

TB-DNN: A Thin Binarized Deep Neural Network with High Accuracy

Jie Wang*, Xi Jin*, Wei Wu*

* School of Microelectronics, University of Science and Technology of China, Hefei, Anhui Province, China

wangj1e@mail.ustc.edu.cn, jinxi@ustc.edu.cn, wuw1993@mail.ustc.edu.cn

Abstract— Deep neural networks (DNNs) have been widely used in many artificial intelligence (AI) applications. However, due to the huge demand for computing and storage resources as well as the high power consumption, deploying DNN models on embedded devices is full of challenges. Recent works have shown that DNN models can be compressed by removing their inner redundancy without obviously performance decay. In this work, we propose a two stage pipeline way to compress the ResNet-14 model and test it on CIFAR-10 and SVHN dataset respectively. Firstly, we use a filter level pruning method to remove the less important filters with different compression rate, and a considerable computation costs are reduced. Secondly, we binarize the pruned model to further reduce the model size and computational complexity. The training results show that we achieve 87.7% accuracy with only 1.86Mb model size on CIFAR-10 and 96.2% accuracy with 1.34Mb on SVHN. Compared to the original model, we have 57% to 68% FLOPs reduction and 45.6× to 63.1× model size compression at the cost of roughly 4% accuracy drop. Finally, we implement the thin binarized ResNet-14 model on the Xilinx KC705 board with a shared, flexible accumulator, which can save 46.8% logic resources. And the entire network parameters are store into on-chip RAM, which can greatly reduce the energy consumption and memory overhead caused by off-chip accesses. The experimental results show that on CIFAR-10 dataset, we achieve an overall performance of 1200 FPS, energy efficiency of 571 FPS/W, which denote 2.3× and 3.6× improvements over the most recent work.

Keywords— Model compression, binarized neural network (BNN), field-programmable gate array (FPGA), fully on-chip

I. INTRODUCTION

In recent years, deep neural networks (DNNs) have been applied in many fields such as intelligent transportation [1], marine surveillance [2], speech recognition [3] and medical images[4]. However, with the high computational complexity, memory footprint and power consumption, the development of DNNs in resource-limited embedded platforms is a great challenge.

To solve the problems, several methods have been proposed by reducing the inner redundancy of DNN models. These methods mainly include filter pruning [5], sparsity [6], low precision DNNs [7], and binary DNNs (BNNs) [8]. The filter pruning [5] uses the statistics information computed from its next layer to prune the weak filters, which can effectively reduce more than half of the parameters and computation cost of ResNet-50 model and with roughly 1% accuracy drop on

ILSVRC-12 benchmark. In [6], the importance of a neuron connection is determined by its weight, and all connections whose weight is lower than a threshold will be removed, thus, a dense layer is converted to a sparse layer. Finally, on ImageNet dataset, this method can reduce the number of parameters at a factor of 9× to 13× without incurring accuracy loss. However, this compression method only works for the fully connect layer. Ref [7] chose 16 bits precision rather than 32 bits and proved that it has lightly impact on recognition accuracy. However, in post-training quantization, when the numerical precision is lower than 8 bits, the network's accuracy will steeply drop. Ref [8] proposed one extreme quantization scheme called binarized neural network (BNN) which constraints both the weights and activations to +1 or -1. Compared to DNN, BNN can not only compress the parameter size by 32× but also replace the complexed float-point multiplication operation with simple bitwise logic operation, which make it possible to deploy neural networks on resource-limited and high energy-efficient embedded devices.

In hardware implementation, it is obviously that due to the high power consumption, GPU-based accelerators cannot fit in the embedded platforms. Differently, FPGA benefits from much higher energy efficiency and its LUT-based fabric is very suitable for the bit-wise operation of BNNs. Thus, many FPGA-based accelerators of BNN have been proposed [9-13].

In this paper, we propose a combined model compression way to further compress the model, and implement the optimized model with high throughput on FPGA. Our specific contributions are as follows:

- We propose a two stage pipeline way that combined filter pruning and binarization together to further compress the model, and the training results show that we achieve state-of-the-art performance in accuracy with the least model size on the CIFAR-10 and SVHN dataset.
- To achieve a high performance and high energy efficiency accelerator, a shared, flexible and reusable accumulator unit is designed in our work, we achieve 2.3× and 3.6× improvements in FPS and energy efficiency over the most recent work.

The rest of this paper is organized as follows: Section II reviews previous work on model compression for DNNs. Section III provides the algorithm optimization method and the optimization results. Section IV describes the design and

optimization of our accelerator and the comparison results with other BNN accelerators. Section V is the conclusion of this paper.

II. RELATED WORK

With the inner structural redundancy of DNNs, deep compression techniques have been studied to reduce the computation complexity and memory footprint. Ref [5] delete the whole weak convolution filters which can effectively reduce more than half of the model size and computation cost with only 1% recognition performance decay on ILSVRC-12 benchmark, moreover, the pruned model can keep the same structure to the original model. However, this scheme has not compressed the model in numerical precision, which means the operations of the pruned model are still float-point operations. Ref [14] used a sparsity and low precision quantization combined way to compress the model, which can successfully reduce the model size by $35\times$ to $49\times$ without loss of classification accuracy on ImageNet dataset. But the pruning method used in [14] makes the model become sparse, which may not adequately reduce the computation costs in the convolutional layers. Inspired by [14], we propose a similarly two stage pipeline way: filter pruning and binarization, that work together to further reduce both the model size and computation cost, while the weight matrices are still regular, and the complex multiply operations are replaced by the bit-wise operations.

III. ALGORITHM OPTIMIZATION

We propose a two stage pipeline method to compress the ResNet-14 model. Firstly, we use the filter pruning method to remove the “useless” filters with different compression rate, in this way, we can get a “thin” pruned model whose parameters and computation cost are reduced. Then we binarize the pruned model to further reduce the model size and computational complexity. The whole process is shown in Figure 1.

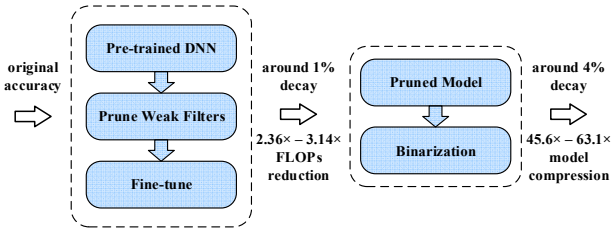


Figure 1. The two stage pipeline compression: filter pruning and binarization. Filter pruning reduces the FLOPs by $2.36\times$ to $3.14\times$ at the cost of roughly 1% accuracy drop, while binarization further compress the model by $45.6\times$ to $63.1\times$.

A. Filter Pruning

For filter pruning, we use the method that proposed by [5] to prune our DNN model: evaluate the importance of each feature map, remove the “useless” channel of feature maps and its associated convolution kernel, and fine-tune the remained neural network. The filter pruning framework is illustrated on the left side of Figure 1.

1) Pruning Strategy

As described in [5], with the special structure in ResNet, there exists some restrictions when pruning. For example, the number of input channels must be equal to the output channel numbers of the current block to complete the sum operation. For ResNet-14 model, there are only two convolutional layers in each block, so we choose to only prune the filters in the first convolutional layer. And the input channels of the second convolutional layer’s filter is reduced at the same time, which is illustrated in Figure 2.

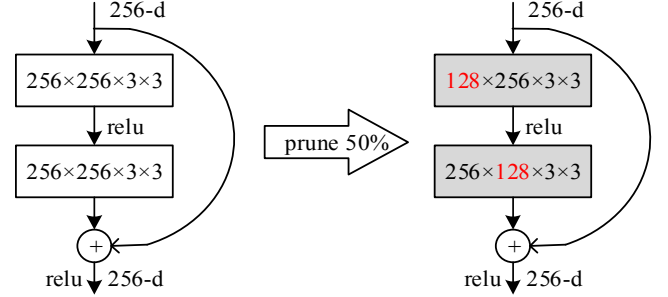


Figure 2. Illustration of the ResNet pruning strategy.

2) Pruning Results

With the pruning method described above, we use different compression rate (preserve 70%, 50%, 30% filters in each pruned layer respectively) to prune the ResNet-14 model and test both on CIFAR-10 and SVHN dataset. The results are shown in Table 1. Compared to the original ResNet-14 model, we can find that at the cost of roughly 1% accuracy drop, there are around 57.5% to 68.5% FLOPs and 26.7% to 46.2% model size can be reduced on CIFAT-10 and SVHN dataset respectively.

Table 1. FILTER PRUNING RESULTS OF RESNET-14

Model	CIFAR-10 Top-1	SVHN Top-1	Model size	#FLOPs
Original	91.79%	99.14%	84.8Mb	213.97M
Preserve-70	90.85%	98.22%	62.12Mb	90.85M
Preserve-50	90.28%	97.92%	45.64Mb	67.42M
Preserve-30	89.12%	97.25%	28.72Mb	42.47M

B. Binarization

We adopt the method proposed by [8] to binarize our pruned model. Here we just make some description about the activation function and the batch normalization layer.

1) HardTanh vs ReLU

Rectified linear unit (ReLU) has been widely used in neural networks due to its simplicity and the ability to enable fast training, which is defined as

$$ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \quad (1)$$

Differently to ReLU, the derivative of HardTanh function will never be zero even when a neuron’s input is negative, which means all the parameters of neural network can get update during the back propagation phase, and finally make the

neural network convergent faster. The HardTanh function is defined as

$$\text{HardTanh}(x) = \begin{cases} 1, & \text{if } x > 1 \\ -1, & \text{if } x < -1 \\ x, & \text{otherwise} \end{cases} \quad (2)$$

During the binarization stage, we test both ReLU and HardTanh, and the training result shows that HardTanh function's performance is better as illustrated in Figure 3.

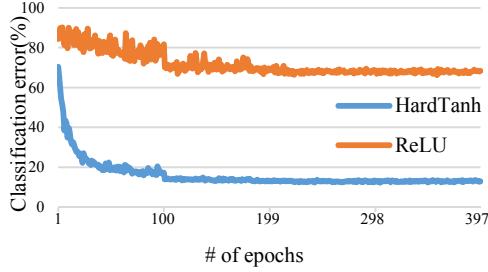


Figure 3. Training results of using HardTanh and ReLU as activation function.

2) Batch Normalization

Batch normalization [15] can effectively reduce the information lost during binarization and has been an essential technique in BNN. The mathematical function is given in Equation (3),

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \gamma + \beta \quad (3)$$

where x and y are input and output, μ , σ , γ and β are all obtained during the training phase, and will be fixed during inference. Compared to the bitwise logic operation in BNN, the areas for the multiplier and adder that cost by float-point operation (caused by μ , σ , γ and β) are huge. Thus, we choose to remove all the batch normalization layer in the shortcut layer, which is shown in Figure 4. In this way, we can successfully simplify the three consecutive layers (batch normalization layer, HardTanh layer and binarization layer) into a simple normbinarize (NB) layer, more details can be found in [9].

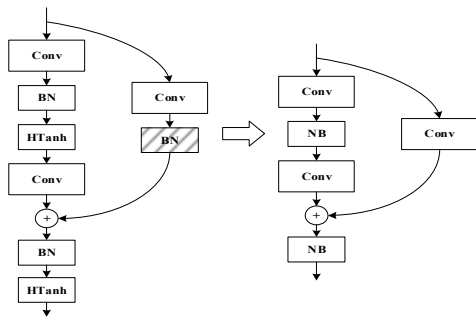


Figure 4. Batch normalization layer transformation: construct an independent NB layer.

3) Binarization Results

We binarize the pruned model with different compression rate. Furthermore, in order to realize a truly binarized neural network, we pad all the input feature maps of convolutional

layer with +1 rather than 0, then all the numerical values in our neural network can be represented by 0 and 1 with only 1 bit, except for the first convolutional layer. The whole binarization training process on CIFAR-10 and SVHN are shown in Figure 5 and Figure 6 respectively. And we can find that the +1 padding operation has no obvious effect on the BNN.

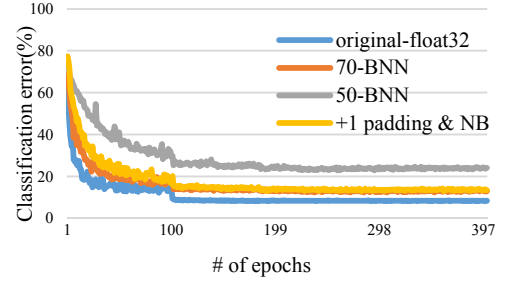


Figure 5. TB-DNN test on CIFAR-10 dataset: 70-BNN and 50-BNN refer to the binarization result with different pruning rate. +1 padding & NB refers to the binarization result with +1 padding and transforming the batch normalization layer.

Differently to CIFAR-10, we also test the binarization result with 70% pruning rate on SVHN dataset, and there is no obviously performance loss.

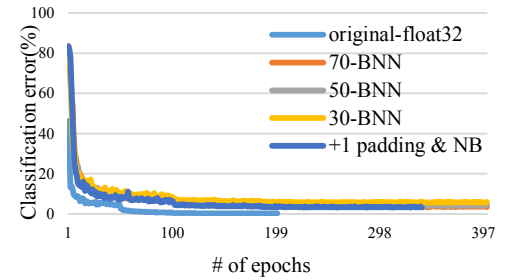


Figure 6. TB-DNN test on SVHN dataset.

C. Model Compression Results

Table 2 lists the performance comparison with other recent BNNs in accuracy and model size. We can find that our work has a significant improvement in model compression. Although [16] achieves around 1% higher accuracy on CIFAR-10 dataset, its model size is 7.3× of us. And we use around 50% parameters of [16] to get a same accuracy on SVHN dataset. In summary, after comprehensive comparison with other recent BNN work, our work achieves state-of-the-art performance in accuracy with the least model size.

Table 2. MODEL COMPRESSION RESULTS OF RESNET-14 ON CIFAR-10 AND SVHN DATASET

		[11] (2017)	[17] (2018)	[16] (2019)	Ours
CIFAR-10	Top-1	80.1	85.9	88.6	87.7
	Model size(Mb)	13.4	2.3	13.6	1.86
SVHN	Top-1	94.9	-	96.9	96.9 (96.2)
	Model size(Mb)	13.4	-	3.6	1.86 (1.34)

IV. ACCELERATOR ARCHITECTURE

A. Architecture Overview

The TB-DNN architecture on CIFAR-10 dataset is shown in Figure 7, mainly includes the fixed-point convolutional layer (the first convolutional layer), binary convolutional layer, normbinarize layer (NB), global average pooling layer (pl), shortcut layer and fully connected layer (fc), and there are only two different kernel sizes, 1×1 for the shortcut layers, 3×3 for the rest convolutional layers. In order to achieve a high performance accelerator, here we choose to employ independent modules for the different layers, except for the shortcut layer and the fc layer, which can be implemented by the 1×1 convolution module in common.

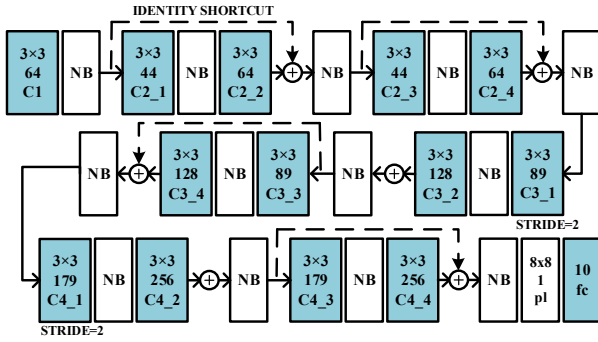


Figure 7. TB-DNN architecture on CIFAR-10 dataset. C: convolutional layer with first row as filter size and second row as the number of convolution kernels. NB: normbinarize unit. pl: global average pooling layer with the first row as window size and second as output size. fc: fully connected layer.

Correspondingly, the overall hardware architecture is shown in Figure 8. With the scheduling of the controller, the input data and weights are loaded from the DRAM to the local memory by the DMA at first, then they will flow between the different computing units to complete the calculation. Apart of them, the accumulator unit is designed to perform different functions according to the states, either completing the convolution operation of the first layer or completing the accumulation of different channels' feature maps in the hidden layer.

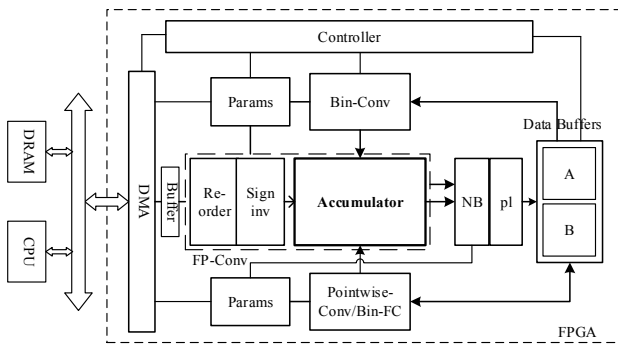


Figure 8. Overall hardware architecture.

B. Design Consideration

With the algorithm optimizations proposed in the previous section, we are able to store all the feature maps, weights and

some reference values (for the batch normalization layer) into the on-chip RAMs, thus, the demand to memory bandwidth is greatly satisfied. Besides, the original multiply operations are replaced by the sign inversions or XNOR operations, lots of logical resources can be saved by the way. As a result, there are mainly three aspects to consider while designing the accelerator to achieve higher performance.

DSP48E based high-parallelism system. On the one hand, we are committed to completing more than one three dimensional convolution operations at one time, and the max value of the input channels equals to 256 as shown in Figure 7. On the other hand, for the convolution operation of the first layer and the accumulation of different channels' feature maps in the hidden layer, the input data is multi-bit fixed-point, hence we use the DSP48E to complete the accumulator rather than the LUT slice. With the limited resource of DSP48E in FPGA, Table 3 shows the performance and resource utilization with different values of f_k for the accumulator unit, f_k is filter parallelization factor, whose input channels equal to 256.

Table 3. COMPARISON OF DIFFERENT PARALLEL FACTORS

f_k	DSP	FPS
1	266	577
2	532	1200
3	798	1810

Maximizing filter reuse. For the design of TB-DNN, there are two aspects of filter reuse. First, the one input feature map is involved in the computation of the shortcut layer and the block layer with different convolution kernels. Second, for the all convolutional layers, the input feature map is convolved with different filters to generate the different channels of output feature map.

High-performance memory system. To achieve higher performance, a large number of feature maps and weights must be read in the same clock cycle. Thus we choose to split the feature maps and weights into smaller ones by the channel and store them in multiple memory unit horizontally. With the "Serial write, read out in parallel" method, we can successfully read multiple channels of data from the different block RAMs simultaneously.

In this paper, we choose to configure f_k with 2, and the number of 3-dimensional convolution operations can be executed at one time of different convolutional layers are presented in Figure 9.

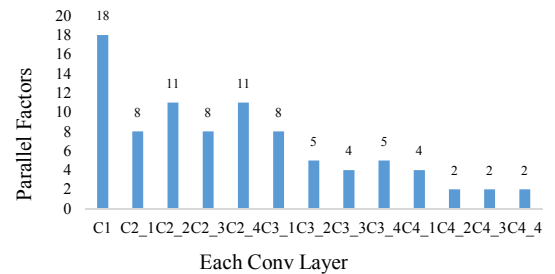


Figure 9. The parallel factors of each layer.

C. Accumulator Unit

Figure 10 shows the core architecture of the accumulator unit, which is a 27-input adder tree (adder27). And the adder27 unit consists of three 9-input adder trees (adder9) and a 3-input adder tree (adder3). More particularly, the adder9 unit consists of two 4-input adder trees and a 3-input adder tree. With the scheduling of the controller, the adder27 unit can be converted to independent adder3 unit, adder4 unit and adder9 unit or other cascaded adder tree module. And the whole accumulator unit consists of a number of adder27 units, which is determined by the filter parallelization factor f_k .

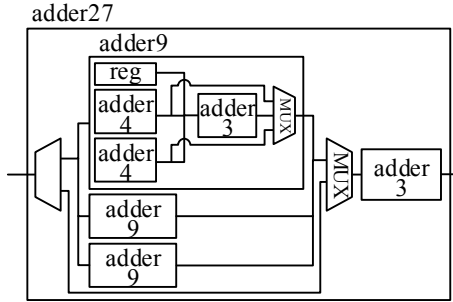


Figure 10. The architecture of accumulating kernels.

D. Experimental Results

We implement the TB-DNN model on the Xilinx KC705 evaluation board in hardware description language, and Table 4 shows the resource utilization summary for the whole network, and we can easily conclude that a lot of logic resources are saved due to the binarized neural network. As for the DSP resource, it is used to complete the accumulation of feature maps, not the multiplication operations in the original convolutional operations.

Table 4. FPGA RESOURCE UTILIZATION SUMMARY

Resource	LUT	BRAM	FF	DSP
Used	17068	130	28347	532
Available	203800	445	407600	840
Utilization (%)	8.4	29.2	7.0	63.3

Table 5 compares our implementation against other FPGA based BNN accelerators and our implementation results in 1.3× to 7.1× better performance in FPS and 1.47× to 15.9× better in energy efficiency than state-of-the-art implementations.

Table 5. RESULTS IN COMPARISON WITH OTHER FPGA-BASED ACCELERATORS

	[10] (2017)	[17] (2018)	[16] (2019)	Ours
Clock(MHz)	143	143	143	100
FPS	168	930	520	1200
GOPS	207.8	-	722	1006
Power[W]	4.7	2.4	3.3	2.1
FPS/W	35.8	387.5	157	571
GOPS/W	44.2	-	218	479

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a two stage pipeline way: filter pruning and binarization, to compress the ResNet-14 model. The training result shows that on CIFAR-10 dataset, we achieve 87.7% accuracy with only 1.86Mb model size. Compared to the original model, we achieve 57% FLOPs reduction and 45.6× model size compression. And on SVHN dataset, we compress the model with a little higher compression rate, and finally we achieve 96.2% accuracy with 1.34Mb model size, 68% FLOPs reduction and 63.1× model size compression. Then we modify the model by remove the batch normalization layer of the shortcut layer and padding with +1 rather than 0 to achieve a truly BNN, and the training results show that the accuracy drop that caused by this operation is negligible. Finally, we implement the compressed model on Xilinx KC705 board with a shared, flexible accumulator. The experimental results show that on CIFAR-10 dataset, we achieve an overall performance of 1200 FPS, energy efficiency of 571 FPS/W, which denote 2.3× and 3.6× improvements over the most recent work. In the future work, the proposed model compression method will be tested on larger DNN models and datasets.

REFERENCES

- [1] SOON F C, HUI Y K, CHUAH J H, et al. PCANet-Based Convolutional Neural Network Architecture For a Vehicle Model Recognition System [J]. 2019, 20(2): 749-59.
- [2] ZHAO J, GUO W, ZHANG Z, et al. A coupled convolutional neural network for small and densely clustered ship detection in SAR images [J]. 2019, 62(4): 42301.
- [3] YING Z, PEZESHKIM, BRAKEL P, et al. Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks, F, 2016 [C].
- [4] XIA K J, YIN H S, WANG J Q J C C. A novel improved deep convolutional neural network model for medical image fusion [J]. 2019, 22(3): 1-13.
- [5] LUO J-H, WU J, LIN W. Thinet: A filter level pruning method for deep neural network compression; proceedings of the Proceedings of the IEEE international conference on computer vision, F, 2017 [C].
- [6] HAN S, POOL J, TRAN J, et al. Learning both weights and connections for efficient neural network; proceedings of the Advances in neural information processing systems, F, 2015 [C].
- [7] SUDA N, CHANDRA V, DASIKI G, et al. [ACM Press the 2016 ACM/SIGDA International Symposium - Monterey, California, USA (2016.02.21-2016.02.23)] Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '16 - Throughput-Optimized OpenCL-based FPGA Acce [J]. 16-25.
- [8] HUBARA I, COURBARIAUX M, SOUDRY D, et al. Binarized neural networks; proceedings of the Advances in neural information processing systems, F, 2016 [C].
- [9] LI Y, LIU Z, KAI X, et al. A 7.663-TOPS 8.2-W Energy-efficient FPGA Accelerator for Binary Convolutional Neural Networks (Abstract Only) [J]. 2017.
- [10] ZHAO R, SONG W, ZHANG W, et al. Accelerating binarized convolutional neural networks with software-programmable fpgas; proceedings of the Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, F, 2017 [C]. ACM.
- [11] UMUROGLU Y, FRASER N J, GAMBARELLA G, et al. Finn: A framework for fast, scalable binarized neural network inference; proceedings of the Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, F, 2017 [C]. ACM.
- [12] NAKAHARA H, YONEKAWA H, SASAO T, et al. A memory-based realization of a binarized deep convolutional neural network; proceedings of the International Conference on Field-programmable Technology, F, 2016 [C].
- [13] FU C, ZHU S, SU H, et al. Towards Fast and Energy-Efficient Binarized Neural Network Inference on FPGA [J]. arXiv preprint arXiv:181002068, 2018.

- [14] HAN S, MAO H, DALLY W J. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding [J]. arXiv preprint arXiv:1510.00149, 2015.
- [15] IOFFE S, SZEGEDY C. Batch normalization: Accelerating deep network training by reducing internal covariate shift [J]. arXiv preprint arXiv:1502.03167, 2015.
- [16] Guo P, Ma H, Chen R, et al. A High-Efficiency FPGA-Based Accelerator for Binarized Neural Networks[J]. Journal of Circuits System and Computers, 2019.
- [17] YANG L, HE Z, FAN D. A Fully Onchip Binarized Convolutional Neural Network FPGA Implementation with Accurate Inference; proceedings of the Proceedings of the International Symposium on Low Power Electronics and Design, F, 2018 [C]. ACM.
- [18] Hua B S , Tran M K , Yeung S K . Pointwise Convolutional Neural Networks[J]. 2017.



Jie Wang received his B.S. degree from Taiyuan University of Technology, and he is currently a M.S. student in school of microelectronics in University of Science and Technology of China, Anhui, China, under the supervision of Prof. Xi Jin. His current research work is mainly on FPGA-based Hardware Accelerator design.



Xi Jin received the B.S. degree from University of Science and Technology of China, Anhui, China, and he is currently an associate professor in Department of Physics in University of Science and Technology of China, Anhui, China. His research interests include SOC design technology, VLSI design, computer-aided design methodologies for SoC system integration and FPGA-based Hardware structure design.



Wei Wu received his B.S. degree from University of Science and Technology of China, and he is currently a M.S. student in Department of Physics in University of Science and Technology of China, Anhui, China, under the supervision of Prof. Xi Jin. His current research work is mainly on FPGA-based Hardware Accelerator design.