Hardware Accelerator Design Approach for CNN-based Low Power Applications

Govinda Rao Locharla, Revathi Pogiri

Abstract*: Field Programmable Gate Array (FPGA) based CNN accelerator is getting popular due to its high performance at lower power requirements. Since the convolution process requires the huge number of the multiply and accumulate (MAC) operations it costs more amount of area and power. In this paper, a generalized pipelined architecture for the CNN model is reported and the functionality of the key elements is quantitatively presented. This pipelined architecture employs the limited number of functional units and schedules the operation over the more number of clock cycles. This pipelined approach helps in achieving lesser hardware complexity, therefore, lesser power and area requirements at the cost of speed. The architecture presented in this paper can be customized for given CNN model by configuring Image size, Kernel sizes, Kernel buffer, pooling and activation type, etc. Finally, the hardware requirements of CNN architecture for LeNet-5 is reported as a case study and analyzed.

***Index Terms*: ASIC, CNN, FPGA, GPU, LeNet, MAC.**

I. INTRODUCTION

Convolutional neural network (CNN) modeling is a popular deep learning technique in the field of machine learning. This technique is widely adopted in wide range of applications like search engines at data centers, IoT, Robot vision, Surveillance, Embedded vision applications for different markets including the industrial, medical and automotive, … etc. [1-6]. There are various articles reported on CNNs accelerators using GPUs, FPGAs and ASICs. Among these platforms, GPUs are emerged as platform for accelerating Convolutional Neural Network due to their high performance and high parallel structure [7]. Moreover, the growth of machine learning in recent years has contributed to the emergence of GPUs. In general a GPU is composed of array of mini graphic processors and each mini processor possesses a computation unit and local cache [2], [6]. A shared high-speed bus across multiple mini processors in GPU and the high speed interconnect interface for fast data exchange makes GPU solution more power hungry. Hence hardware design for CNN in hardware accelerators would require less power and cost compared to GPUs at a marginal compromise of the computation capability [8 -19]. Though the performance of FPGA is much lower compared to GPU, power consumption is significantly low [5]. This can color FPGA based accelerator as highly suitable solution for low power applications where the speed is not a top priority. Deep learning algorithm can be optimized for minimum hardware and higher data access in FPGA based solution compared to the GPU based solution. Implementation of hardware accelerators is crucial for CNNs based IoT system applications with higher performance and power efficiency [19]. In recent years, FPGA based accelerators drawn the research interest due to its higher performance, energy efficiency, shorter development cycle, and the capability of recon configuration [1], [8 - 11]. Therefore, the hardware based acceleration of ANN processing got more popularity [12]–[14]. In order to enhance ANN learning performance, special-purpose analog circuits are adopted in [15]. FPGA based ANN designs are described in [13], [16] for fine-grain parallelization. The parallel processing based neural network computation is adopted in used in [17]. Application Specific Integrated Circuit (ASIC) chips for commercial hardware solutions are reported in [14]. In order to achieve appropriate system control and online learning, DSP processor, and the customized hardware are integrated into an FPGA [9]. Influence of bit-width optimization on learning performance for cost-effective hardware implementation is presented in [4]. Efficient hardware implementation with bitwise neural network is proposed in [18].

The rest of this paper is organized as follows. The math behind the CNN operation is quantitatively presented in Section II. Section III describes the design approach for CNN based hardware accelerator. Hardware requirements of CNN architecture for LeNet-5 are reported and analyzed as a case study in Section IV. Finally, the conclusions of this paper are presented in Section V.

II. MATHEMATICAL STUDY

A convolutional Network (Conv Net) is basically a hierarchical architecture which is inspired by biological neurons and it can be trained for various recognition, detection, and segmentation tasks. Convolution net is a feed-forward architecture that can be broadly split into two blocks: Feature extraction block and the Classification block as shown in Fig 1.

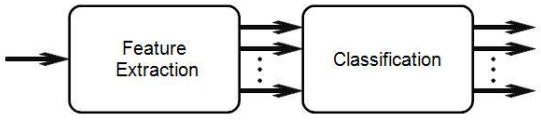

Fig 1. The Block diagram of the CNN accelerator

The feature extraction block is powered by multiple linear convolutions, pooling, and activation operations. The classification block is basically a fully connected neural network.

Revised Manuscript Received on May 06, 2019

Govinda Rao Locharla, Department of ECE, GMR Institute of Technology, Rajam-532127, AP, India.

Revathi Pogiri, Department of ECE, Sri Venkateswara College of Engineering , Etcherla, AP, India.

A.Convolution

Convolution is an operation that extracts the features of the given image. The Convolution layer uses a filter matrix over the array of image pixels and performs convolution operation to obtain the feature map. This math operation takes image matrix and a kernel matrix as inputs. In the convolution process, the matrix resulted by sliding the kernel matrix on the image matrix as shown in Fig 2. Here, the dot product computations are the elements of the resulted matrix, where the dot product is called as the 'Feature Map'. In the CNN model, the filters serve as feature detectors for the input image. Feature map values are different for different filter matrices even for the same input image.

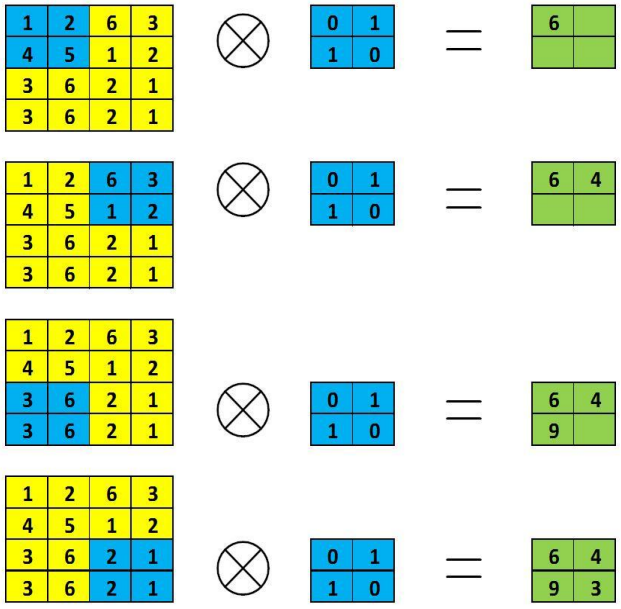

Fig 2. Convolution Process

The mathematical representation of the convolution process [19] shown above is given by the equation (1).

$$O[ofi][row][col] = b[ofi] + \sum_{n=0}^{Fi-1}\sum_{m=0}^{K-1}\sum_{o=0}^{K-1} I[n][s \times row + m][s \times col + o] \times w[ofi][n][m][o]$$

$0 \le ofi < Fo,\ 0 \le n < Fi,\ 0 \le r < R,\ 0 \le c < C$ … (1)

Here, *ofi:* Current output-feature's index number, *Fi*: Total number of the input channels and *Fo*: Total number of the output features. *row* and *col*: Current output-feature's data row and data column index; *s:* Stride size, *w:* Filter weight, *b:* Filter bias weight. *K:* Kernel size, *R:* Row size and C: column size. There are four parameters used to determine the output size of the feature map and that control the convolution procedure:

(i) Depth of the filter: Depth of the filter or kernel is an important feature for the convolution operation. Depth of a kernel or filter is same as the depth of convolution input. For example, for an input of the size (5x5x3), the depth of the filter to be used for convolution is 3. On the other hand, for 10 channels in the input image, the depth of the filters must be used is 10 which is equal to the depth of the input.

(ii) Stride: Stride value gives the amount of slide or jump of the kernel window over the input image. It is the number of pixels skipped after overlap. For example, if a filter is moved by one pixel for each overlap, the stride size will be one and if it jumps by 2 pixels then the stride size will be 2 and so on. The larger strides may result in smaller feature maps and there can be loss of feature information.

(iii) Zero-padding: Sometimes filter size may vary such that it is not fitting the input image perfectly. In such situations, it is necessary to pad the input image. Padding means adding zeros to the boundary of the image so that it fits. Dropping the part of the image where the filter does not fit is called valid padding where it keeps the valid part of the image only. This phenomenon is known as non-zero padding or a 'narrow convolution'.

(iv) Kernel size: Kernel size is also very crucial in a convolution operation since it decides the output feature map size. Consider an image matrix of dimension (I x I x C) where I is the height and width of the image and C is the number of channels and a filter of dimension (k x k x d) where k is the width and height of the filter or kernel and d can either be the number of channels same as C or it can vary for each kernel or filter. Then output size can be found as in equation (2).

$$S = I - k + 1 \quad \ldots (2)$$

B. Pooling

After a convolution layer, a pooling layer is used. The pooling layer reduces the dimensions of the image. It reduces the image size by mapping a set of pixels to a single value, which both shortens the training time and prevents from over fitting. Spatial pooling can also be called as down sampling or subsampling that reduces the feature map size and retains important information. The pooling extracts a set of neighboring pixels information from in each channel. The pooling window size determines the number of elements taken into consideration to find the maximum element. Fig 3 depicts the pooling operation on an input feature map. Spatial pooling can be categorized into two types

.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

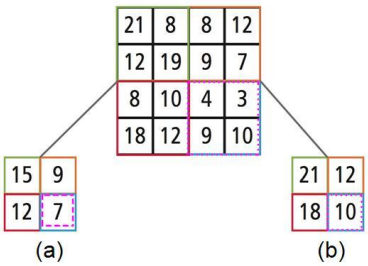

Fig 3. Pooling: (a) Average pooling (b) Max pooling

(i) Max Pooling: The most common type of pooling used is max pooling. It takes the largest number (matrix element) from the rectified feature map and rejects most of the data. Max pooling extracts

Table 1. The List of Activation Functions

S. No.	Activation Function	Description
1	Sigmoid or Logistic Activation Function	$\emptyset(z) = \frac{1}{1+e^{-z}}$
2	Tanh or Hyperbolic tangent Activation Function	$\emptyset(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
3	ReLU (Rectified Linear Unit) Activation Function	$R(z) = \max(0, z)$ If z < 0, R (z) = 0 and if z >= 0, R (z) = z
4	Gaussian Radial Basis Function	$f(x) = \sum_{i=1}^{N} w_i \varphi(\|x - x_i\|)$
5	Soft plus function	$f(x) = ln(1 + e^x)$
6	Dirac's function	$\delta(x)$
7	Unit Step Function	$f(x) = 0\ if\ x < 0$ $= 1\ if\ x > 0$
8	Truncated power function	$f(x) = log_e(1 + e^x)$

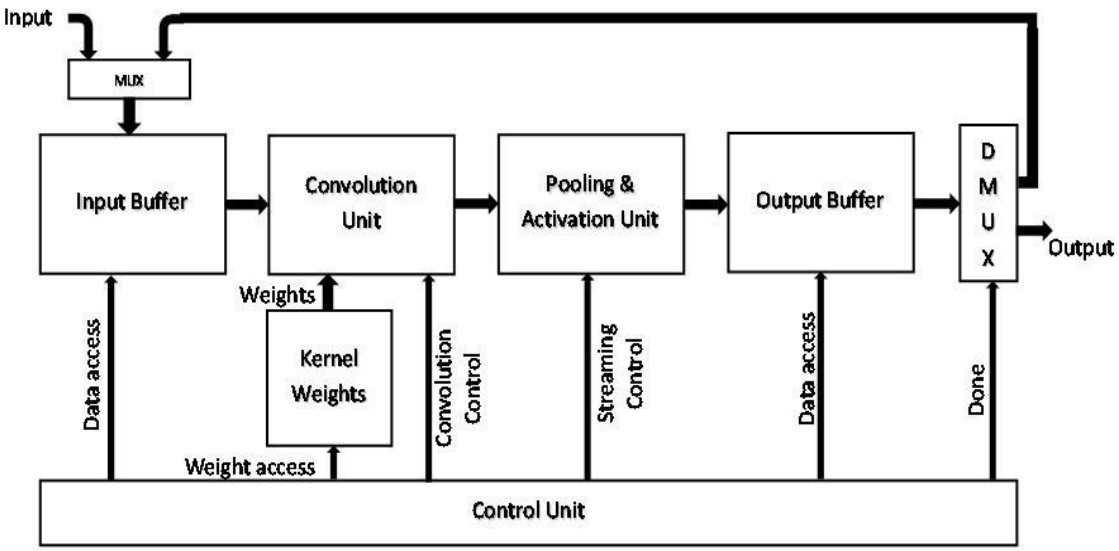

Fig 4. CNN Accelerator for feature extraction

the most important features like edges, etc. Fig 3 demonstrates the functioning of Max pooling. Here, the max pooling window size is four. Hence, the maximum element out of every four pixels is mapped to the output feature map. Hence, the output feature size is half of the input feature size.

(ii) Average Pooling: Average pooling, on the other hand, does not reject any of the input pixels and retains more information, in comparison to max pooling. It takes the average of the pixels in the pooling window in a rectified feature map. Fig 3 depicts the average pooling operation.

C. Activation

An activation function is performed in a node placed between or end of neuron net and it decides whether the neuron would fire or not. The activation function maps the resulting values between 0 and 1 or -1 and 1 etc. Activation function can also be called as transfer function. Few of the activation functions used to determine the neuron output are listed in the Table 1.

III. THE DESIGN APPROACH FOR CNN BASED HARDWARE ACCELERATOR

Generalized pipelined architecture for the CNN processor shown in Fig 4 can be customized for the given CNN model like LeNet, VGG Net, etc., and it is useful in FPGA or ASIC based implementation of the respective net model for the low power applications.

A. Input Buffer

The input image to be classified must be loaded into the input buffer before starting the process. This can be a DRAM or a register array.

B. Kernel Buffer

The filter weighs are stored in the kernel buffer and the appropriate kernel is timely fetched from the kernel buffer for the convolution.

Table 2. The Hardware Complexity for LeNet-5

Layer		Feature Map depth	Output Size	Kernel Size	Stride	Multiplications	Additions
Input Image		1	32x32	-	-	-	
C1	Convolution	6	28x28	5x5	1	25×24×6	(25×24-1)×6
S2	Pooling	6	14x14	-	-	-	
C3	Convolution	16	10x10	5x5	1	25×6×16	(25×6-1)×16
S4	Pooling	16	5x5	-	-		
F5	FC Layer-1	-	120	-	-	120×400	399×120
F6	FC Layer-2	-	84	-	-	84×120	119×84
F7	Output Layer	-	10	-	-	10×84	83×10

C. Convolution Unit

Convolution unit performs the multiplication of the image with various kernels and accumulates the products. This unit consists of one large array of processing elements (PEs) to convolve multiple kernels with the given image. For example, Convolution engine array for LeNet-5 is made of sixteen 3x3 kernel-sized convolution processors [19]. The output of each filter is gathered at the output of each PE array and feedback to the memory for next layer computation.

D. Control Unit

The control unit controls the data streaming across the architecture. It performs the input buffer addressing, Kernel buffer addressing, Convolution streaming control, Pooling & activation timing and Output buffer addressing.

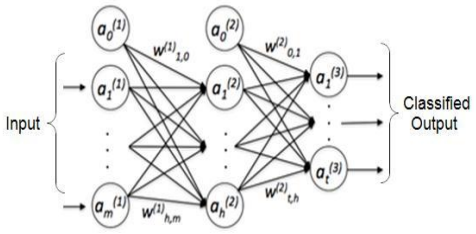

Fig 5. Fully Connected Neural Network for Classification

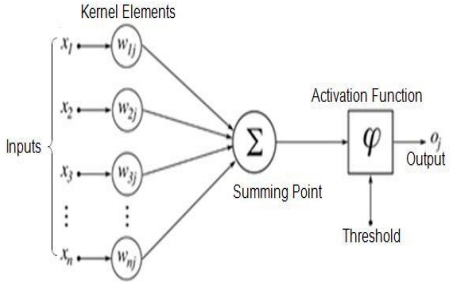

Fig 6. Activation Process

E. Output Buffer

The partially computed data and the output data are loaded into the output buffer. This also can be a DRAM or a register array.Once the convolution, pooling, and activation are done across different layers, the extracted features are fed to the classification block (fully connected net) shown in Fig 5. Here, operation at each node is illustrated in Fig 6. All the inputs are multiplied with trained weights and added, transformed by the activation function before reaching the next layer.

IV. HARDWARE COMPLEXITY AND ANALYSIS

The pipelined architecture for the feature extraction shown in Fig 4 can be configured for any CNN model by appropriately loading the kernel buffer, configuring the stride size for each layer operation, selecting pooling type, and activation function. The number of additions and multiplications required for LeNet-5 is shown in the Table 2. The number of multipliers or adders required at the convolution layer is the maximum of the number of multipliers or adders required at C1, C3, F5, F6, and F7. These numbers of additions and multiplications suggest the hardware-complexity in terms of adders and the multipliers. The hardware complexity of various CNN-architectures like VGG Net, Res Net, Alex Net, etc. can be calculated in a similar way. In order to achieve the optimum complexity of MAC units, adoption of the fixed width multipliers can be suggested.

V. CONCLUSION

In this paper, literature on hardware accelerators is briefed and math behind the CNN operation is reviewed. Generalized pipelined architecture for the CNN processor proposed in this paper can be customized for the given CNN model like LeNet, VGG Net, etc., and it is useful for FPGA or ASIC based implementation of the

respective net model for the low power applications. The hardware complexity this architecture for the LeNet-5 is analyzed in terms of additions and multipliers as a case study.

REFERENCES

1. C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun. "CNP: An FPGA-based Processor for Convolutional Networks," in proc. Field Programmable Logic and Applications, IEEE, 2009, pages 32-37.
2. Google. Improving photo search: A step across the semantic gap. http://googleresearch.blogspot.com/2013/06/improving-photo-search-step-across.html.
3. J. L. Holi and J.-N. Hwang, "Finite precision error analysis of neural network hardware implementations," IEEE Trans. Comput., vol. 42 (3), 1993, pp: 281–290.
4. H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. "An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation". In Proc. The 24th Intern. Conf. on Machine Learning-07, New York, 2007. ACM, pp: 473-480.
5. M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf. "A Massively Parallel Coprocessor for Convolutional Neural Networks" in proc. 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2009, pp: 53-60.
6. https://www.electronicspecifier.com/artificial-intelligence/cnn-hardware-accelerator-brings-improved-performance
7. S. S. L. Oskouei et al., "GPU-based Acceleration of Deep Convolutional Neural Networks on Mobile Platforms," Distrib. Parallel Clust. Comput, 2015.
8. D. Aysegul, J. Jonghoon, G. Vinayak, K. Bharadwaj, C. Alfredo, M. Berin, and C. Eugenio. "Accelerating Deep Neural Networks on Mobile Processor with Embedded Programmable Logic". In proc. NIPS 2013. IEEE.
9. S. Jung and S. S. Kim, "Hardware Implementation of a Real-time Neural Network Controller with a DSP and an FPGA for Nonlinear Systems," IEEE Trans. Ind. Informat., Feb. 2007, Vol. 54(1), pp. 265–271.
10. S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi. "A Dynamically Configurable Coprocessor for Convolutional Neural Networks" In ACM SIGARCH Computer Architecture News, volume 38, 2010, pp: 247-257.
11. M. Peemen et al.,"Memory-Centric Accelerator Design for Convolutional Neural Networks", In proc. 31st International Conference on Computer Design (ICCD), IEEE, pages 13-19.
12. C. Lindsey and T. Lindblad, "Review of Hardware Neural Networks: A Users Perspective," in Proc. 3rd Workshop Neural Netw., Sep. 1994, pp. 26–30.
13. J. Zhu and P. Sutton, "FPGA Implementations of Neural Networks—A Survey of a Decade of Progress," in Proc. 13th Int. Conf. Field-Program. Logic Appl., 2003, pp. 1062–1066.
14. F. Dias, A. Antunes, and A. Mota, "Artificial Neural Networks: A Review of Commercial Hardware," Eng. Appl. Artif. Intell., vol. 17, pp. 945–952, Dec. 2004.
15. B. E. Boser, E. Sackinger, J. Bromley, Y. Le Cun, and L. D. Jackel, "An Analog Neural Network Processor with Programmable Topology," IEEE J. Solid-State Circuits, vol. 26, no. 12, pp. 2017–2025, Dec. 1991.
16. Y. Maeda and T. Tada, "FPGA Implementation of a Pulse Density Neural Network with Learning Ability Using Simultaneous Perturbation," IEEE Trans. Neural Netw., vol. 14, no. 3, pp. 688–695, May 2003.
17. K. Oh and K. Jung, "GPU Implementation of Neural Networks," Pattern Recognit., vol. 37, pp. 1311–1314, Jun. 2004.
18. M. Kim and P. Smaragdis, "Bitwise Neural Networks," in Proc. Int. Conf. Mach. Learn., 2015, pp. 6–11.
19. Du, Li, et al. "A Reconfigurable Streaming Deep Convolutional Neural Network Accelerator for Internet of Things." IEEE Transactions on Circuits and Systems I: Regular Papers 65.1 (2018): 198-208.

AUTHORS PROFILE

Govinda Rao Locharla is working as Associate Professor in the department of Electronics and Communication Engineering, GMR Institute of Technology, Rajam, India. He received B. Tech. degree from JNTU Kakinada, M.Tech degree in VLSI System design from National Institute of Technology, Trichy, and Ph.D. degree from National Institute of Technology, Roukela, India. His research interest includes Algorithms to architecture mapping in the field of Machine learning & Signal processing and the ASIC, FPGA implementations.

Revathi Pogiri is working as Asst. Professor in the department of Electronics and Communication Engineering, Sri Venkateswara College of Engineering and Technology, Srikakulam, India. She got the M.Tech. degree in Digital Electronics and Communication Systems from JNTU Kakinada, India. She received B.Tech. degree in Electronics and Communication Engineering from JNTU Kakinada, India. Her research interest includes VLSI implementations in Machine learning applications and Signal processing.