Angel-Eye: A Complete Design Flow for Mapping CNN onto Customized Hardware

Kaiyuan Guo*†, Lingzhi Sui†, Jiantao Qiu*, Song Yao† and Song Han‡
*Department of Electronic Engineering
Tsinghua University, Beijing, China
Email: gky15@mails.tsinghua.edu.cn, yu-wang@tsinghua.edu.cn
†Deephi Technology Co., Ltd, Beijing, China
‡Stanford University, Stanford, California, USA

***Abstract*—Convolutional Neural Network (CNN) has become a successful algorithm in the region of artificial intelligence and a strong candidate for many applications. However, for embedded platforms, CNN-based solutions are still too complex to be applied if only CPU is utilized for computation. Various dedicated hardware designs on FPGA and ASIC have been carried out to accelerate CNN, while few of them explore the whole design flow for both fast deployment and high power efficiency. In this paper, we propose Angel-Eye, a programmable and flexible CNN processor architecture, together with compilation tool and runtime environment. Evaluated on Zynq XC7Z045 platform, Angel-Eye is 8× faster and 7× better in power efficiency than peer FPGA implementation on the same platform. A demo of face detection on XC7Z020 is also 20× and 15× more energy efficient than counterparts on mobile CPU and mobile GPU respectively.**

I. Introduction

Convolutional Neural Network (CNN) is one of the state-of-the-art artificial intelligence algorithms. With a large model and enough training data set, CNN generates complex features for certain tasks, which outperforms traditional handcrafted features. Thus CNN can help achieve the top performance in regions like image classification, object detection and even stereo vision. Some audio algorithms also involves CNN as one of the feature extraction steps.

Despite the outstanding performance, CNN is hard to be implemented in daily applications, because of its high computation complexity. Large CNN models can involve up to nearly 40G operations (multiplication or addition) [1] for the inference of one 224×224 image. Larger images in real applications can scale this number up. Thus this kind of applications are usually implemented as a cloud service on large servers. For personal devices, traditional CPU platforms are hardly able to handle CNN models with acceptable processing speed. For tasks like object detection where real-time processing is required, the situation is worse.

GPU offers high degree of parallelization, thus is a good candidate for compute CNN. But usually a batch of images should be processed together to fully utilize the computation capacity. For video processing, this means larger latency for each frame. On the other hand, recent work have shown that the data precision in CNN processing can be reduced from 32-bit floating point to 16-bit fixed point or even less. This means it is possible to achieve better energy efficiency than GPU with customized hardware.

Implementing customized hardware of CNN requires hardware and software co-design. The network should be adjusted to a hardware friendly form. Recent work on CNN have shown that the data format can be compressed from 32-bit floating point to fixed point. This greatly reduces the power and area cost of the hardware. We have shown that 8-bit fixed point is enough for VGG network [1]. Han et al. [2] compressed the data to 4-bit by weight sharing. Recent work even tries 1-bit weight for classification [3].

On the hardware side, an architecture with high efficiency and high flexibility is needed. Zhang et al. [4] proposed a floating point accelerator design on FPGA, where the loop unrolling parameter design space is explored for a certain network. But the situation for running different networks are not considered. ASIC design by Du et al. [5] consumes only 320mW power by a mesh grid design. However, it only supports CNNs with very limited model capacity due to on-chip memory constraint.

Previous work mainly maps the software model to hardware structure manually. But the great variety of the topologies of state-of-the-art CNNs brings challenge to hardware designers. ResNet [6], which is the winner of Image-Net Large-Scale Vision Recognition Challenge (ILSVRC) 2016, implements a 152-layer model with shortcut layers. But the winner of 2015 [1] uses cascaded 19 layers in their network.

So a mapping tool is needed to bridge software model with hardware structure. One choice is to automatically generate the hardware code for a certain network. This pushes the hardware performance to the extreme while the flexibility to different networks is low. Another choice is to use a flexible hardware structure and leave the mapping work to software side, which is the choice of our work. This can response to the changes in network topology quickly, no matter the platform is FPGA or ASIC. It also supports different network switch at runtime.

In this paper, we extend our previous work [7] to a complete design flow for mapping CNN onto customized hardware. Generally three parts are included in this flow:

- A data quantization strategy to compress the original network to a fixed-point form.

- A parameterized and runtime configurable hardware architecture to support various networks and fit into various platforms.
- An instruction set together with a compiler that bridges different topologies of convolutional neural networks with the proposed hardware architecture.

Our experiments on FPGA show that the proposed design flow delivers CNN acceleration with high energy efficiency. The rest of this paper is organized as follows. Section II introduces the background of CNN. Related work is then discussed in Section III. Details of the flow is shown in Section IV. We show the experimental results in Section V and concludes this work in Section VI.

II. Preliminary of CNN

A CNN consists of a set of *layers*. As the name suggests, the most important layers in CNNs are the Conv layers. Besides, FC layers, Non-Linearity layers, and pooling layers (down-sampling layer) are also essential in CNN.

Conv layer applies Conv operations on input feature maps and can extract high-level features from the inputs. An example is shown in Figure 1 (a), where the feature maps are blue and the 3-D Conv kernel is green. Each pixel of each output feature map is the inner product of a part of input with a 3-D convolution kernel.

FC layer applies a linear transformation on the input feature vector. It is usually used as the classifier in the final stage of a CNN. A simple FC layer with 4 input and 3 output is shown in Figure 1 (b) where each connection represents a weight of the model.

Non-linearity layer helps increase the fitting ability of neural networks. In CNN, the Rectified Linear Unit (ReLU), as shown in Figure 1 (c), is the most frequently used function [8]. Hyperbolic tangent function and sigmoid function are also adopted in various neural networks.

Pooling layer is used for down-sampling. Average pooling and max pooling are two major types of pooling layers. For a pooling layer, it outputs the maximum or average value of each subarea in the input feature map. Pooling layer can not only reduce the feature map size and the computation for later layers, but also introduces translation invariance. A simple max pooling layer with a 2×2 kernel is shown in Figure 1 (d).

A practical CNN for face alignment which we use for the verification of our design is shown in Figure 2. It calculates the coordinates of 5 character points of human face given the face image. Conv layers, Pooling layers and Non-linearity layers are interleaved to extract features. An FC layer at the end generates the coordinates of these points from extracted features.

III. Related Work

Though many regions in machine learning benefits from neural network like algorithms, one of the main drawback it brings is the high computation complexity, especially for CNNs. Various accelerator designs for CNN have been proposed. In this section, some state-of-the-art CNNs are introduced first. Then we give an overview of existing CNN accelerators.

A. State-of-the-art CNN Model

State-of-the-art CNN models differ greatly from earlier network in topology. Recent work is focusing more on the design of Conv layers than on FC layers. As in VGG network [1], 3 FC layers with more than 0.12 billion weights are used for the final classification. ResNet [6], the winner of Image-Net Large-Scale Vision Recognition Challenge (ILSVRC) 2015, implements 152-layers where only the last layer is fully connected. Shortcut structure is also introduced in Conv layers to reinforce the learning ability. Network with no FC layer is also proposed [9].

Convolution kernels in CNN is also changing. Early CNN designs [8] [10] adopted convolution kernels of size 11×11 for Conv layers, which are much larger than the 3×3 kernels in VGG networks [1] and the 1×1 kernels in SqueezeNet [11]. It is a trend to use smaller kernels in Conv Layers. Computation complexity reduces while the network performance remains.

B. CNN Accelerator

Various architectures have been proposed to accelerate CNN, including both ASIC and FPGA designs. As discussed in [4], one Conv Layer can be expressed as a 6-layer loop. The key point in CNN accelerator design is the unrolling strategy of the loops for each layer.

Fixed loop unrolling strategy is commonly applied in CNN accelerator designs. Zhang et al. [4] analyzed the data sharing relation of different iterations of a loop to evaluate the cost of unrolling. Calculation on different input channels and that for different output channels are of the lowest cost to be parallelized. But image pixel level and kernel level parallelization are not fully explored. nn-X [12] adopted 2-D convolver design of size 10×10, which achieves kernel level parallelization. Our previous work [7] uses a 3×3 convolver design targeting at VGG network. ShiDiannao [5] implements a mesh grid style structure to achieve parallelization on image pixel level. But the routing cost is rather high. Similar strategy is also adopted by [13].

Configurable loop unrolling costs much in data routing but can fit into different network topologies better. Chen et al. [14] proposed a 2-D PE array design optimized for CNN. Global bus is used to broadcast and collect data from PEs. The connections are configurable to group different PEs together as convolvers of different sizes. The drawback is the routing cost on extra bits to identify the target PE of the data.

IV. Flow Description

The overall structure of the design flow is shown in Figure 3. Three parts will be introduced in this section: data quantization, hardware architecture, instruction set and compiler.

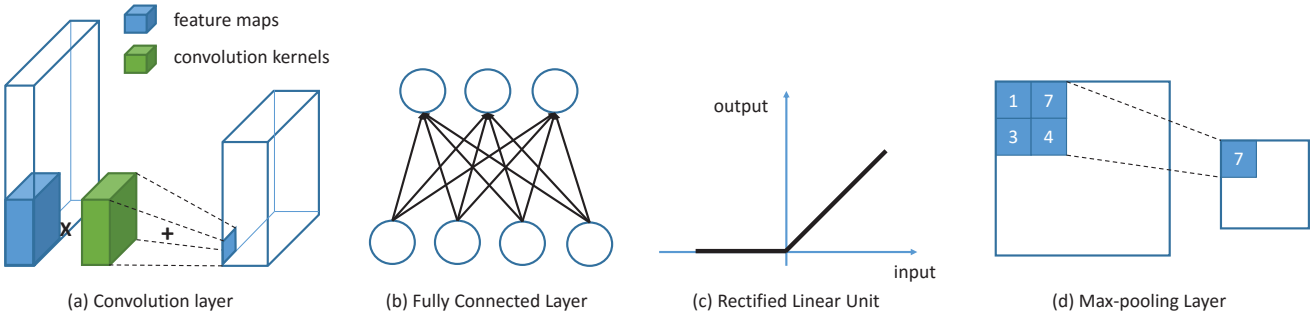

Fig. 1: Typical layers in CNN: (a) Convolutional layer; (b) Fully-Connected layer (dense matrix multiplication); (c) Non-linear layer with Rectified Linear Unit; (d) Max-pooling layer with 2×2 kernel.

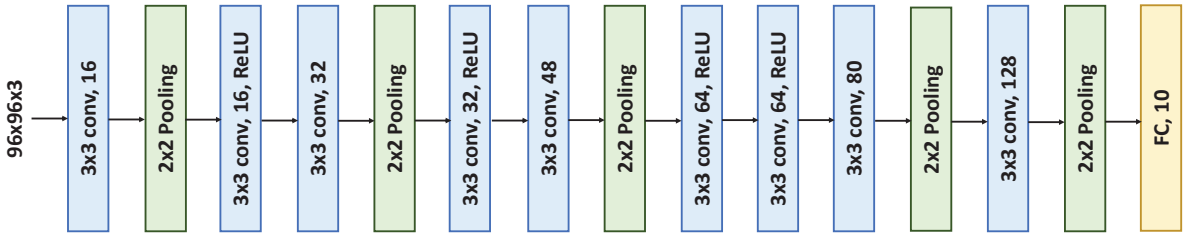

Fig. 2: A practical CNN model for face alignment. For each layer, the kernel size, output channel number and nonlinearity type is given.

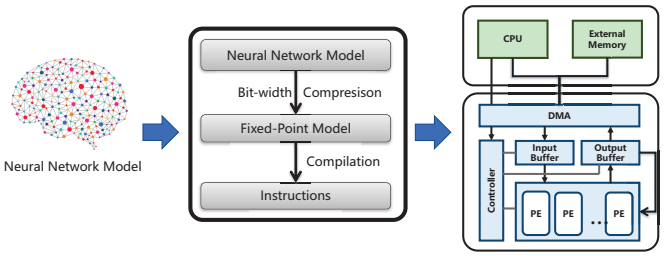

Fig. 3: Design flow from CNN model to hardware acceleration

A. Data Quantization

Data quantization with less bit is a way to compress the network. Dynamic range of data across different layers in a CNN is usually large. Thus a uniform quantization to all the layers may incur great performance loss. We propose a quantization strategy where the radix point position is dynamic across different layer. The strategy tries to find the best radix point position in each layer given the bit-width. This is hardware friendly because only extra shifters are needed to align the data. Fixed-point adders and multipliers remain unchanged.

The optimization target is to minimize the residual error of the last layer's output between the fixed point network and the floating point network. The solution is the radix position of the data of each layer. Calculate the error of one solution requires a test of tens of pictures, which will be of high cost if all the possible solutions are to be tested. To limit the solution space, we use a greedy method and optimize the radix position layer by layer. The data quantization flow is shown in Figure 4.

B. Hardware Architecture

The proposed hardware architecture is shown in Figure 5. It can be divided into four parts: PE array, On-chip Buffer, External Memory and Controller. Compared with our previous work [7], we isolate the core logic from a host ARM core and add some features to make a more general design.

PE Array: Two levels of parallelism are implemented by PE array: inter-PE and intra-PE. Different PEs share the same input channels and use different kernels to calculate different output channels in parallel. Detailed structure of a single PE is shown in Figure 6. Within each PE, different convolvers calculate 2D convolution on different input channels in parallel. The convolvers employ the line buffer design [15] and achieve kernel level parallelization. A 3×3 kernel design is adopted because this size is most commonly used in state-of-the-art CNNs. It also supports kernels of smaller size by padding and kernels of larger size by time division multiplexing. The number of PE and convolvers are parameterized to fit into different platforms.

On-chip Buffer This part separates PE Array with External Memory. This means data I/O and calculation can be parallelized. Output buffer also offers intermediate result to PE Array if more than one round of calculation is needed for an output channel. We introduce a 2-D description interface to manage the data. This enables the user to fully utilize the on-chip buffer. For example, a ping-pong strategy can be implemented by manually split the address space for ping and

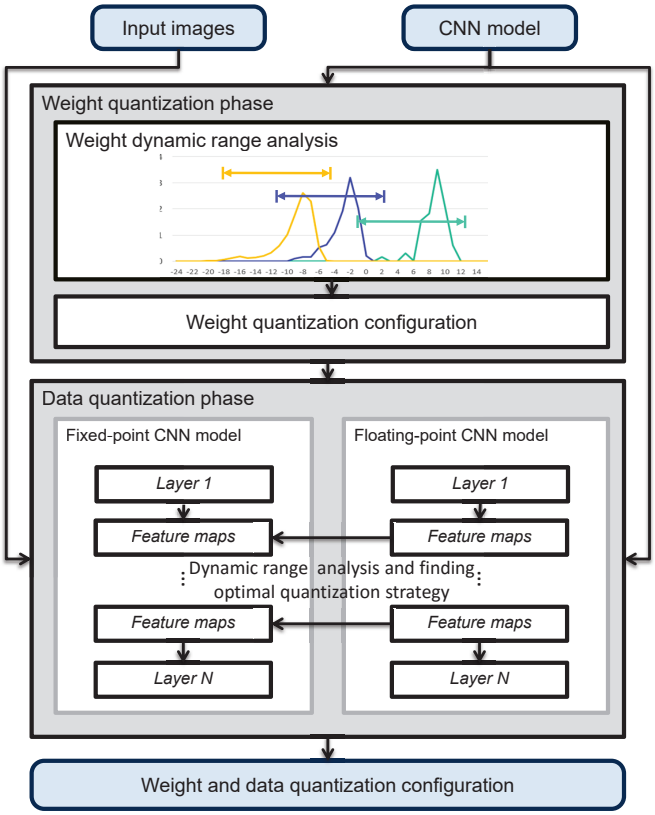

Fig. 4: Data quantization flow [7]

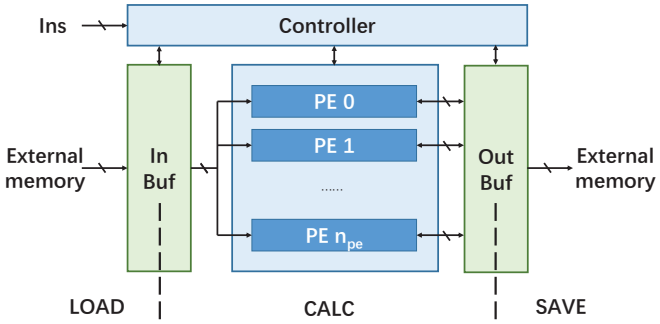

Fig. 5: Overall architecture of Angel-Eye

pong. For small layers, all the input channels can be loaded on chip. This reduce redundant external memory access.

External Memory For state-of-the-art CNN, On-chip Buffer is usually insufficient to cache all the parameters and data. External memory is used to save all the parameters of the network and the result of each layer.

Controller This part receives, decodes and issues instructions to the other three parts. Controller monitors the work state of each part and checks if the current instruction can be issued. Thus the host can send the generated instructions to Controller through a simple FIFO interface and wait for the work to finish by checking the state registers in Controller. Figure 7 shows the structure of this part. The data dependency

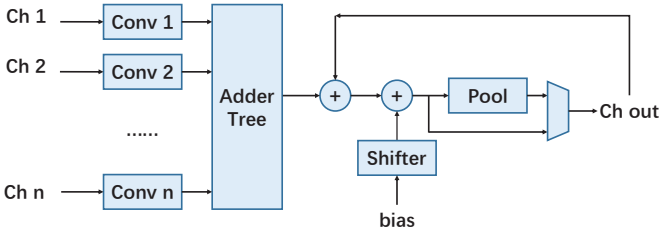

Fig. 6: Structure of a single PE

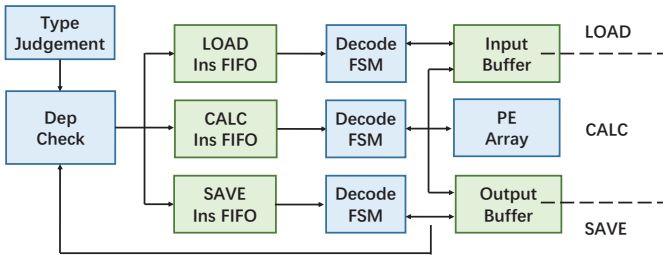

Fig. 7: Structure of Controller

problem will be discussed in section IV-C.

C. Instruction Set and Compiler

For this CNN accelerator, an instruction set of three instructions: LOAD, SAVE, and CALC, is proposed, corresponding to the I/O with external memory and the calculation done by PE Array. Address and size parameters for data in External Memory and On-chip Buffer are all embedded in instructions. Padding, pooling and data shift options can be set in CALC instructions. In this case, the hardware can support different networks by simply changing software.

Although I/O with external memory can be parallelized with calculation, there is latent data dependency problem. We set three flag bits in each instruction denoting if this instruction depends on the last previous LOAD or CALC or SAVE instruction. The schedule strategy can thus be managed during compilation and no extra workload is needed for the host at runtime.

A compiler is proposed to map a network descriptor file to hardware instructions. For each layer, the compiler tries to optimize the calculation time first. If various choices exist, it tries to minimize external memory access to reduce the power and bandwidth cost. The compiler is parameterized to support different hardware designs.

V. Experiment

In this section, the proposed data quantization strategy is analyzed on different state-of-the-art CNNs. The hardware peformance is then evaluated with the quantized networks.

A. Data Quantization Result

The proposed data quantization strategy is evaluated on four networks: CaffeNet, which is a replication of AlexNet, VGG network, GoogLeNet [16], and SqueezeNet. 50 images are used to optimize the radix position of each layer. 5000 images

TABLE I: Hardware Parameter and Resource utilization

design	data	#PE	#Conv	FF	LUT	BRAM	DSP	Clock
XC7Z045	16-bit	2	64	127653(29%)	182616(84%)	486(89%)	780(87%)	150MHz
XC7Z020	8-bit	2	16	24184(23%)	27215(51%)	68(49%)	198(90%)	100MHz

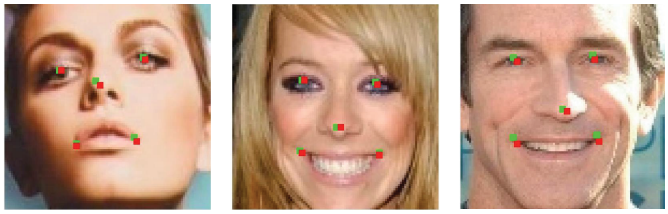

Fig. 8: Five point face alignment result. Red points: floating-point network result. Green points: 8-bit fixed point network result.

are used to test the classification accuracy of the network. Experimental results are shown in Table II.

For CaffeNet, the top 5 hit rate decreases about 1% by 8-bit fixed point quantization. For VGG network, 16-bit fixed point compression introduce only 0.06% extra top-5 error rate. That for the 8-bit form is 0.62%, which is negligible in common cases. For GoogLeNet and SqueezeNet, less than 1% accuracy loss is brought by 8-bit quantization.

We also tested this strategy on the face alignment network in Figure 2. Compared with classification, this task requires a higher data precision. Example alignment results are shown in Figure 8. The coordinate error is within 2 pixels.

TABLE II: Exploration of different data quantization strategies with state-of-the-art CNNs.

Network	Data bitwidth	Weight bitwidth	Top1 Accuracy	Top5 Accuracy
CaffeNet	float		53.90%	77.70%
	16	16	53.90%	77.12%
	8	8	53.02%	76.64%
VGG16	float		68.10%	88.10%
	8	16	66.58%	87.38%
	8	8 or 4	66.96%	87.60%
GoogLeNet	float		68.62%	88.82%
	16	16	68.60%	88.82%
	8	8	68.40%	88.64%
	6	6	65.14%	86.50%
SqueezeNet	float		57.02%	79.72%
	16	16	57.04%	79.72%
	8	8	55.82%	79.16%
	6	6	39.28%	63.86%

[1] The weight bits "8 or 4" means 8 bits for CONV layers and 4 bits for FC layers.

B. Hardware Performance

Two FPGA based design of the hardware architecture is carried out. A 16-bit version of the design is implemented on the Xilinx XC7Z045 chip which targets at high performance applications. An 8-bit version is implemented on the Xilinx XC7Z020 chip which targets at low power applications.

The hardware parameter and resource utilization of our design is shown in Table I. By choosing the design parameters properly, we can fully utilize the on-chip resource. Note that we are not using all the resource on XC7Z020 because the design coexists with an HDMI display logic for our demo. The clock speed is also affected by the routing difficulty on XC7Z020 where the resource is quite limited.

TABLE III: Performance comparison of Angel-Eye on XC7Z045 with other FPGA designs [7]

	[17]	[12]	[4]	Angel-Eye
Platform	Virtex5 SX240t	Zynq XC7Z045	Virtex7 VX485t	Zynq XC7Z045
Clock(MHz)	120	150	100	150
Bandwidth (GB/s)	–	4.2	12.8	4.2
Quantization Strategy	48-bit fixed	16-bit fixed	32-bit float	16-bit fixed
Power (W)	14	8	18.61	9.63
Performance (GOP/s)	16	23.18[1]	61.62	187.80
Energy Efficiency (GOP/J)	1.14	2.90	3.31	19.50

[1] The performance is of the face detector application in [12] where the 552M-op network is running at 42 frames per second.

TABLE IV: Performance comparison of Angel-Eye on XC7Z202 with TK1 on 5-point face alignment task

	CPU(TK1)	GPU(TK1)	Angel-Eye(XC7Z020)
Performance (GOP/s)	2.73	7.31	19.2
Speed(ms/f)	38.3	14.3	5.45
Power(W)	5-10		2

For the implementation on XC7Z045, we test it with VGG-16 network. The result together with that of similar work is shown in Table III. It can be seen that the proposed system gives out the best energy efficiency. Only the convolution layers are considered in this experiment though FC layer is also supported.

For the implementation on XC7Z020, a face detection task is used to test the whole system. Face proposals are first generated by traditional CV algorithm. The CNN shown in Figure 2 is then run on FPGA to do alignment. We aggressively compress the model to 8-bit fixed point for this task and the test result is quite accurate. A demo video is available at: https://youtu.be/m4e1SV89Dpg. The performance comparison with NVIDIA Terga K1 is shown in Table IV. Angel-Eye on XC7Z020 is $7\times$ and $2.6\times$ faster and is also $20\times$ and $15\times$ more energy efficient.

VI. Conclusion

In this paper, we propose a complete flow for mapping CNN onto customized hardware. This includes network quantization, flexible hardware architecture, and a compiler bridges the gap between them. Experimental results show that the generated design leads both in speed and energy efficiency compared with peer FPGA designs or implementations on mobile GPU. Future work will focus on better network compression strategy with corresponding hardware.

ACKNOWLEDGMENT

Yu Wang's work was supported by 973 project 2013CB329000, National Natural Science Foundation of China (No. 61373026, 61261160501), the Importation and Development of High-Caliber Talents Project of Beijing Municipal Institutions, Microsoft, Xilinx University Program, and Tsinghua University Initiative Scientific Research Program.

REFERENCES

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

[2] S. Han, J. Pool, J. Tran *et al.*, "Learning both weights and connections for efficient neural network," in *NIPS*, 2015, pp. 1135–1143.

[3] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," 2016.

[4] C. Zhang, P. Li, G. Sun *et al.*, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *FPGA*. ACM, 2015, pp. 161–170.

[5] Z. Du, R. Fasthuber, T. Chen *et al.*, "Shidiannao: shifting vision processing closer to the sensor," in *ISCA*. ACM, 2015, pp. 92–104.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[7] J. Qiu, J. Wang, S. Yao *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA*. ACM, 2016, pp. 26–35.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[9] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015, pp. 3431–3440.

[10] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*, 2014, pp. 818–833.

[11] F. N. Iandola, M. W. Moskewicz, K. Ashraf *et al.*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[12] V. Gokhale, J. Jin, A. Dundar *et al.*, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *CVPRW*, 2014, pp. 682–687.

[13] J. Sim, J.-S. Park, M. Kim, D. Bae, Y. Choi, and L.-S. Kim, "A 1.42tops/w deep convolutional neural network recognition processor for intelligent ioe systems," in *ISSCC*. IEEE, 2016.

[14] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *ISSCC*. IEEE, 2016.

[15] B. Bosi, G. Bois, and Y. Savaria, "Reconfigurable pipelined 2-d convolvers for fast digital signal processing," *VLSI*, vol. 7, no. 3, pp. 299–308, 1999.

[16] C. Szegedy, W. Liu, Y. Jia *et al.*, "Going deeper with convolutions," *arXiv preprint arXiv:1409.4842*, 2014.

[17] S. Chakradhar, M. Sankaradas, V. Jakkula *et al.*, "A dynamically configurable coprocessor for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 247–257.