RNA: An Accurate Residual Network Accelerator for Quantized and Reconstructed Deep Neural Networks

Cheng Luo[1], Yuhua Wang[1], Wei Cao*[1], Philip H.W. Leong[2], Lingli Wang[1]
[1]State Key Laboratory of ASIC and System, Fudan University
[2]School of Electrical and Information Engineering, University of Sydney
*caow@fudan.edu.cn

***Abstract*—With the continuous refinement of Deep Neural Networks (DNNs), a series of deep and complex networks such as Residual Networks (ResNets) show impressive prediction accuracy in image classification tasks. Unfortunately, the structural complexity and computational cost of residual networks make hardware implementation difficult. In this paper, we present the quantized and reconstructed deep neural network (QR-DNN) technique, which first inserts batch normalization (BN) layers in the network during training, and later removes them to facilitate efficient hardware implementation. Moreover, an accurate and efficient residual network accelerator (RNA) is presented based on QR-DNN with batch-normalization-free structures and weights represented in a logarithmic number system. RNA employs a systolic array architecture to perform shift-and-accumulate operations instead of multiplication operations. QR-DNN is shown to achieve a $1\% \sim 2\%$ improvement in accuracy over existing techniques, and RNA over previous best fixed-point accelerators. An FPGA implementation on a Xilinx Zynq XC7Z045 device achieves 804.03 GOPS, 104.15 FPS and 91.41% top-5 accuracy for the ResNet-50 benchmark, and state-of-the-art results are also reported for AlexNet and VGG.**

I. Introduction

In the recent years, Deep Neural Networks (DNNs) have made impressive breakthroughs in image classification and object detection. In the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2015, Kaiming He et al. proposed Residual Networks [1], or ResNet, with a top-5 prediction accuracy of 96.4%, surpassing human-level performance.

However, ResNets employ an extremely deep network architecture with high computational complexity and large memory requirements, making hardware acceleration an important research topic. Moreover, ResNets consist of many residual units including bottleneck architectures [1] as well as residual connections [1] and adopting batch normalization layers [2]. These units are critical for accuracy but their efficient hardware implementation offers new challenges for researchers.

To address this problem, extensive quantization methods are proposed to quantize neural networks with low-precision weights and activations, which drastically reduce the computational cost and memory requirement with little loss in accuracy. However, most of them are technically difficult for the complex residual units. The main disadvantage of those methods is that they either avoid handling batch normalization layers at the expense of increasing the difficulty of hardware implementation [3, 4], or cause significant reduction in accuracy by quantizing the batch normalization layers directly [5].

We propose a hardware-software co-design technique targeted for lowering the bandwidth/memory requirements and improving network accuracy. From a software perspective, a comprehensive quantization and reconstruction optimization algorithm (QR-DNN) [1] is designed to generate quantized and reconstructed networks in the training phase without significant reduction in accuracy. Furthermore, we present RNA, a residual network accelerator consisting of a systolic array architecture with dedicated processing elements as well as special processing units, performing real-time processing on FPGA platforms with memory access and hardware resource constraints. Specifically, this paper makes the following contributions:

- The quantized and reconstructed (QR-DNN) training technique which introduces and removes batch normalization layers to control precision requirements in low-precision models without changing the network structure.
- A high-throughput systolic, multiplier-free DNN accelerator with efficient convolutional flow and dedicated units, supporting bottleneck architectures and residual connections.
- Combining all the contributions above, an implementation and performance analysis of our residual network accelerator (RNA) on a Xilinx Zynq XC7Z045 FPGA for well-known network models including AlexNet [6], VGG-16 [7] and ResNet-50, which achieves 687.8 / 878.1 / 804.0 GOPS and 82.14% / 89.81% / 91.41% of Top-5 accuracy. To the best of our knowledge, this is the most accurate 4-bit DNN accelerator reported to date.

II. Residual units

Residual units consist of bottleneck architectures, residual connections and batch normalization layers:

- *Bottleneck Architecture*: The bottleneck architecture in ResNets employs convolution layers (1×1, 3×3, 1×1), where the first and last 1×1 layers are responsible for reducing and restoring dimensions, leaving the 3×3 layer as a bottleneck with smaller input/output dimensions.
- *Batch Normalization Layer*: The batch normalization layer (BN) standardizes input activations with their means μ_b and variances σ_b^2, then the batch-normalized results are further scaled and shifted with parameters (γ, β).
- *Residual Connection*: The residual connection is employed between the input and output layers of bottleneck architecture. It recasts the original function $y = H(x)$ to a residual function $y = F(x) + x$ to address the degradation problem: the increasing network depth degrade accuracy.

[1]Source code is available at https://github.com/wdlctc/QR-DNN.

III. QR-DNN Optimization Algorithm

In this section, we propose a novel batch-normalization focused reconstruction technique to improve network accuracy and optimize network structure. Also, we take the advantage of logarithmic quantization to reduce the computational complexity and the bandwidth/memory requirements.

A. Reconstruction technique

Our reconstruction technique consists of four main phases including Insertion phase, Replacement phase, Quantization phase and Combination phase as described in follows.

Phase 1 Insertion: This phase inserts a batch normalization layer after each CONV layer as Equation (1) to generate a batch-normalization-variant network. Then we train the network to generate a highly accurate network model. Specially, network models that already contain the batch normalization layers (ex:ResNets) directly skip this phase.

$$\begin{aligned} CONV->y &= (Wx+B) \\ BN->y &= \frac{x-\mu_b}{\sqrt{\sigma_b^2+\epsilon}}\gamma+\beta \\ CONV+BN->y &= \frac{(Wx+B)-\mu_b}{\sqrt{\sigma_b^2+\epsilon}}\gamma+\beta \end{aligned} \tag{1}$$

Phase 2 Replacement: Once the training process is completed or the input network is a batch-normalized network, the batch normalization layers are removed and replaced by equivalent scale layers with parameters $(\grave{\gamma}, \grave{\beta})$ to scale and shift input activations. Equation (2) explains the equivalence replacement operation in this phase, making the resulting network mathematically equivalent to the original network.

$$\begin{aligned} y &= \frac{(Wx+B)-\mu_b}{\sqrt{\sigma_b^2+\epsilon}}\gamma+\beta \\ y &= (Wx+B)\grave{\gamma}+\grave{\beta} \\ \Rightarrow \grave{\gamma} &= \frac{\gamma}{\sqrt{\sigma_b^2+\epsilon}}, \quad \grave{\beta} = \beta - \frac{\mu_b}{\sqrt{\sigma_b^2+\epsilon}} \end{aligned} \tag{2}$$

Phase 3 Quantization: For hardware efficiency and classification accuracy, a state-of-the-art logarithmic quantization method is introduced in the quantization phase, whose details are described in the next subsection. In addition, this part can also be simply replaced with other quantization methods (ex: Ristretto[3]) and improve their classification accuracy.

Phase 4 Combination: Once the quantization process is completed, CONV layers absorb the following scale layers using Equation (3). Then, we quantize all activations and biases into 8-bit fixed-point numbers. Consequently, a quantized and reconstructed DNN (QR-DNN) is generated, which achieves the desired accuracy with optimized structures.

$$\begin{aligned} y &= (\overline{W}x+B)\overline{\gamma}+\grave{\beta} \\ y &= (\overline{\overline{W}}x+\overline{\overline{B}}) \\ \Rightarrow \overline{\overline{W}} &= \overline{W}\times\overline{\gamma}, \quad \overline{\overline{B}} = quantize(B\times\overline{\gamma}+\grave{\beta}) \end{aligned} \tag{3}$$

B. Quantization methods

The quantization methods used in the quantization phase is a modified version of the standard logarithmic quantization and fixed-point quantization. The values represented by the logarithmic representation and fixed-point representation can be written as follows:

$$\begin{aligned} N_{log} &= (-1)^s \times 2^{exp-exp_{min}} \\ N_{fixed} &= (-1)^s \times 2^{-fl} \times mantissa \end{aligned} \tag{4}$$

where exp is the number with logaritmic quantization, $mantissa$ is the fixed representation, and exp_{min} and fl defines the range.

These methods converts 32-bit floating weights and activation to 4-bit logarithmic representation and 8-bit fixed-point representation. Especially, we quantize the minimum set of parameters to zero values represented by special numbers "Zero" and treat negative zero numbers as a special trigger operand "Tig" for hardware purposes.

Moreover, a special strategy is presented for suppressing accuracy reduction caused by quantization. This technique first averages and quantizes the scale parameters $\grave{\gamma}$ to a unique quantized parameter $\overline{\gamma}$, then compensates resulting accuracy reduction by fine-tuning the CONV parameters. As a result, the combination phase can directly multiply convolutional layer parameters by a power-of-two constant.

Then, an incremental network quantization method proposed by [4], which can simultaneously realize logarithmic quantization and avoid accuracy reduction, is chosen to quantize CONV weights. This method divides the large and small weights of CONV into two separate groups, then iteratively quantizes the larger group and retrains the smaller group. Experiments show that the retraining method can significantly minimize accuracy reduction caused by the logarithmic quantization.

Finally, all parameters and activations are quantized into low precision with a slight loss in accuracy, and the original complex 32×32 floating-point multiplications are replaced by 8×4 shifting operations. Furthermore, by introducing zero values and trigger operands into quantized values, the hardware performance is further improved, and it is able to support special hardware operations.

IV. Hardware Accelerator Design

The QR-DNN optimization algorithm effectively reduces the structural complexity and the computational cost, which allows us to implement an accurate and efficient residual network accelerator (RNA). Fig.1 (a) illustrates the architecture overview of RNA, where a systolic array in the red box and a global controller together with there on-chip memories in the blue box. Each block of systolic array architecture has a processing element (PE) to perform shift-and-accumulate operations as shown in Fig.1 (b). Also, Fig.1 (c) shows an example of post-processing unit including various blocks for various functions.

A. Systolic Array Flow

A systolic array architecture is employed to minimize off-chip bandwidth and maximize on-chip utilization for the convolution operation. The convolution operation can be viewed as a sequence combining two input matrices, I and W, in a dedicated flow to generate the result matrix O, which can be efficiently handled as follows. Suppose the input activation matrix I has dimensions of $M^2 \times N_{in}$ and the weight matrix W has dimensions of $N_{in} \times N_{out} \times k^2$. Here M, k, N_{in} and N_{out}

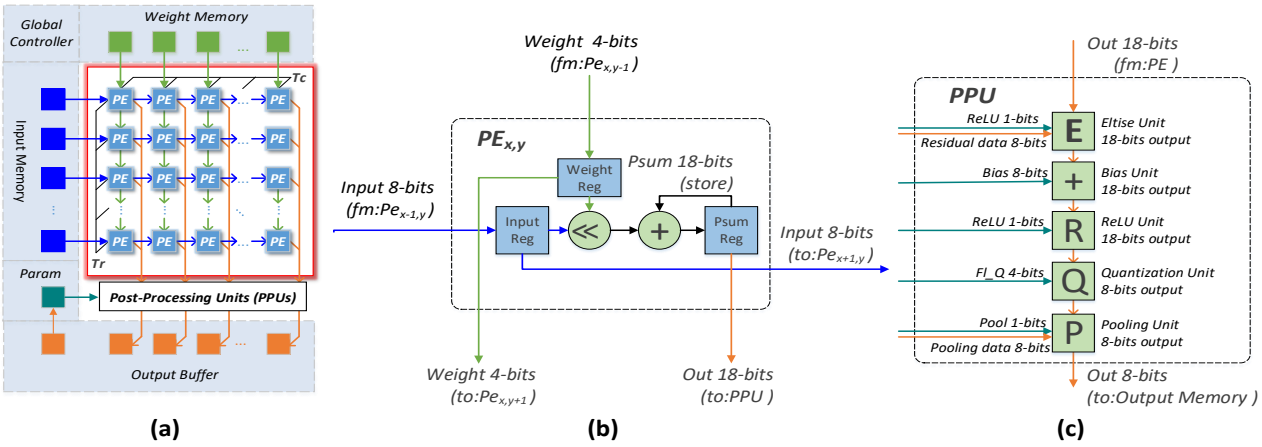

Figure 1. Residual Network Accelerator; (a) Architecture Overview; (b) Processing Element (PE); (c) Post-Processing Unit (PPU)

represent the input feature map size, the convolution kernel size, the input dimensions and output dimensions, respectively. Initially, $T_r \times N_{in}$ elements of matrix I and $N_{in} \times T_c \times k^2$ elements of matrix W are brought into the systolic array to compute $T_r \times T_c$ elements of matrix O, where (T_r, T_c) represent the number of rows and columns of systolic array. Finally, special operands "Tig" are activated to trigger the output and reset the accumulator. This flow continues until all M^2 columns of matrix I have been shifted with all N_{out} rows of matrix W. For a convolution operation, it takes $(M^2/\ T_r) \times (N_{out}/\ T_c)$ iterations to produce all elements of result matrix O.

The convolutional flow allows for efficient use of the convolution operation which completely overlap I/O and computation times in order for high throughput and real-time responses. The blocking and interleaving methodology proposed by [8] is introduced to minimize the required bandwidth and maximize data reuse. Moreover, this flow can effectively supports bottleneck architecture with different convolution kernel sizes.

B. Processing Element

Fig. 1 (b) shows the architecture of a processing element unit $PE_{x,y}$. It receives a 4-bit weight from top $PE_{x,y-1}$ then passes it down to $PE_{x,y+1}$, and receives an 8-bit activation from left $PE_{x-1,y}$ then passes it right to $PE_{x+1,y}$. The PE consists of a shifter and an accumulator to produce a 18-bit shifting result of the weight and activation and accumulate it in the $Psum$ register. Once the shift-and-accumulate operations are completed, this PE is supposed to receive two trigger operands "Tig" associated with the weight and activation to trigger the output and reset the accumulator register.

The bit-width of Psum after shift-and-accumulate operations is much larger (typically 18 bits to avoid overflow) than the input activation and weight (8 bits and 4 bits, respectively). In this case, our dedicated convolutional flow moves the smaller bit-width activations and weights while holding the larger bit-width results inside PE. This strategy can effectively reduce the power consumption as well as design area, and minimize the output bandwidth by generating 8-bit results instead of a 16-bit Psums. Since the minimum parts of weights and activations are quantized as zero, the shift-and-accumulate operation is suspended when any PE receives a "Zero" value from either a weight path or an activation path.

C. Post-Processing Unit

Before being written into the output buffer, results must be trimmed through the post-processing units. Fig. 1(c) illustrates the blocks inside the post-processing units namely **the Bias, Eltwise, ReLU, Quantization and Pooling**, which are responsible for the corresponding layers.

The Eltwise units are responsible for the residual connections. Here we followed the suggestions of [5] to sequentially compute the layers in the residual connection first and then the layers in the bottleneck architectures. The Bias units are used to add bias parameters to the output results. The ReLU units convert negative values into zero. The quantization units are implemented to quantize 18-bit results into 8-bit output activations by rounding down. The Pooling units are comprised of comparators and accumulators, which are associated with the max-pooling function and the average-pooling function.

V. Experimental Results

We evaluate our designs in terms of accuracy and efficiency. To measure accuracy, the QR-DNN optimization algorithm is run on ImageNet-2012 datasets using the well-known DNN models including ALexNet, VGG-16 and ResNet-50. For fair comparison, all pre-trained floating-point models are downloaded from Caffe Model Zoo. We use one NVIDIA GeForce GTX TITAN X GPU platform and Caffe deep learning framework [9] to perform our algorithm.

Table II summarizes the inference accuracy of different DNNs obtained by various quantization methods. Here the pre-trained floating-point network references (ref) are downloaded from Model Zoo. The fixed-point networks (fixed) are generated by the modified Ristretto tool [3] integrated with our reconstruction method. The logarithmic networks (QR) are created by our QR-DNN optimization algorithm.

Importantly, Table II reveals that our optimization algorithm can improve the accuracy of AlexNet and VGG by $1\% \sim 2\%$ above the baseline references, and maintain the accuracy reduction of residual network within 1%. Compared with the fixed-point quantization, the logarithmic quantizaion causes accuracy loss within 0.5%, in return of halved weight bit-width and optimized hardware implementation. Compared to the symmetric quantization (SYQ) technique [14], our results for AlexNet, VGG-16, and ResNet-50 show an improvement

Table I
COMPARISON OF DIFFERENT DNN ACCELERATORS WITH VARIOUS DNN MODELS

Accel-erator	Platform	Logic	DSPs	RAMs	Power (W)	Model	Performance (GOPS)	Throughput (FPS)	Accuracy (Top-5)	Bit-width (in,w)
FINN [10] FPGA'17	Zynq XC7Z045	46.2K LUTs	0	186 (36K)	11.7	AlexNet	2465.5	21.9K	80.1%	1/1
Jiao et al [11] FPL'17	Zynq XC7Z020	44K LUTs	89	105.5 (36K)	2.26	DoReFa-Net	410.22	106	73.1%	2/2
Qiu et al [12] FPGA'16	Zynq XC7Z045	183K LUTs	780	486 (36K)	9.6	VGG-16	136.97	4.45	86.66%	16/16
Yone et al [13] PDPS'17	Zynq UltraScale+ MPSoC ZU9EG	192K LUTs	4	683.5 (36K)	22	VGG-16	921.60	31.8	82.25%	1/1
Moss et al [8] FPGA'18	Altera Arria 10	1,150K LEs	1280	-	48	VGGNET	31180	114	-	1/1
Ma et al [5] ISCAS'17	Altera Arria 10	128K ALMs	1046	2167 (20K)	-	ResNet-50	285.07	36.81	90.40%	16/16
RNA	Zynq XC7Z045	203K LUTs	0	443 (36K)	10.56	AlexNet	687.78	270.75	82.14%	8/4
						VGG-16	878.11	30.19	89.81%	8/4
						ResNet-50	804.03	104.15	91.41%	8/4

Table II
TRAINING TIME AND ACCURACY OF REFERENCE FLOATING-POINT MODELS, FIXED VARIANTS AND QR-DNNS.

Network	Width (in,w)	Accuracy Top-1	Accuracy Top-5	Extra time
AlexNet ref	32/32	57.24%	80.23%	0
AlexNet fixed	8/8	60.54%	82.54%	$<$ 1m
AlexNet QR	8/4	**60.34%**	**82.14%**	**48h**
VGG-16 ref	32/32	68.54%	88.65%	0
VGG-16 fixed	8/8	71.12%	90.30%	$<$ 1m
VGG-16 QR	8/4	**70.19%**	**89.81%**	**144h**
ResNet-50 ref	32/32	75.18%	92.16%	0
ResNet-50 fixed	8/8	74.49%	91.87%	$<$ 1m
ResNet-50 QR	8/4	**73.95%**	**91.41%**	**86h**

of 1.3%, 1.3% and 0.5% respectively. One limitation of our algorithm is the duration of several days' training time. But it is worthwhile to exchange accuracy gains with training times which is performance-independent with hardware accelerators.

For the hardware evaluation, we use Verilog-HDL language to implement the residual network accelerator (RNA) on ZC706 evaluation board, which consists of a Xilinx XC7Z045 FPGA, dual ARM Cortex-A9 cores and 1 GB DDR3 memory with a peak bandwidth of 4.3GB/s. We evaluate the hardware performance at 150 MHz with zero timing slack.

The hardware metrics of the residual network accelerator (RNA) are shown in Table I, which consists of 203K logic LUTs and 443 36Kb BRAMs. The accelerator realizes 804.03 FPS for ResNet-50 model. For AlexNet and VGG, we set $batch\_size = 8$ to improve the performance of full connection layer, which realizes 687.78/878.11 GOPS for AlexNet/VGG-16 models. Here our performance is constrainted by bandwidth limitations, causing the performances of AlexNet and ResNet degrade 190/74 GOPS compared with VGG-16.

Moreover, comparisons between our design and other FPGA accelerators on various matrices are performed. In comparison to references [5, 12], our accelerator with 4-bit logarithmic parameters outperforms the 16-bit fixed-point accelerators in terms of performance and accuracy, proving that our acceleration can effectively accelerate the residual networks. Some high throughput accelerators from [10] and [8], performing binarized neural network, outperform our accelerator in performance but cause significant accuracy reduction. Moreover, our accelerator can achieve similar throughput and better accuracy compared with binarized accelerators from [11, 13].

VI. CONCLUSION

In this work, we propose an accurate and efficient residual network accelerator. The QR-DNN optimization algorithm converts a floating-point network to a low-precision batch-normalization-free network, and our systolic RNA architecture implements the quantized and reconstructed networks efficiently. Our techniques are shown to be suitable not only for residual networks, but also other well-known networks, such as AlexNet and VGG. Employing the systolic array architecture with dedicated shift-accumulation processing elements, our solution achieves $1\% \sim 2\%$ improvement in accuracy over existing techniques, and speedup over previous fixed-point architectures.

Acknowledgement: This research was supported under the Australian Research Councils Linkage Projects funding scheme LP130101034 and Zomojo Pty Ltd. We also appreciate the constructive suggestions of the anonymous reviewers.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.
[2] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ICML*.
[3] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," in *ICLR*, 2016.
[4] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," in *ICLR*, 2017.
[5] Y. Ma, M. Kim, Y. Cao, S. Vrudhula, and J. S. Seo, "End-to-end scalable fpga accelerator for deep residual networks," in *IEEE International Symposium on Circuits and Systems*, 2017.
[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.
[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Computer Science*, 2014.
[8] D. J. M. Moss, P. H. W. Leong, S. Krishnan, E. Nurvitadhi, P. Ratuszniak, C. Johnson, J. Sim, A. Mishra, D. Marr, and S. Subhaschandra, "A customizable matrix multiplication framework for the intel harpv2 xeon+fpga platform: A deep learning case study," in *FPGA*, 2018.
[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe:convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014.
[10] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *FPGA*, 2016.
[11] L. Jiao, C. Luo, W. Cao, X. Zhou, and L. Wang, "Accelerating low bit-width convolutional neural networks with embedded fpga," in *FPL*, 2017.
[12] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, and S. Song, "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA*, 2016.
[13] H. Yonekawa and H. Nakahara, "On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an fpga," in *Parallel and Distributed Processing Symposium Workshops*, 2017.
[14] M. B. Julian Faraone, Nicholas Fraser and P. H. Leong, "Syq: Learning symmetric quantization for efficient deep neural networkseal-time fpga-based anomaly detection for radio frequency signals," in *CVPR*, 2018.