Deep Learning Binary Neural Network on an FPGA

Yuteng Zhou, Shrutika Redkar and Xinming Huang
Department of Electrical and Computer Engineering
Worcester Polytechnic Institute, Worcester, Massachusetts 01609, USA
Email: {ytchou,svredkar,xhuang}@wpi.edu

***Abstract*—As a popular deep learning technique, convolutional neural network has been widely used in many tasks such as image classification and object recognition. Convolutional neural network exploits spatial correlations in the images by performing convolution operations in local receptive fields. Convolutional neural networks are preferred over fully connected neural networks because they have fewer weights and are easier to train. Many research works have been conducted to reduce the computational complexity and memory requirements of convolutional neural network, to make it applicable to the low-power embedded applications with limited memories. This paper presents the architecture design of convolutional neural network with binary weights and activations, also known as binary neural network, on an FPGA platform. Weights and input activations are binarized with only two values, +1 and -1. This reduces all the fixed point multiplication operations in convolutional layers and fully connected layers to 1-bit XNOR operations. The proposed design uses only on-chip memories. Furthermore, an efficient implementation of batch normalization operation is introduced.When evaluating the CIFAR-10 benchmark, the proposed FPGA design can achieve a processing rate of 332,158 images per second with with accuracy of 86.06% using 1-bit quantized weights and activations.**

I. Introduction

Deep neural network is an active research field in the past few years. It has a variety of applications such as computer vision [1], speech recognition [2], natural language processing [5], regression [15], and robotics [8]. Many neural network structures have been developed for different applications. In the area of computer vision, convolutional neural network (CNN) is one of the most dominant approaches in recent year. In Image-Net Large Scale Vision Recognition challenge (ILSVRC), CNNs are the most widely used neural networks [18], [17], [11], which can provide high accuracy on image classification tasks.

However, CNNs are computationally expensive and require much more memory resources than traditional computer vision approaches. Thus, Graphic Processing Units (GPUs) are becoming solutions [4] to implement CNNs at high speed and to handle complex models by exploiting parallelism. However, GPUs are not fit for embedded applications, because they are power hungry. As a result, many researchers investigated ways to implement deep neural networks on low power hardware [9], [10], [12]. Embedded FPGA platform is a possible solution that can be used to implement and accelerate CNNs for real time object recognition. However, state of the art CNN models are very complex. They require many computational resources and a large memory to store millions of high precision weights. FPGA has a limited on-chip memory size. When external memory is used, memory bandwidth becomes a serious issue affecting throughput of CNN models. Courbariaux et. al proposed the Binarized Neural Network (BNN) [7], in which all weights and input values of each layer are quantized to +1 or -1. In CNN, most of the heavy computations are floating-point multiplications of weights by inputs during both training and testing. Binarized neural network replaces all multiply-accumulate operations by simple XNOR-addition operations. As shown in [7], the classification performance of BNN is close to CNN when evaluated on MNIST, CIFAR-10 and SVHN. Using GPU as an accelerator for training and testing CNN model can reduce computational time significantly [13]. However, power consumption of a GPU is too high for many real-time embedded applications.

This paper modifies binarized neural network to be hardware compatible and implements it on Xilinx FPGA. The system is targeted on Xilinx Zynq ZC706 and also on Virtex 7 FPGA. It supports the rate of 60 frames per seconds. The rest of the paper is organized as follows. In Section II, background of CNN is presented. Section III discusses binary neural network forward propagation and backward propagation flow. In Section IV, proposed hardware architecture of binary neural network is presented. Results of the FPGA implementation are presented in Section V, followed by conclusion sin Section VI.

II. Convolutional Neural Network

Unlike other neural networks, CNN takes advantage of the fact that input is usually an image such that CNN arranges its neurons in three dimensions corresponding to width, height and depth. The first layer reads an input image and outputs a sequence of feature maps. Number of feature maps is equal to number of filters. The next layer inputs the feature maps generated by previous the layer and gives another sequence of feature maps. Final layer reduces feature maps to single vector of class scores, with highest score referring to the class of input image.

Convolutional layer requires most of the computational resources in CNN. It receives input feature maps and weights of that layer and then performs 3D convolution operation, which can be described mathematically as in (1).

$$Y[n,i,j] = \sum_{d=0}^{D-1} \sum_{y=0}^{K-1} \sum_{x=0}^{K-1} W[n,d,2-x,2-y] * X[d,i+x,j+y] \quad (1)$$

In this expression, input feature map is of size $D \times W \times H$ and output feature map is of size $N \times W \times H$, where N is

978-1-5090-6389-5/17/$31.00 ©2017 IEEE

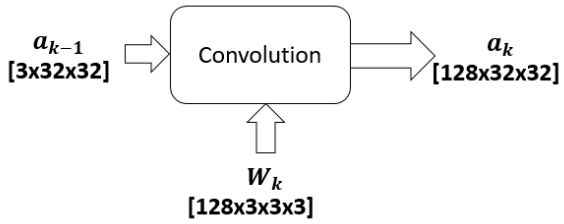

Figure 1. Illustration of the first convolution layer

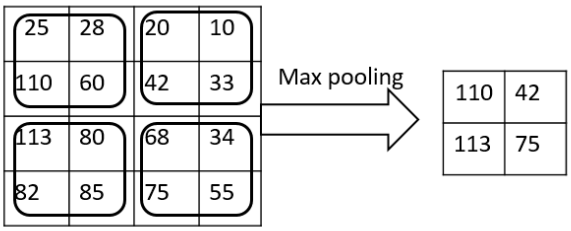

Figure 2. An example of pooling layer operations

number of feature maps. $K \times K$ is the kernel size. Above expression gives (i, j) value of n^{th} feature map. For example, in our first layer implementation, input feature map is an image of size $3 \times 32 \times 32$ and the kernel is of size 3×3 and there are 128 weights of size 3×3 and depth 3, which then generates output feature map of size $128 \times 32 \times 32$. This convolutional operation is shown in Fig. 1.

Between successive convolution layer, often there is a pooling layerIt down samples input data, which reduces computation as well as controls over fitting. Down sampling the result also provides a form of distortion and scale invariance. Fig. 2 shows operation of pooling layer. In our implementation, we used max pooling layer.

Batch normalization solves the problem of internal covariance shift by normalizing inputs at the beginning of training [14]. It allows higher learning rates and takes care of bad initializations. In our implementation, batch normalization is inserted after every convolutional and fully connected layer and before applying non-linearity. Operations of batch normalization [14] can be described in (2) .

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta \tag{2}$$

Here x is the input data , y is the output data, β is the learnable bias, γ is the learnable scale factor, μ is the running mean, δ is the running average factor. During training, μ and σ^2 are mean and variance of current input mini batch x, and during testing they are replaced by average statistical mean and variance over training data. Batch normalization layer forces activations throughout the network to take a unit Gaussian distribution and accelerates the training and also reduces overall impact of weight's scale.

In addition, there is a fully connected layer that has full connections to all the activations in the previous layer. Activations of fully connected layer are computed by matrix multiplication followed by applying non-linearity.

III. Binarized Neural Network

In the recent work of Courbariaux et. al [6], [7], they proposed the binarized convolutional neural network, where binary weights and activations were employed. During testing, they used binary weights and activation to generate class scores, while during training they used binary weights and activation to compute parameter gradients. But training process requires real-valued weight, since Stochastic Gradient Descent (SGD) during back propagation requires high-precision weights. SGD performs a large amount of small changes in small steps and then noise is averaged out to obtain an updated real-valued weight at the end of back propagation.

For binarizing real valued weights and activations, Courbariaux et. al [7] used bpth deterministic and stochastic binarization functions. Although stochastic binarization is more correct and precise quantization process, it requires hardware to generate random bits while quantizing. On the other hand, deterministic binarization function is straightforward to implement on hardware. Hence, in our implementation, we adopt the deterministic binarization process as described in (3).

$$x^b = sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & otherwise \end{cases} \tag{3}$$

where x^b is the binarized approximation of real valued variable x. However, when sign function is used to binarize weights and activations in forward propagation, it causes problem in back propagation, since derivative of sign function is zero everywhere. Hence, all the incoming gradients to sign function would be multiplied by zero. This is harmful in network learning process. In order to solve this problem, straight through estimator is used during backpropagation [3]. Consider sign function as $y = sign(x)$. Then, straight through estimator makes backpropagation through sign function by treating derivative of sign function as an identity function. If g_y is a computed gradient with respect to y, then gradient with respect to x, g_x is computed as in (4).

$$g_x = \begin{cases} g_y & |x| \leq 1 \\ 0 & otherwise \end{cases} \tag{4}$$

This cancels the gradient, only when x is outside of -1 to +1 range. Straight through estimator has proven to give a good results [3] and it is simpler to implement in the hardware. In our implementation, we trained the binary neural network using GPU. Pre-trained weights from training process are used to implement BNN hardware realization on FPGA.

IV. Hardware Architecture of BNN

In BNN hardware realization, weights are used only after binarization for convolutional and fully connected layers. Hence, pretrained weights from training are binarized and then stored in registers or BRAMs of FPGA. This step saves significant amount of memory. Since now, instead of 8 bit/16 bit weights, 1 bit weights are stored into on-chip memory of

FPGA, memory usage reduces by 8/16 times. Kernel size for convolution layer is chosen to be 3x3 [16].

In layer 1 of BNN, input CIFAR-10 RGB image is of dimensions $3\times32\times32$ and each pixel is of 8 bit precision. Since weights to this layer are binarized, the convolution operation in (1) can be rewritten as in (5), where $W_b = sign(W)$.

$$Y[n,i,j] = \sum_{d=0}^{D-1}\sum_{y=0}^{K-1}\sum_{x=0}^{K-1} W_b[n,d,2-x,2-y]*X[d,i+x,j+y] \tag{5}$$

Architecture realization of first layer is as shown in Fig. 3. This figure shows computation of N feature maps when N kernels of size $3 \times 3 \times 3$ interacts with RGB input image. Number of output feature maps are equal to number of kernels. Since W_b can only take two values +1 or -1, multiply-accumulation operations of convolution changes to series of additions (when W_b is +1) and series of subtractions (when W_b is -1) as shown in Fig. 4.

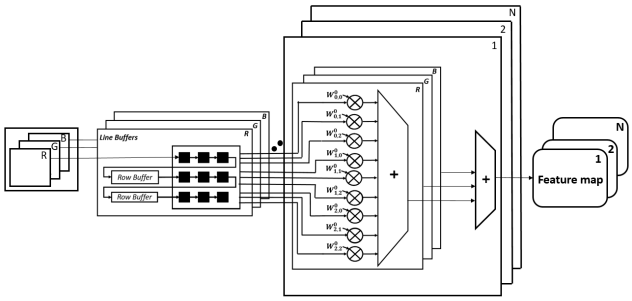

Figure 3. Hardware architecture of the first convolution layer

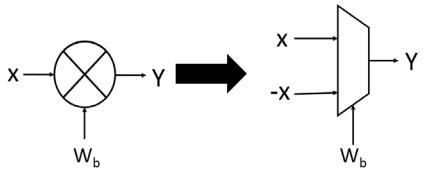

Figure 4. Multiply-accumulate operation implemented as addition/subtraction in the first layer

All other convolution layers except for the first layer have binarized weights as well as binarized input activations. Hence, the realization of all other convolution layer is same as shown in Fig. 5. This figure shows computation of N output feature maps when input feature maps and N number of 3D kernels are applied to convolutional layer. In this realization, all the multiply operations in convolutional layer are replaced by XNOR operations. Since 2D convolution operation occupies 90% of computation cost in CNN, replacing most of the multipliers by XNOR gate reduces complexity of computations by significant amount. In other words, replacing most of the 32-bit floating point multiply accumulations by 1 bit XNOR count operations reduces hardware requirements by about 32 times.

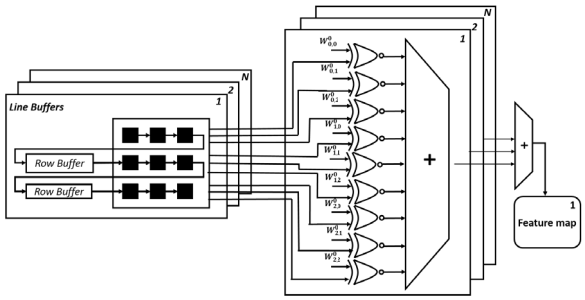

Figure 5. The 2nd, 3rd and 4th convolution layer realization

For fully connected layers, inputs are binary feature maps from previous layers and binary weights stored in the registers. Again, fully connected layers in CNN consist of mainly multiplication operations. Hence, all the multiplication operations in fully connected layer are also replaced by XNOR operations. The usual problem faced with fully connected layer in any CNN model is a large number of weights, which require large memory for storage. However, in our implementation, since weights are binary, they are easily stored into on-chip memory of FPGA. External memory is not required. Fully connected layer operation is implemented as shown in Fig. 6. This figure shows computation of one output class score out of $1 \times N$ classes.

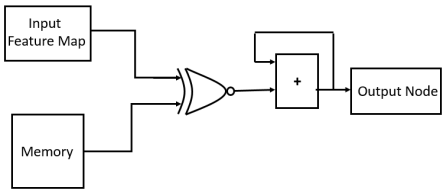

Figure 6. Fully connected layer realization

Realization of batch normalization requires four fixed-point stored parameters, $\mu, \sigma, \gamma, \beta$, for each of the input activation layer, according to equation (2). However, in our implementation, we have reduced it to two stored parameter (one fixed point parameter and one binary parameter), for each input activation layer. This is possible due to shifting and scaling of expression (2) to (6), as follows.

$$y = q_b(x + p) \tag{6}$$

$$where\, p = \left(\frac{\beta\sigma}{\gamma} - \mu\right),\ q = \frac{\gamma}{\sigma}\, and\, q_b = sign(q)$$

After every batch normalization operation, sign function is applied to quantize input to the value of +1 or -1. We have taken advantage of this, by neglecting magnitude of scaling function $\frac{\gamma}{\beta}$ and only storing sign of scaling function as a binary parameter. Because, only sign of the scaling function will have a contribution in quantizing final batch normalized output. In our implementation, we have batch normalization layer after

every convolutional and fully connected layer. By using this shift and scale operation, now memory requirement for storing batch normalization parameter reduces from 32 bits (8 bits per parameter * 4 parameters) to 9 bits (8 bit per parameter * 1 + 1 bit sign), for each activation layer.

V. FPGA Implementation Results

At first, We implement the architecture with 2 convolutional and 2 dense layer on Zynq ZC706 FPGA platform. It has very low resource usage but also yield a low accuracy of 66.63%. Subsequently, we increase the number of convolutional layers to 5, which improves the accuracy to 86.06%. The improved BNN requires a much larger FPGA such as Virtex-7 980T platform. Synthesis results are obtained from Xilinx Vivado 2013.4. The comparison between 2 architectures is listed in Table I. With this resource utilization on Virtex-7 980T FPGA, the system can run at a maximum frequency of $340.13\,MHz$. Total processing latency of each frame is 1670 clock cycles, which is about $4.9\mu s$. The definition of latency is the number of clock cycles from the first pixel entering into the input image patch to the classification result produced at the output node. Since the input image patch size is 32-by-32 in this work, the maximum throughput of the proposed design can reach $10^9/32/32 \times 340.13/1000 = 332,158$ images per second with a constant latency of $4.9\mu s$.

Table I shows that with 5 convolutional layers, our BNN can reach an accuracy of 86.06% on CIFAR-10 dataset. For a small FPGA, the BNN with only 2 convolutional layers can be considered.

Table I
Comparison of FPGA Implementation Results

Comparisons	Architecture 1	Architecture 2
Platform	Zynq ZC706	Virtex-7 980T
slice LUTs usage	9.27%	91%
slice registers usage	5.33%	82%
Accuracy	66.63%	86.06%
convolutional layers	2	5
fully connected layers	2	2

VI. Conclusion

In this paper, we present the architecture and FPGA implementations of binary neural network. Binary neural network is a simplified convolutional neural network with binary weights and activations. Thus the entire design can be implemented using on-chip memories only. By avoiding the latency and bandwidth limitation of an external memory, the proposed hardware design can provide very high throughput with efficient resource usage. The BNN model is trained and evaluated using CIFAR-10 dataset. With 5 convolutional layers, it can achieve an accuracy of 86.06%. The FPGA implementation has tge maximum throughput of 332,158 images per second, each with a constant latency of only $4.9\mu s$. The proposed design is particularly suitable for low-power embedded system applications with limited memory.

References

[1] Richard Andrew, Hartley and Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[2] Ann, Wei-Ning Glass, Hsu, Yu Zhang, and James Lee. Exploiting depth and highway connections in convolutional recurrent deep neural networks for speech recognition. *cell*, 50:1, 2016.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] Adam Coates, Paul Baumstarck, Quoc Le, and Andrew Y Ng. Scalable learning for object detection with gpu hardware. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4287–4293. IEEE, 2009.

[5] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3123–3131, 2015.

[7] Matthieu Courbariaux, Itay Hubara, COM Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training neural networks with weights and activations constrained to+ 1 or-.

[8] Pomerleau Dean A. *Neural network perception for mobile robot guidance*, volume 239. Springer Science & Business Media, 2012.

[9] Erjin, Jiantao Yu, Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Jincheng Zhou, Tianqi Tang, Ningyi Xu, Sen Song, et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 26–35. ACM, 2016.

[10] Aysegul Eugenio, Dundar, Jonghoon Jin, Vinayak Gokhale, Berin Martini, and Culurciello. Memory access optimized routing scheme for deep networks on a mobile coprocessor. In *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*, pages 1–6. IEEE, 2014.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[12] Maurice Henk, Peemen, Arnaud AA Setio, Bart Mesman, and Corporaal. Memory-centric accelerator design for convolutional neural networks. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 13–19. IEEE, 2013.

[13] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4107–4115. Curran Associates, Inc., 2016.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[15] Su-Mei Lin. Analysis of service satisfaction in web auction logistics service using a combination of fruit fly optimization algorithm and general regression neural network. *Neural Computing and Applications*, 22(3-4):783–791, 2013.

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[17] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[18] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.