Energy Efficient design for Accelerated Sparse Convolutional Neural Network

Rahul Gupta and G. K. Sharma
ABV-Indian Institute of Information Technology and Management, Gwalior 474015, India
Email: rahulgupta291093@gmail.com, gksharma@iiitm.ac.in

***Abstract*—The state-of-the-art deep learning architecture, Convolutional Neural Network (CNN), widely used in applications such as speech recognition, face detection, natural language processing & computer vision. MAC units which are an integral part of CNN require large computations and memory resources. They result in more power dissipation for a low power embedded device such as IOT. Hence, the hardware implementation of CNN to produce high throughput is one of the challenges nowadays. Therefore, sparsity is introduced in weights by a non-linear method with little compromise in accuracy. Experiment results also show the enhancement of 52% sparsity with a 4% loss in accuracy. In addition, to perform SIMD operation, an indexing module is proposed in the fully-connected layer to perform only effective operations without multiplication. It is used along with sparsity to give better results as compared to existing work. Cadence RTL compiler results show that the proposed indexing module saves 1.3 nj of energy as compared to existing method.**

***Index Terms*—Sparsity, Ternary weights, Quadratic clip function, Activation Indexing Module**

I. Introduction

Convolutional Neural Networks (CNNs) are broadly accepted as deep learning algorithm used for computer vision, speech recognition & human activity detection applications. Based on these applications, different architectures are proposed such as LeNet [9], AlexNet [10], VGG [11] & GoogLeNet [12] to deliver high throughput. Such dense architectures can be efficiently implemented on software as TensorFlow, Keras & Theano frameworks are available. However, the hardware implementation of these dense CNNs is still a challenge. As they require a lot of floating point multiply and accumulate (MAC) operations.

One of the promising ways of implementing dense CNN is by exploiting sparsity. As there exist a considerable amount of ineffectual activations and weights whose values are zero. Therefore in Sparse CNN, a significant amount of energy can be saved by reducing these MAC operations. Also, it is observed that over 60% of the activations and weights are zero in many convolutional layers of AlexNet [10]. In this direction, [7] proposed concise convolution rule (CCR) to transform convolution into multiple effective and ineffective sub-convolutions. In ineffective convolutions, neurons and synapses which do not contribute to the final results are eliminated. It is also proven that a large fraction of activations & weights can be pruned to zero with a little loss of accuracy. A precision-reduced approach, Binary Connect [15] simplify computation by restricting weights to only two possible values. It eliminates the multiplications and can be treated as a regularization method. But the Binary connect is not satisfactory. Based on this, Ternary Connect (TC) [1] transform the weight to three possible values (i.e, +1, 0, -1). It improves the performance of CNN model as well as help in parameters reduction. An Updated version of TC i.e, Sparse Ternary connect (STC) [3] counters uneven distribution of zeros in Convolutional kernels and weight matrix in the fully-connected layer. It also eliminates floating point addition operations. However, the MAC unit is required to process these data concurrently. To accelerate the computation process, SIMD-like MAC structure is proposed by [4] for sparse CNN model. In that paper Dual Indexing Module (DIM) is proposed which align irregular distribution of non-zero values of activation and weight. But in this technique floating point multiplications remain the bottleneck for a low power device.

In this work LeNet architecture is trained by ternary weights. These weights are obtained by Quadratic Clip Function (QCF) which improves the sparsity as in case of existing work. There after these ternary weights are used in Activation Indexing Module (AIM). AIM performs only effective accumulation operations by eliminating multiplications.

II. Background and Related Work

A. Ternary Connect (TC)

[1] presents a method called Ternary Connect (TC) which is an upgrade version of Binary Connect (BC) to eliminate floating point multiplications in CNN. TC eliminates most of the multiplication operations during the training phase for deep neural networks. Whenever convolution operation is performed, TC first clips the original weights in the range of [-1, 1], these clipped weights are converted into the ternary set as shown in figure 1.

Ternary weights which are also termed as Stochastic weights used in place of the floating-point or real-value weights [2]. It is defined as:

$$CF = Max(-1, Min(ow, 1))$$

TC uses Clipped weights W ∈ [-1, 1] to convert to the ternary weights $W_t \in \{-1, 0, 1\}$ in the form of probability.

If W $\geqslant$ 0,

$$P(W_t = 1) = W; P(W_t = 0) = 1 - W$$

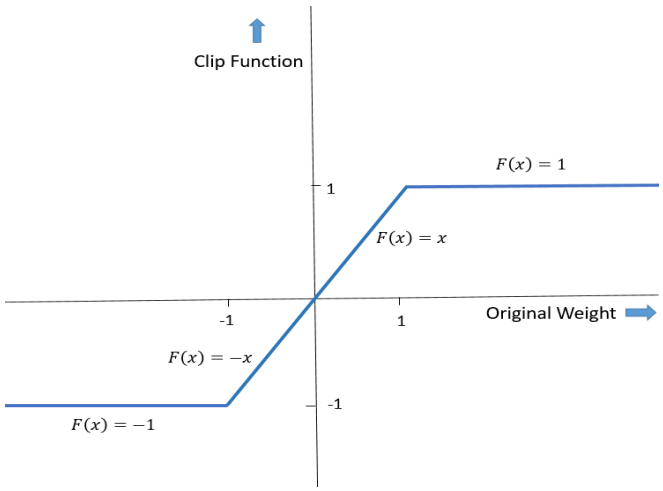

Figure 1. Existing clip function clips the original weights in [-1,1] range.

If W < 0,

$$P(W_t = -1) = -W; P(W_t = 0) = 1 + W$$

Figure 2 shows the worst case in which no zero is present after conversion. It is because no weight in the original kernel lies in (-0.1, 0.1). However, in this technique, the floating

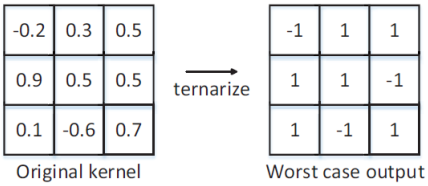

Figure 2. Worst case output in case of TC [3].

point additions remain the barrier for implementing on low-power and resource-efficient hardware device. It is due to uneven distribution of a certain amount of zero-weights on convolutional filters/kernels, which makes it difficult to take the advantage from zero-weights in the terms of hardware implementation.

B. Sparse Ternary Connect (STC)

[3] introduced a method called Sparse Ternary Connect (STC) which enhanced the sparsity into the TC method which makes the zero-weights evenly spread across filters as shown in Figure 3. It further decreases the floating point addition operations in convolutional layers and diminishes the size of computation units in actual implementation on hardware. STC performs the task in both forward and backward propagations. First of all, the real-value weights are transformed into the sparse ternary weights. Thereafter the convolution operations are performed with the transformed weights. These weights must be kept in sufficient precision while updating during backpropagation. STC introduced the parameter ρ which decides the number of adders required for the accumulation. The value of ρ = 0.5 for a given (3 X 3) matrix states at least for four zeros no adders are required.

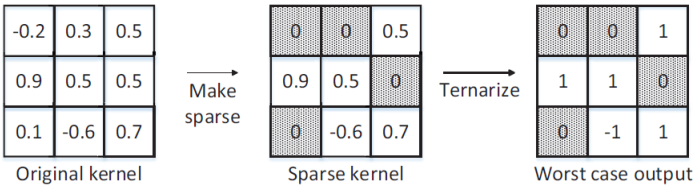

Figure 3. Worst case output in case of STC [3].

C. Dual Indexing Module (DIM)

[4] presents a novel approach Dual Indexing Module (DIM) to effectively and efficiently handle the positioning of randomly distributed non-zero elements in activations and weights. Figure 4 shows DIM algorithm in which the indices

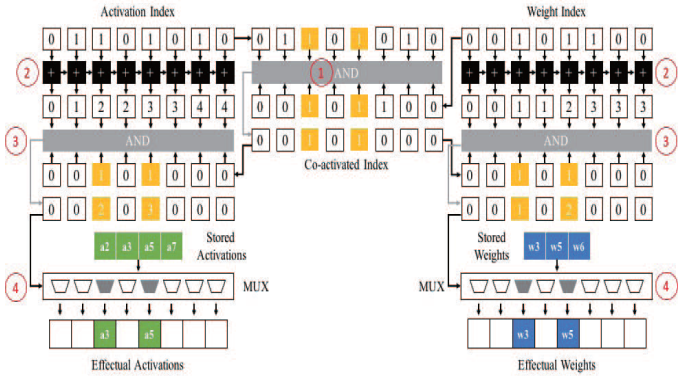

Figure 4. The architectural design of Dual Indexing Module [4].

of weights and activations are checked concurrently and the effectual activation-weight pairs are recoded and bring out for computation. It can be easily integrated into CNN architecture in the form of SIMD accelerator to support compressed-sparse format with little overhead. With DIM, a SIMD-like accelerator can take favor of sparse CNNs in the form of small memory footprint and cut down of energy consumption.But DIM technique requires multiplication operations which can be avoided by using STC/TC. Also, it needs extra multiplexers which may not be used in the computation.

III. Proposed Work

This section explains proposed algorithm and architecture which address the identified shortcomings in the recent related research.

A. Ternary weight conversion

It is a method by which floating point weights are converted into fixed point (2 bit) weights. Here 1 bit represents signed value and other 1 bit represents magnitude value. First floating point weights are passed to clip function which keeps the weights (W_c) in [-1,1] range. Then these weights are converted into Ternary set by Ternary Set Conversion.It is

defined as:
If $W_c \geqslant 0$,

$$P(W_t = 1) = W; P(W_t = 0) = 1 - W_c$$

If $W_c < 0$,

$$P(W_t = -1) = -W; P(W_t = 0) = 1 + W_c$$

Figure 5 shows floating point weights are converted into Ternary weights by using two methods i.e, Existing Clip Function & Quadratic Clip Function. In this conversion, QCF generates more sparse value (shown in red color) as compare to ECF.

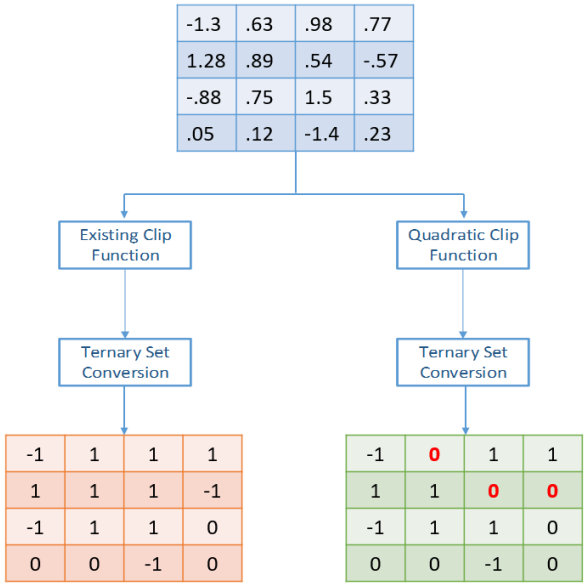

Figure 5. Ternary weight conversion flow.

1) Quadratic Clip Function: First technique deals with the enhancement of Sparsity in weights of the kernel matrix. [5] & [3] proposed clip function which fails to create sparsity in the worst case scenario. In this direction, Quadratic clip function is proposed to enhance the Sparsity by imposing non-linearity which can deal with the worst case scenario. It is define as:
If $W \geqslant 0$,

$$W_c = Min(W^2, 1)$$

If $W < 0$,

$$W_c = Max(-W^2, -1)$$

Figure 6 is the plot between the clip function (at Y-axis) and the original weight (at X-axis). The curve which yields the clip weight is divided into 4 segments. The Clip function output is equal to 1 for the original weight greater than 1, and equal to -1 for the original weight less than -1. $F(x) = x^2$ & $F(x) = -x^2$ will train the original weight near zero due to non-linearity and pass this to ternary set which finally makes most of the weights equal to zero.

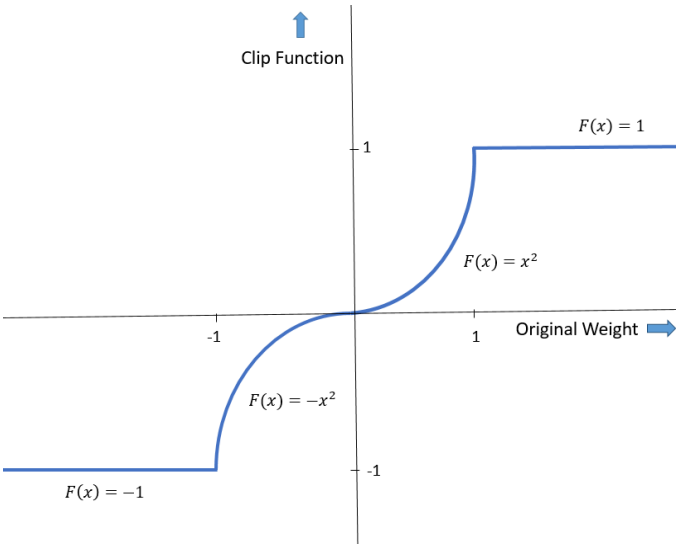

Figure 6. Clip function which clip the original weights in [-1,1] range.

2) Algorithm for Ternary weight updation: Here, algorithm 1 is the pseudo code to show how ternary weights via Quadratic clip function are calculated in the fully-connected layer. The process for algorithm 1 is divided into two part:
1. Forward propagation
2. Backward propagation

Algorithm 1 Ternary weights are calculated for the weight matrix of dimension (m, n) in the fully-connected layer l. Here, QCF refers to the Quadratic Clip Function & TSC is the Ternary Set Conversion.

Require: Weights (W) and bias (b) are updated after each iterations by calculation of cost function (C) & learning rate (η).

Forward propagation:

for each element in W **do**
$W_c \leftarrow$ ***QCF****(W)*
$W_t \leftarrow$ ***TSC****(W_c)*
end for
Compute fully-connected output y knowing W_t and b.

Backward propagation:

Gradients of present layer are calculated from error signal of next layer δ_{l+1}.
Compute $\delta_l = \frac{\partial C}{\partial y}$ knowing δ_{l+1} and W_t
Compute $\delta_l = \frac{\partial C}{\partial W_t}$ and $\delta_l = \frac{\partial C}{\partial b}$ knowing δ_{l+1} and y
Update W: W $\leftarrow$ clip($W - \eta \frac{\partial C}{\partial W_t}$)
Update b: b $\leftarrow b - \eta \frac{\partial C}{\partial b}$

In forward propagation, the Ternary weights W_t are calculated from the original weights W. In that process, firstly the

clipped weights W_c are generated by passing W to Quadratic clip function and then ternary set conversion convert W_c to W_t. Now the output of the fully-connected layer is obtained from W_t & b.

In backward propagation, the updation of weights and bias are taking place. In that process, firstly the cost function C and learning rate η are calculated and then the current layer is computed by knowing the error signal from the next layer.

B. AIM architecture

Second technique deals with the proper alignment of Weights and Activations in the fully-connected layer to get the benefit from Sparsity. [1] proposed Dual indexing module (DIM) to efficiently examine the positioning issue. But in DIM technique weights are not present in Ternary format to reduce multiplier in MAC operations. Moreover, it requires multiplexers which may not be used in the computation. Therefore, an Activation indexing module (AIM) is proposed which uses the concept of Sparsity in weights and activations in the fully-connected layer to make its operation multiplier less.

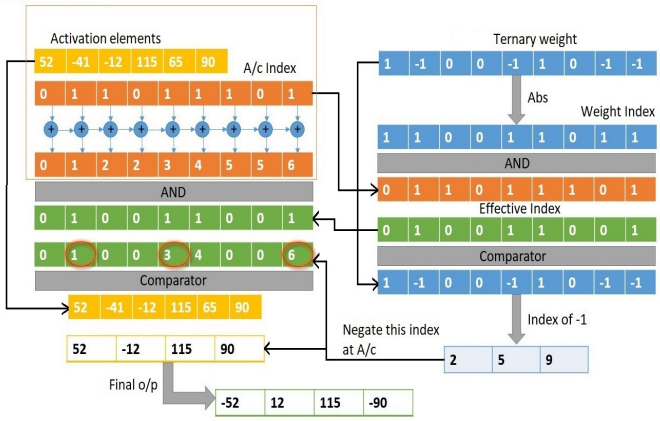

Figure 7. The hardware design of Activation Indexing Module.

Figure 7 shows the architecture of AIM algorithm for FPGA design. This algorithm is executed in 4 steps. In this, indexing of activation is calculated for the first time during operation.

Step 1: In this Ternary weights are converted into absolute one which can be treated as the weight index.

Step 2: In this AND operation is performed between the activation index and the weight index. Its output is stored in the effective index.

Step 3: Now this effective index is used simultaneously in two places. In first one, it is passed to a comparator along with Ternary weights. It gives an effective value of index where Ternary weights are -1. In second case, it helps in finding the index of non-zero activation elements.

Step 4: This is the last step in which the accumulation of effective activation elements takes place. This can be done by changing the sign activation elements where Ternary weights are -1.

IV. Implementation Results and Analysis

LeNet architecture shown in figure 8 is implemented on Python 3.6 IDLE. This architecture is used for image classification application. For this MINIST dataset is used which consist of 60,000 images of handwritten digits (0-9). Each grey-scale image is having a dimension of (28, 28). Here, 50,000 images are used for training whereas 10,000 images are used for testing. In LeNet first two layers are convolutional layers and the last three are fully-connected layers. In each convolutional layer, 8 filters of dimension (5, 5) are used to compute the next layer whereas for fully-connected layer the dimension of weight matrix depends on the individual layer. This model is trained for 2 epochs by considering original weights, ternary weights via ECF & ternary weights via QCF. Accuracy for each model is calculated during testing.

After software implementation, a fully-connected layer of (128, 1) dimension & a weight matrix of (10, 128) dimension is implementated on Vivado 2018.1 by targeting Kintex-7 xc7k70tfbg484-1 FPGA. It gives 10 output neurons after processing through the design unit and the Rectified linear unit. Design unit consists of architectures such as AIM, DIM & conventional method to give different fully-connected layers. Here, simulation results are verified at 100 MHz clock frequency.

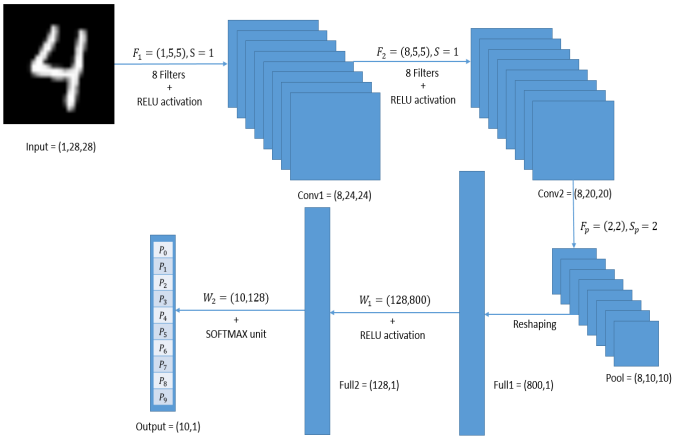

Figure 8. LeNet-5 layer architecture

A. Software implementation results

Table I shows the comparison of sparsity in the weight matrix of two fully-connected layers. In this proposed Quadratic Clip Function (QCF) generates a large number of zeros as compare to Existing Clip Function (ECF).

Table II shows the overall accuracy of LeNet model by using different methods on MNIST dataset. Here results are obtained after training CNN model for 2 epochs. In this, Ternary set via QCF gives 4% more accuracy as compare to Ternary set via ECF. It is due to increase in non-linearity provided by QCF which helps in quick learning of parameters. It can be concluded from both the table that QCF is better Clip function than ECF in terms of both sparsity & accuracy.

Table I
SPARSE COMPARISON

Method	*FC_1 Parameters* **(128,800)**	*FC_2 Parameters* **(10,128)**
Ternary set via ECF*(Zeros)*	39,159	497
Ternary set via ECF*(Sparse %)*	38.33%	38.82%
Ternary set via QCF*(Zeros)*	53,153	660
Ternary set via QCF*(Sparse %)*	51.90%	51.56%

Table II
OVERALL ACCURACY COMPARISON

Method	**MNIST**
Full precision weights	98.40%
Ternary set via ECF	89.59%
Ternary set via QCF	93.48%

1) Cost function plot: Cost function measures the loss between predicted and actual output. This loss can be minimized by updating parameters through a number of iterations. Classification applications generally use Cross-entropy loss as a cost function. It measures the performance of a CNN classifier whose output is a probabilistic value between 0 and 1. Cross-entropy loss decreases as the predicted probability converges to actual label & increases if it diverges from the actual label.
Figure 9 shows the plot of cost function v/s number of iterations for full precision, Ternary parameters via ECF & Ternary parameters via QCF respectively. Here CNN model is trained for 2 epoch. Initially cost via ECF & QCF is large as compared to cost via full precision but after few iterations, it optimizes around zero the same as in the case of full precision. It shows ternary weights give similar performance to that is given by full precision.

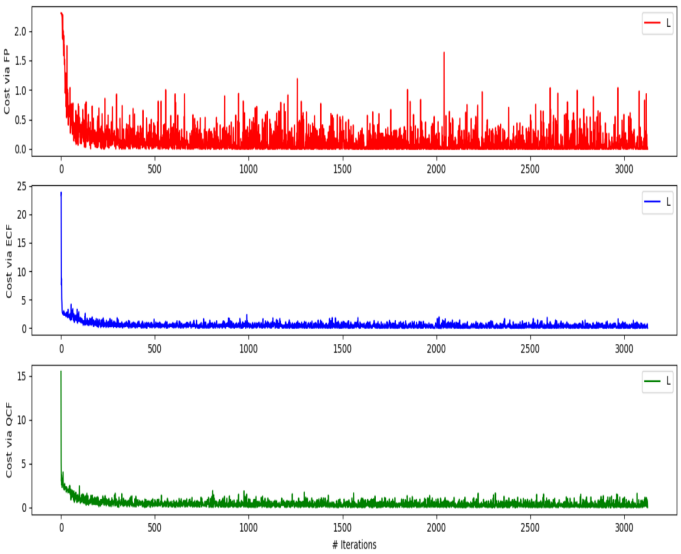

Figure 9. Cost v/s Iterations plot for different methods.

2) Digits recognition bar graph: MINIST, Modified National Institute of Standards and Technology database, consists of a dataset of images of handwritten digits from 0-9 which act as an input to CNN model. This CNN model is being trained by 50,000 images for 2 epoch. After training during inference, it is being tested by 10,000 images which predict the probability among 10 classes.
It show the prediction of handwritten digits (0-9) after inference in the bar graph. Here digits are on x-axis and probability of prediction is on the y-axis. The probability of prediction is calculated from the softmax unit which gives the probability of all the digits. Among these digits whichever digit is having the highest probability, that class of image is chosen as an input image. Now, this class is compared with the actual class of image, if it matches then digit is recognized otherwise CNN model fails to detect the image. This process is carried out for all the testing images and the corresponding probability of digits is calculated in the form of a bar graph. Digits recognition for ternary parameters via QCF is shown in figure 10.

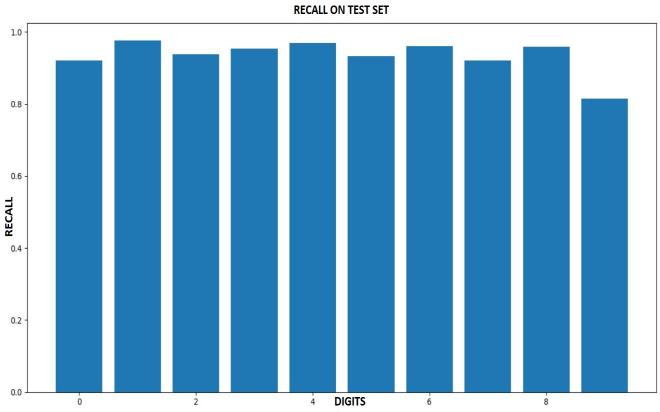

Figure 10. Accuracy of digits via Ternary weights (QCF).

B. Hardware implementation results

Table III shows the consumption of LUTs and Flip-flops by two methods such as Dual Indexing Module (DIM) & Activation Indexing Module (AIM) in the fully-connected layer. It is observed that the proposed AIM algorithm consumes less hardware as compared to other methods. This validates its usage in Fully-connected layer.

Table III
COMPARISON OF FC LAYER VIA DIFFERENT METHODS

Fully-connected layer	*LUTs*	*Flip-flops*
FC layer via DIM	610	480
FC layer via AIM	307	342

1) Cadence RTL compiler results: Below tables show the results of gate level netlist obtained after synthesis of Verilog code in Cadence RTL compiler at 180 nm technology.

Table IV
METHODS COMPARISON

Method	*Cells*	*Cell area(μm^2)*	*Total Power(mW)*
DIM	1982	67722	11.493
AIM	1568	53133	8.375

Table V
COMPARISON OF FC LAYER VIA DIFFERENT METHODS

Fully-connected layer	*Cells*	*Cell area(μm^2)*	*Total Power(mW)*	*Delay(ns)*
FC layer via DIM	93180	3150899	481.6494	9.982
FC layer via AIM	72977	2249255	305.2385	10

This gate level netlist helps in obtaining area consumption & power dissipation for different architectures. The Same is obtained for AIM & DIM method as shown in table IV. Here it can be concluded that AIM algorithm requires less number of cells, cell area, and power as compared to DIM algorithm. By considering the above results, these methods are applied in the fully-connected layer as shown in table V for which delay is also calculated. Here also, it is observed that the proposed AIM algorithm consumes less power and area as compare to DIM. As energy is equal to the product of power & delay. So the energy consumption of the FC layer via AIM is 3.05 nj which is quite less than that of the FC layer via DIM (4.807 nj). This Makes the FC layer via AIM algorithm an energy efficient architecture.

V. CONCLUSION

A Convolutional neural network requires large memory and computation resources to effectively implement on hardware. In order to meet this high requirement, this dissertation work emphasizes on the concept of Sparsity to reduce the number of parameters to be trained. As it increases the number of zero weight present in the filters. Sparsity can also be used as a regularization technique while training the CNN model, as it prevents over-fitting. Thus, in this work, it is used as an alternative to other technique. Moreover, the proposed quadratic clip function (QCF) used in the Ternary set increases the learning rate by introducing non-linearity in weights. It helps in performing effective convolution operations by considering only non-zero pairs of weight and activation element. As 52% of the weights become zero by QCF with a 93.4% accuracy in the software implementation.

Now the combination of ternary weights via QCF with proposed activation indexing module (AIM) algorithm reduces the MAC operation in the fully-connected layer to only simple accumulation. This results in saving of considerable hardware resources (LUTs by 49.6% & Flip-flops 40.3%) that would be utilized by the multipliers. The experimental results of Cadence RTL compiler also show AIM saves 1.3 nj of energy. Overall, it can be said that with little compromise in accuracy, CNN can be efficiently implemented on hardware for various applications such as Image classification, neural style transfer & human action detection, etc.

REFERENCES

[1] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," *arXiv preprint arXiv:1510.03009*, 2015.

[2] D. Kim, J. Ahn, and S. Yoo, "A novel zero weight/activation-aware hardware architecture of convolutional neural network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1462–1467.

[3] C. Jin, H. Sun, and S. Kimura, "Sparse ternary connect: Convolutional neural networks using ternarized weights with enhanced sparsity," in *Design Automation Conference (ASP-DAC), 2018 23rd Asia and South Pacific*. IEEE, 2018, pp. 190–195.

[4] C.-Y. Lin and B.-C. Lai, "Supporting compressed-sparse activations and weights on simd-like accelerator for sparse convolutional neural networks," in *Design Automation Conference (ASP-DAC), 2018 23rd Asia and South Pacific*. IEEE, 2018, pp. 105–110.

[5] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of fpga-based deep convolutional neural networks." in *ASP-DAC*, 2016, pp. 575–580.

[6] 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.

[7] J. Li, G. Yan, W. Lu, S. Jiang, S. Gong, J. Wu, and X. Li, "Ccr: A concise convolution rule for sparse neural network accelerators," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 2018, pp. 189–194.

[8] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 1–13.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[13] M. Hailesellasie and S. R. Hasan, "A fast fpga-based deep convolutional neural network using pseudo parallel memories," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[14] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 243–254.

[15] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[16] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 2016, p. 20.

[17] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2414–2423.

[18] G. Lacey, G. W. Taylor, and S. Areibi, "Deep learning on fpgas: Past, present, and future," 2016.