other popular activation function include hyperbolic tangent [9] and the Rectified Linear Unit (ReLU).

$$I_i = \frac{1}{1 + e^{-C_i}} \tag{2}$$

In many applications, the results obtained from the activation function would serve as inputs to a fully connected layer. Figure 2 shows a sample representation for a two-layer Fully Connected Network. The input nodes I_1 to I_4, represent the results coming from the previously mentioned activation function. Output nodes O_1 to O_4 can be treated as Data Processors with corresponding outputs S_1 to S_4. The connection from every input node to every output node carries a weight value. Seen in the figure are the four weights W_1 to W_4 linked to output node O_1. The output S_1 is simply the sum of the products of the inputs and its corresponding weight values. This is shown in Equation 3. In many applications, a fixed value known as bias, is added to the sum. Similar to S_1, S_2 to S_4 would also be a function of the product of its respective input weight values and the corresponding input associated with it.

$$S_1 = I_1W_1 + I_2W_2 + I_3W_3 + I_4W_4 \tag{3}$$

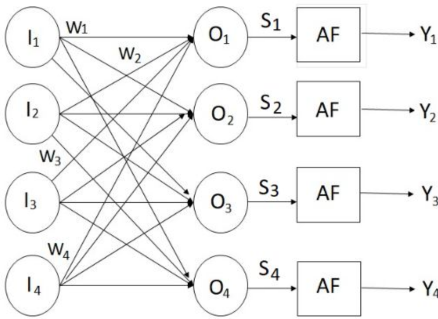

Figure 2: Sample Fully Connected Layer with Activation Function

The computed outputs, S_1 to S_4 would undergo another activation function. Once again, previously mentioned functions such as Sigmoid and Hyperbolic Tangent are some of the available choices. For classification however, Softmax activation function [24] is a popular choice. Equation 4 shows the Softmax Activation Function. S_i is representing each output S_1 to S_4 as seen in Figure 2. S_{max} is the maximum value among S_1 to S_4. N represents the number of output nodes. For Figure 2, N = 4.

$$Y_i = \frac{e^{(S_i - S_{\max})}}{\sum_1^N e^{(S_i - S_{\max})}} \quad (4)$$

The variables Y_1 to Y_4 seen in Figure 3.2 represent the official outputs of the CNN. These computed outputs using Equation 4, are compared against some target values set by the designer. The goal is to make the Y values reach the target value. If the target is not yet met, then the CNN must update its Fully Connected Layer Weights and also the Filter weights in preparation for the next round of iteration. Updating the Fully Connected Layer weights is done using Gradient Descent [23]. Equation 5 shows the formula for updating the first weight, W_1. ε_1 represents an error for output Y_1 i.e. the difference between Y_1 and its target value. This equation is inspired by [25].

$$\frac{\partial \varepsilon_1}{\partial W_1} = \frac{\partial \varepsilon_1}{\partial Y_1}\frac{\partial Y_1}{\partial S_1}\frac{\partial S_1}{\partial W_1} \quad (5)$$

If W_{1new} is to represent the new or updated value for W_1, then W_{1new} is computed as shown in Equation 6 [23]. Symbol α is a learning rate whose value is set by the designer. The updating of all the other Weight values would follow the format of Equations 5 and 6.

$$W_{1new} = W_1 - \alpha \frac{\partial \varepsilon_1}{\partial W_1} \quad (6)$$

3.2 Some Mathematical Operations in K-Means Clustering

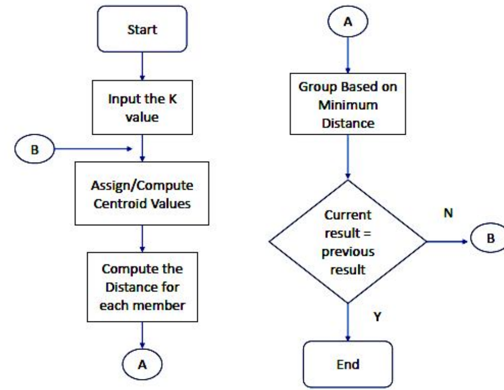

Figure 3 : K-Means Clustering Behavior

Shown in Figure 3 is a flowchart description of how the K-means clustering algorithm works. The variable K represents the number of clusters set by the designer. The number of clusters would also determine the number of centroid values. Each input undergoes a Euclidean distance computation with every centroid value. Given two points $A(q_1,q_2, \ldots q_N)$ and $B(v_1,v_2, \ldots v_N)$, the Euclidean distance, d(A,B) between points A and B is shown in Equation 7. The distance values for every input are subsequently analyzed to search for the minimum The minimum values determine to which cluster does an input belong to. The next round of iteration is started by getting the average of every input belonging to a particular cluster. All clusters will undergo average computation. This average values serve as the new centroid values for the next iteration. The iteration stops when the current set of centroid values is the same as the previous set of centroid values.

$$d(A,B) = \sqrt{(q_1 - v_1)^2 + (q_2 - v_2)^2 + \ldots + (q_N - v_N)^2} \quad (7)$$

4. METHODOLOGY

In doing the hardware modeling for both the CNN and K-means clustering, the design entry tool used is VHSIC Hardware Description language or VHDL. The VHDL code is then synthesized to a hardware equivalence using the Virtex 5 FPGA library. XILINX ISE series is the tool used for VHDL coding, synthesis and FPGA implementation. Multipliers, multiplexers and other components mentioned are all coded in VHDL.

Figure 4 shows how the CNN is transformed to be able to acquire external data for its weights. The left side of the figure shows the CNN's block behavior during its usual training routine. For every iteration during the training process, the CNN computes a new set of weights also known as updated weights. The updating of weights is intended to bring the output targets to the desired value. The updated weights can be treated as output ports that are fed back to the module itself. A write signal within the module is asserted high to let the fed

back weights replace all the existing weights inside the CNN block. The right side of Figure 4 shows the insertion of a 2 x 1 multiplexer. With the use of multiplexer, the fed back weights can come from an outside source. This outside source will eventually become the clustered weights.

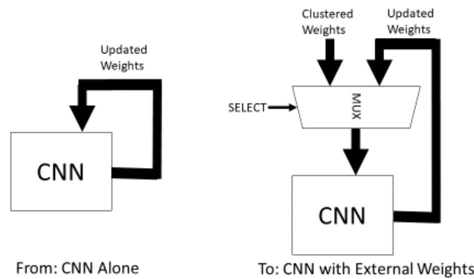

Figure 4: Transforming CNN Block to Receive External Weights

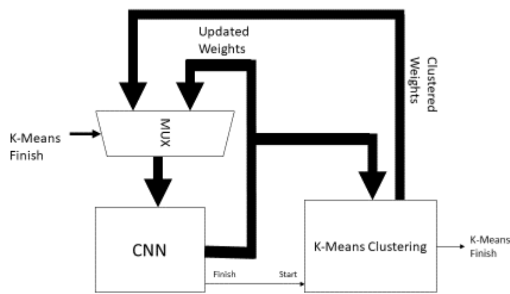

Figure 5: Interfacing the CNN to the K-Means Clustering Block

Figure 5 shows how the CNN block can be interfaced to the K-Means Clustering block. The other input of the multiplexer mentioned in the previous figure as outside source, can now be connected to the Clustering block. The operation of the entire system would start with the CNN doing its training for recognizing the fed images. Once the training is over, it will send a signal to the Clustering block to start the clustering operation. The output weights of the CNN are readily available as inputs to the Clustering block. When clustering operation starts, a series of iterations takes place until its outputs are no longer changing. When this happens, a "finish" signal from the Clustering module is asserted high. This high state signal is assigned to control the select line of the multiplexer. The multiplexer is then set to select the clustered weights inputs as its output. The clustered weights become the new set of weights of the CNN.

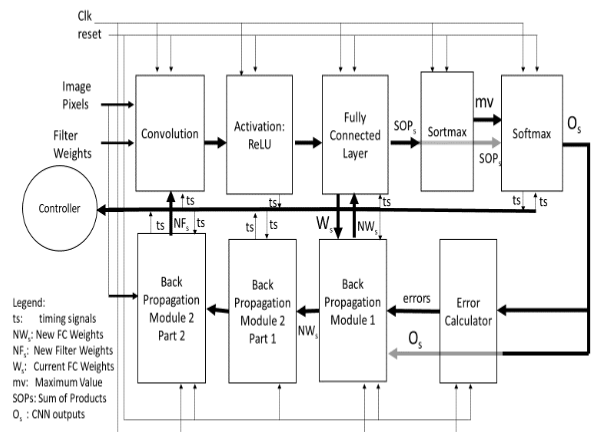

Figure 6: CNN Control and Datapath

Figure 6 shows the hardware model for the control and datapath of the CNN. The input image adopted in this research is a black and white 4 x 4 image. The digit images used for recognition are mapped into the 4x4 pixel image size. For all numerical operations, fixed point is used to save hardware requirement. The input image enters the Convolution block. Three filters of size 3x3 each were used for the convolution process. Inside this block are multipliers and adders that perform the operations represented by Equation 1. The result of the convolution operation enters a ReLU activation function. Inside the ReLU module, every input has a corresponding output. The circuit inside ReLU monitors the sign bit of every input. If the sign bit is a 1, then the input number is negative and the corresponding output is a 0 otherwise, the output equals the input. The output of the activation function then enters the Fully Connected Layer block. With 3 filters in convolution block, the fully connected layer block has 12 inputs and 4 outputs. Inside this block are weight values that undergo multiply and add operation with the inputs as represented by Equation 3. The output of the Fully Connected layer undergoes another activation function called Softmax. Softmax requires the identity of the maximum value from among its inputs. To do this, a search for maximum module known as Sortmax module, is inserted between the Fully Connected Layer and the Softmax module. The maximum value together with the complete set of Fully Connected Layer Output are mathematically processed using the Softmax activation function. The circuit inside the Softmax module made use of lookup table to satisfy Equation 4.

The CNN results are compared with set target values to verify if the target has been achieved. An error is computed which represents the difference between the current result and the target. If the targets are not yet achieved, a backpropagation routine is carried out. The goal of the backpropagation routine is to update the weight values of the Fully Connected layer and the Convolutional layer. With a new set of weights, the image undergoes a new round of iteration from convolution all the way to the Softmax activation function. The results are once again examined to see if the target has been met. If not, the weights are again

updated and the entire operation is repeated until the targets are met.

Table 1: Major Port Summary for CNN

	Major I/O Ports No. of Bits	
Module	**Inputs**	**Outputs**
Convolution	12 (Filter Weights)	1 2
ReLU	12	12
Fully Connected Layer	12	24
Softmax	24	20
Error Calculator	21 (sign extended)	21
Backpropagation 1	21(Error) 12 (Weights) 20(Softmax)	12 (Weights)
Backpropagation 2 Part1	12 (Weights) 20 (Softmax)	32
Backpropagation 2 Part1	32 (part 1)	12

As seen in figure 6, there are two backpropagation modules 1 and 2. The first module is responsible for updating the weights of the Fully Connected Layer block. Inside this block is a circuit that performs Equations 5 and 6. The second module is responsible for updating the filter weights inside the Convolutional layer block. The second module is further subdivided into 2 submodules named Part1 and Part 2. Part 1 performs the backward pass routine while Part 2 performs convolution using the results from Part 1. The result of the convolution process are used as new set of filter weights. Table 1 shows the port size summary adopted for each module in the design of the CNN.

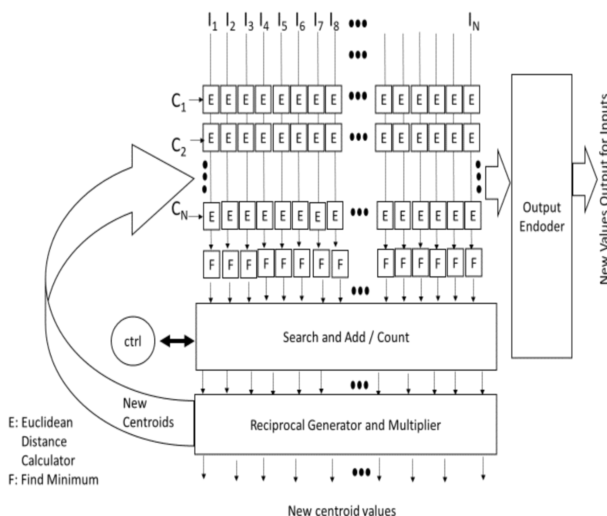

Figure 7: K-Means Clustering Hardware Model

Table 2: Major Port Summary for K-Means Clustering

	Major I/O Ports No. of Bits	
Module	**Inputs**	**Outputs**
Euclidean Distance Calculator	12 (Weights)	12
Find Minimum	12	12(Weights) 6 (Tag)
Search and Add	12 (Weights) 6 (Tag)	20
Reciprocal Generator/Multiplie r	20	32
Output Encoder	12 6 (Tag)	12

Shown in Figure 7 is the hardware model for the K-Means clustering operation. The initial centroid values can randomly assigned at the start of the operation. The number of centroid values is assigned by the user. This number represents the variable K in the K-means clustering operation. This value of K also represents the number of groups for this operation. For every iteration, every group can be treated as being headed by the centroid value. For example, if the user assigns K as 5, this means there are 5 groups and 5 centroid values heading each group. All input data undergo Euclidean Distance calculation with each centroid value. The block labeled E in the figure computes the Euclidean distance. The computed Euclidean distance values for every input undergo a search for the minimum. This is done by the block labeled F. The minimum value dictates to which centroid value group does one particular input belong to. Table 2 shows the port size summary adopted for each module in the design of the K-Means Clustering.

After determining to which group does an input belong to, the system prepares for the next iteration. This is done by getting the average of each group. For every group, the inputs are added and divided by the number of inputs. The computed average serve as the new centroid value for the new round of iteration. The iteration only stops when the current set of centroid values is exactly the same as the previous set of centroid values. It should be noted that the output encoder takes in the significant 12 bits input from the 32 bits output of the Reciprocal Generator/Multiplier module.

5. DATA AND RESULTS

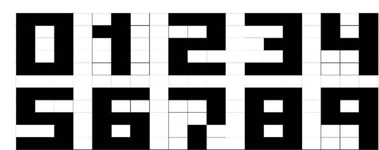

Figure 8: The Images for Training

Table 3 : Sets of Digits Trained

Set	Digits Trained			
A	0	1	4	7
B	7	1	6	9
C	3	5	4	1
D	1	8	2	7

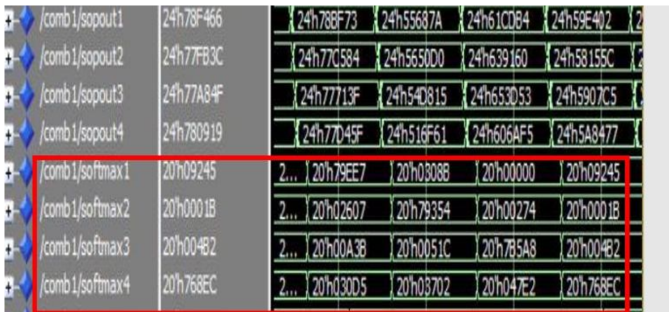

Figure 9: Set A Simulation Screenshot

Table 4: CNN Outputs Final Values After Training

Set	Output1	Output2	Output3	Output 4
A	0.95	0.94	0.96	0.92
B	0.89	0.89	0.90	0.91
C	0.97	0.93	0.95	0.96
D	0.95	0.94	0.94	0.91

The design of the hardware model was carried out using fixed point arithmetic operation. Figure 8 shows the digit images used for training the hardware model. Since the hardware model adopted in this research has four outputs only, the digit images are assembled into sets of four. Table 3 shows the four sets of digits adopted for the training. Figure 9 shows a screenshot for Set A's simulation at the end of the training. It can be seen in this figure that the four outputs, targeted to reach 0.99 during the training process, reached at least 76XXX in hexadecimal representation. Translating to binary, the left most bit is the only integer bit. 76XXX translates to at least 0.92 in decimal format as final value at the end of the training. Just like Figure 9, the other sets were able to converge close to the target value. Table 4 summarizes the final output values for each set. Each set was able to recognize the four digits assigned to it after the training.

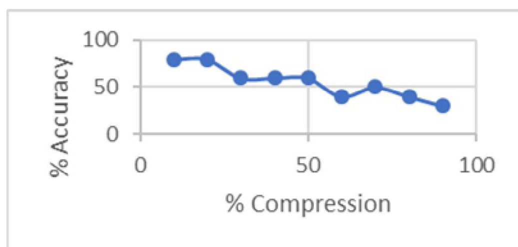

Figure 10: Set A Accuracy vs % Compression Result

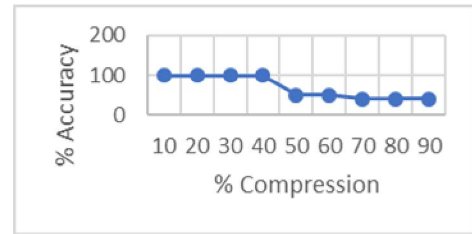

Figure 11: Set B Accuracy vs % Compression Result

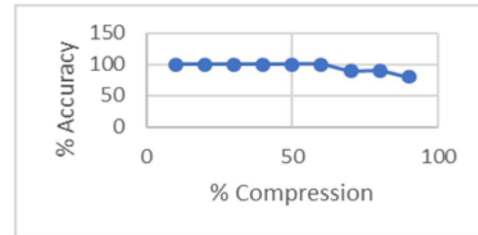

Figure 12: Set C Accuracy vs % Compression Result

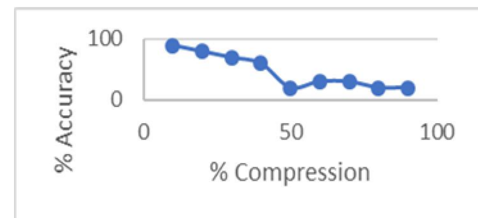

Figure 13: Set D Accuracy vs % Compression Result

Table 5: Effect of Compression on Component Used for module with Weights

Component	Compression				
	0%	50%	60%	80%	90%
Slice LUTs	298	266	273	251	258
Fully Used LUT FF Pair	144	133	126	115	116
Number used as logic	294	262	269	247	254

After training the CNN, weight compression, using K-means clustering was applied and the CNN performance was once again tested. Various compression rates were applied to see up to how much compression ratio can the CNN withstand without any negative effects on the recognition performance. In doing the test, percentage accuracy is based on the CNN's performance with compression and without compression. For example, a 100% accuracy means the performance of the compressed CNN and the uncompressed CNN are the same in terms of recognizing the set of digit images fed to it. Figures 10 to 13 show the result of compression vs accuracy for each set. Of the four sets, Set C showed the best performance at it was able to reach as high as 60% compression while maintaining 100% accuracy as seen in Figure 12. Table 5 shows the FPGA component logic used for various rates of compression. The table represents the specific module where

the weights reside. It can be seen in the table that the higher the rate of compression, the lesser is the component usage in comparison to "no compression applied". No compression is the 0% compression column

6. CONCLUSION

In this research, a CNN architecture was designed using hardware modeling. CNN's ability to train and recognize image using FPGA proved to be possible even without using any external processor. The hardware model is implementable on Virtex 5 FPGA using fixed point arithmetic to save hardware components. A K-means clustering algorithm was also designed using hardware modeling. Both made use of Virtex 5 FPGA library for synthesis. The two designs were connected together to form an automatic CNN weight compression routine right after the CNN's training process. Data gathered shows that the compression does have effect on the component usage. The higher the rate of compression, the more significant is the component savings in comparison to component usage for no compression. The CNN however suffers from accuracy problem for high compression rate. Of the four sets experimented in this research, one set i.e. set C was able to maintain a 100% accuracy for compression rate as high as 60%.

7. RECOMMENDATION

It is recommended for future studies to explore a higher CNN size that can integrate all digits 0 to 9 in just 1 set. This however will require more filters for convolution, hence a higher equivalent circuit size and a higher density FPGA will be needed. The larger circuit size for CNN will also mean more weights in its fully connected layer. More weights will mean more inputs to the clustering module. Thus, the circuit size for the clustering algorithm will also increase. Another area for future study can also focus on providing intelligence to the clustering algorithm when evaluating its effect on the CNN. The clustering will work by itself to search for the best compression rate without any intervention from external users.

Other areas to explore for future studies can focus on big data challenge to the CNN. In this case, hardware modeling of the MapReduce [26] algorithm interfaced to the hardware model of the CNN, can be a good tool for efficient data processing. The hardware model to be designed for MapReduce algorithm can be integrated to the large-scale version of the design done in this paper for future studies. Still, another area to explore for future studies is the implementation of the entire design in Application Specific Integrated Circuit (ASIC) instead of FPGA. The advantage of going ASIC is the smaller circuit size because this implementation is custom-made based on the designer's requirements. Hence, it can lead to lower cost for mass production. CNN has a lot of addition operations happening in it. The K-means clustering algorithm, which has many subtraction operations happening in it, in principle can also be considered as having lots of addition operations happening in it. This is because in many digital applications, subtraction is carried out by adding the minuend to the two's complement of the subtrahend. If CNN with K-means clustering is to be implemented in ASIC, it would be interesting to explore a future study on the effects of using transmission gate logic based Full Adder FinFET [27], for all the adder circuits of the CNN and K-means clustering.

ACKNOWLEDGEMENT

The authors would like to thank ERDT through DOST-SEI for the funding provided in this research. The authors also thank Dr. Melvin Cabatuan for his valuable insights on CNN training.

REFERENCES

1. M. Lopez-Antequera, R. Gomez-Ojeda, N. Petkov, and J. Gonzalez-Jimenez. **Appearanceinvariant place recognition by discriminatively training a convolutional neural network,** *Pattern Recognition Letters*, vol. 92, pp. 89 – 95, 2017. https://doi.org/10.1016/j.patrec.2017.04.017
2. P. Witoonchart and P. Chongstitvatana. **Application of structured support vector machine backpropagation to a convolutional neural network for human pose estimation,** *Neural Networks*. vol. 92, pp. 39 – 46, 2017. Advances in Cognitive Engineering Using Neural Networks. https://doi.org/10.1016/j.neunet.2017.02.005
3. A. Ferreira and G. Giraldi. **Convolutional neural network approaches to granite tiles classification**, *Expert Systems with Applications*, vol. 84, pp. 1 – 11, 2017. https://doi.org/10.1016/j.eswa.2017.04.053
4. K. Ahuja, R. Islam, F. A. Barbhuiya, and K. Dey. **Convolutional neural networks for ocular smartphone-based biometrics**, *Pattern Recognition Letters*, vol. 91, pp. 17 – 26, 2017. Mobile Iris Challenge Evaluation (MICHE-II). https://doi.org/10.1016/j.patrec.2017.04.002
5. Z. Fan, D. Bi, L. Xiong, S. Ma, L. He, and W. Ding. **Dim infrared image enhancement based on convolutional neural network**, *Neurocomputing,* 2017. https://doi.org/10.1016/j.neucom.2017.07.017
6. Y. Lu, S. Yi, N. Zeng, Y. Liu, and Y. Zhang. **Identification of rice diseases using deep convolutional neural networks**, *Neurocomputing*, 2017.
7. R. Morcel, M. Ezzeddine, and H. Akkary**. FPGA-based accelerator for deep convolutional neural networks for the spark environment**, *IEEE International Conference on Smart Cloud*, pp. 126–133, 2016. https://doi.org/10.1109/SmartCloud.2016.31
8. Z. Wang, F. Qiao, Z. Liu, Y. Shan, X. Zhou, L. Luo, and H. Yang. **Optimizing convolutional neural network on**

FPGA under heterogeneous computing framework with opencl, *IEEE Region 10 Conference (TENCON)*, pp. 3433–3438, 2016.
https://doi.org/10.1109/TENCON.2016.7848692

9. S. Ghaffari and S. Sharifian. **FPGA-based convolutional neural network accelerator design using high level synthesize**, *2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS)* , pp. 1–6, 2016.
https://doi.org/10.1109/ICSPIS.2016.7869873
10. A. Rahman, S. Oh, J. Lee, and K. Choi. **Design space exploration of FPGA accelerators for convolutional neural networks**, *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 147–1152, 2017.
https://doi.org/10.23919/DATE.2017.7927162
11. Z. Liu, Y. Dou, J. Jiang, and J. Xu. **Automatic code generation of convolutional neural networks in FPGA implementation**, *International Conference on Field-Programmable Technology (FPT)*, pp.61–68, 2016.
12. R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi. **Caffeinated FPGAs: FPGA framework for convolutional neural networks,** *International Conference on Field-Programmable Technology (FPT)*, pp. 265–268, 2016.
https://doi.org/10.1109/FPT.2016.7929549
13. S.W. Park, J. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H. J. Yoo. **An energy-efficient and scalable deep learning/inference processor with tetra-parallel mimd architecture for big data applications,** *IEEE Transactions on Biomedical Circuits and Systems,* vol. 9, pp. 838–848, Dec 2015.
14. **Origami: A convolutional network accelerator**, *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, 2015.
15. M. N. Bojnordi and E. Ipek. **Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning**, *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–13, March 2016.
https://doi.org/10.1109/HPCA.2016.7446049
16. C. Jinyin, L. Xiang, Z. Haibing, and B. Xintong. **A novel cluster center fast determination clustering algorithm**, *Applied Soft Computing*, vol. 57, pp. 539 – 555, 2017
https://doi.org/10.1016/j.asoc.2017.04.031
17. A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin, **A review of clustering techniques and developments**, *Neurocomputing*, 2017.
https://doi.org/10.1016/j.neucom.2017.06.053
18. R. Ramon-Gonen and R. Gelbard. **Cluster evolution analysis: Identification and detection of similar clusters and migration patterns**, *Expert Systems with Applications*, vol. 83, pp. 363 – 378, 2017.
https://doi.org/10.1016/j.eswa.2017.04.007
19. I. Douven. **Clustering colors**, *Cognitive Systems Research*, vol. 45, pp. 70 – 81, 2017.
https://doi.org/10.1016/j.cogsys.2017.05.004
20. A. Cornujols, C. Wemmert, P. Ganarski, and Y. Bennani. **Collaborative clustering: Why, when, what and how**, *Information Fusion*, vol. 39, pp. 81 – 95, 2018.
https://doi.org/10.1016/j.inffus.2017.04.008
21. K. S. Arikumar and V. Natarajan. **Fuzzy based dynamic clustering in wireless sensor networks**, *Eighth International Conference on Advanced Computing (ICoAC)*, pp. 77–82, 2016.
22. H. M. .W. D. Song Han. **Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding**, *https://arxiv.org/pdf/1510.00149.pdf*, 2016.
23. Tariq Rashid. **Make Your Own Neural Network**.
24. **How to implement the Softmax function in Python**, *[Available Online:] https://stackoverflow.com/questions/34968722/how-to-implement-the-softmax-function-in-python*
25. M. Mazur. **A step by step backpropagation example**, *[Available Online:] https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/* .
26. D. Ahamad, MD M. Akhtar, S. A. Hameed. **A Review and Analysis of Big Data and MapReduce**, *International Journal of Advanced Trends in Computer Science and Engineering*, Vol. 8, No. 1, Jan-Feb, 2019.
https://doi.org/10.30534/ijatcse/2019/01812019
27. Ch.Rajesh Babu, T. Venkatesh, E. J. Rao, U. V. Raju. **Conventional Full Adder FinFET Implementation using Transmission Gate logic,** *International Journal of Advanced Trends in Computer Science and Engineering*, Vol. 7, No. 6, Nov-Dec, 2018.
https://doi.org/10.30534/ijatcse/2018/11762018