PR 022

PipeCNN:An OpenCL-Based FPGA Accelerator for Convolution Neural Networks

Team members：Jianjing An, Diankun Jiang / Beijing Jiaotong University
Instructor: Dong Wang

I. High-level Project Description

PipeCNN is an efficient FPGA accelerator proposed by our team that can be implemented on a variety of FPGA platforms with reconfigurable performance and cost. The PipeCNN project proposed by our team is openly accessible,you can get it on our github website: https://github.com/doonny/PipeCNN. We use the PipeCNN- an efficient FPGA accelerator to demonstrate the following four application designs:

(1) ImageNet classification. ImageNet database was used and a number of five hundred pictures were processed on the test board. For AlexNet, the achieved classification speed is 110 ms per imagewe.

(2) Object recognition via camera. we use a USB camera as a video input, interactively intercept a picture from the video, then parallel computing it on the DE10-NANO platform, the final real-time classification of the target object and display in the VNC interface.

(3) Face recognition. In order to prove that our acceleration system is a universal model, in this section we use the VGG-Net network for face recognition experiments.

(4) Object Detection. Finally, we apply our proposed accelerator in the object detection. We use Faster R-CNN Net to detect the target and draw it out.

II. Block Diagram

2.1 The device of our design

The proposed DCNN accelerator was implemented on Altera Cyclone-V SoC-FPGA based DE10-NANO board. there are two parts in this project, in part 1,the ImageNet database was used and a number of five hundred pictures were processed on the test board. The average processing time was used as the final score. In part 2, we use a USB camera as a video input, interactively intercept a picture from the video, then parallel computing it on the DE10-NANO platform, the final real-time classification of the target object and display in the VNC interface. The device of our design as Fig.1.

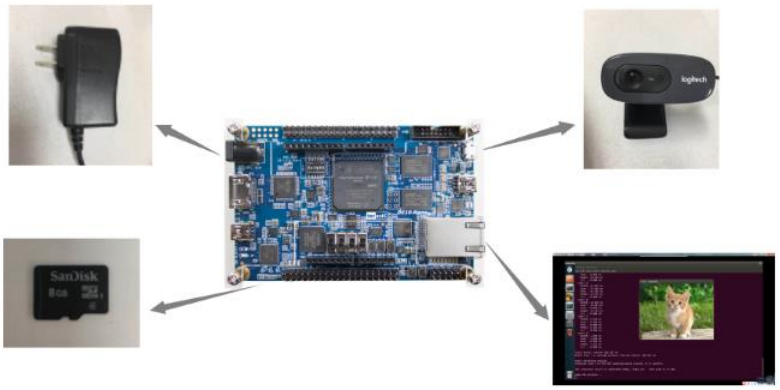

Fig.1 The device of our design

2.2 The top-level architecture of DCNN Accelerator

As shown in Fig.2, Our design consists of a group of OpenCL kernels that are cascaded by using Altera's OpenCL extension Channels. Two data mover kernels, namely MemRD and MemWR, transfer feature map and weight data from/to the global memory (i.e., the external DDR memory) feeding other kernels with high throughput data streams. The Convolution kernel (Conv.) is designed to accelerate the most computeintensive computations in CNNs, i.e., convolution layer and the FC layer. The Pooling kernel performs subsampling operations directly on the output data stream of the Conv. kernel. The Local Response Normalization (LRN) kernel fetches data from global memory and performs normalization on the feature map of neighboring neurons.

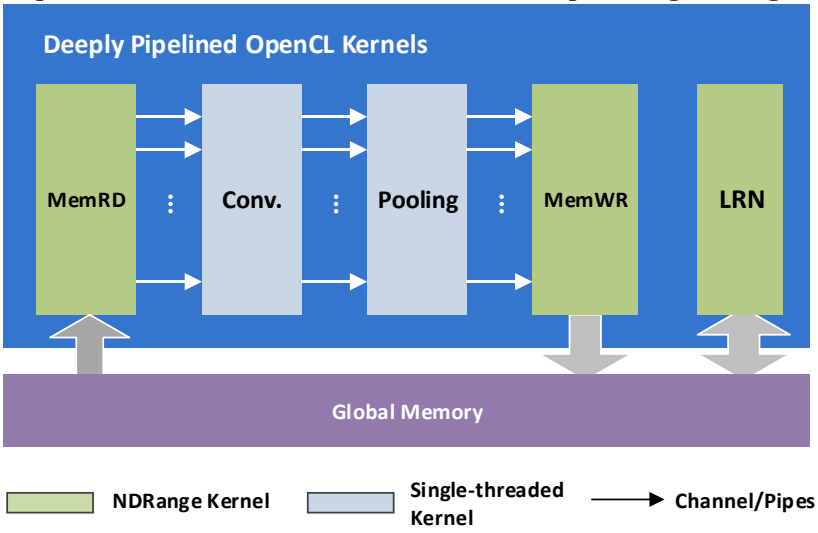

Fig.2：The top-level architecture of DCNN Accelerator

2.3 OpenCL-based FPGA design flow for DCNN accelerator

There is a growing trend among the FPGA community to utilize High Level Synthesis (HLS) tools to design and implement customized circuits on FPGAs. Compared with traditional methodology, the HLS tools provide faster hardware development cycle by automatically synthesizing an algorithm in high-level languages (e.g. C/C++) to RTL/hardware. Fig.3 summarizes the OpenCL-based FPGA accelerator development flow adopted by this work. Hardware circuits, which accelerate compute-intensive algorithms, are first modeled in OpenCL code in the form of kernel functions, and then compiled by HLS compiler to run on the FPGA fabric. A C/C++ code executing on the embedded CPUs provides vendor specific application programming interface (API) to communicate with the implemented kernels. This work uses the Altera OpenCL SDK to compile and profile the OpenCL designs on FPGAs. This toolset also provide the capability to warp RTL modules as C/C++ functions that can be instantiated inside the kernel. The designed host function initiates and launches the hardware kernels in a specific order according to the neural network configurations to accelerate the DCNN computation.

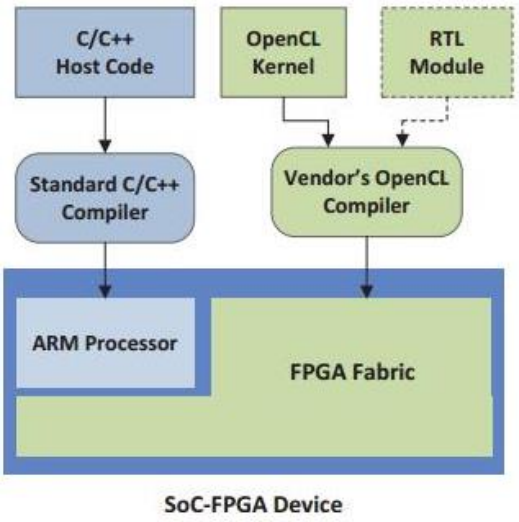

Fig.3: OpenCL-based FPGA design flow for DCNN accelerator

III. Intel FPGA Virtues in Your Project

3.1 Flexibility and reconfigurability

FPGA is a high-specification integrated circuit that can achieve unlimited precision functions through continuous configuration and splicing. unlike CPU or GPU, the bit width of basic data types is fixed, FPGA is flexible. FPGAs provide flexibility to implement the CNNs with limited data precision which reduces the memory footprint and bandwidth requirements, resulting in a better energy efficiency.

3.2 Parallel Computing

Many computer vision applications using deep convolutional neural networks have proven to be effective, but they consume a lot of storage space, memory bandwidth, and computing resources, making it difficult to implement on embedded platforms.The FPGA broke the sequential execution mode and completed more processing tasks in each clock cycle.

3.3 The features of our design

(1)SOC: Our design uses ARM+FPGA heterogeneous computing to reduce the computationally intensive part of CNN operations. Convolution and FC layer use FPGA massive parallel computing resources to accelerate and ensure real-time performance. The data flow reading and display work on the ARM ensures the flexibility and scalability of the program.

(2)Interactive: Using convolutional neural network for object recognition is generally given a picture, then the feature is extracted through convolution, finally the result is classified. In order to increase the practicality, we have an external camera on the DE10-NANO to intercept interesting objects in the form of video streams for classification.

(3)Application interface: In order to have a better display of this design, we use the VNC virtual network console remote control. VNC graphical interface is very friendly, you can intuitively see the target object detection process and screen.

3.4 The advantages of our design architecture

(1)the cascaded kernels form a deep pipeline, which can execute a serial of basic CNN operations without the need of storing interlayer data back to external memory. It significantly relieves the demand on memory bandwidth which is essential for embedded FPGAs.

(2) we use a single hardware kernel to implement both the convolution and FC layers which further improves the efficiency of hardware resource utilizations.

IV. Functional Description

In part 1,the ImageNet database was used and a number of five hundred pictures were processed on the test board. The average processing time was used as the final score.For AlexNet, the achieved classification speed is 110 ms per image as shown in Fig.4.

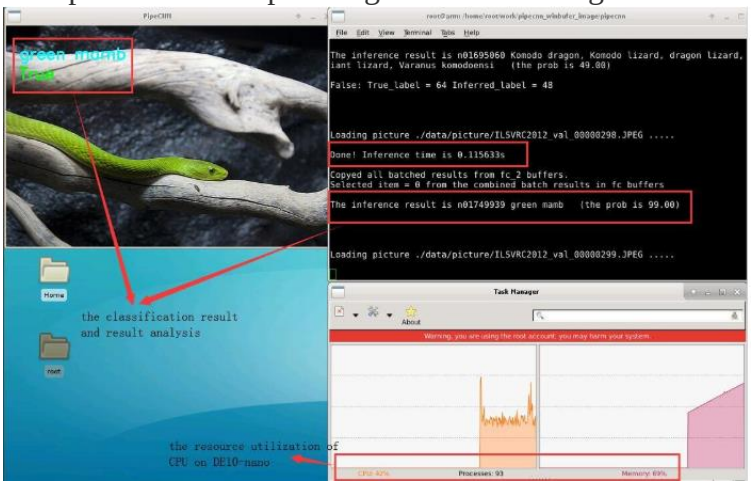

Fig.4: The result of classification acceleration experiment of imageNet

In part 2, we use a USB camera as a video input, interactively intercept a picture from the video, then parallel computing it on the DE10-NANO platform, the final real-time classification of the target object and display in the VNC interface as shown in Fig.5.

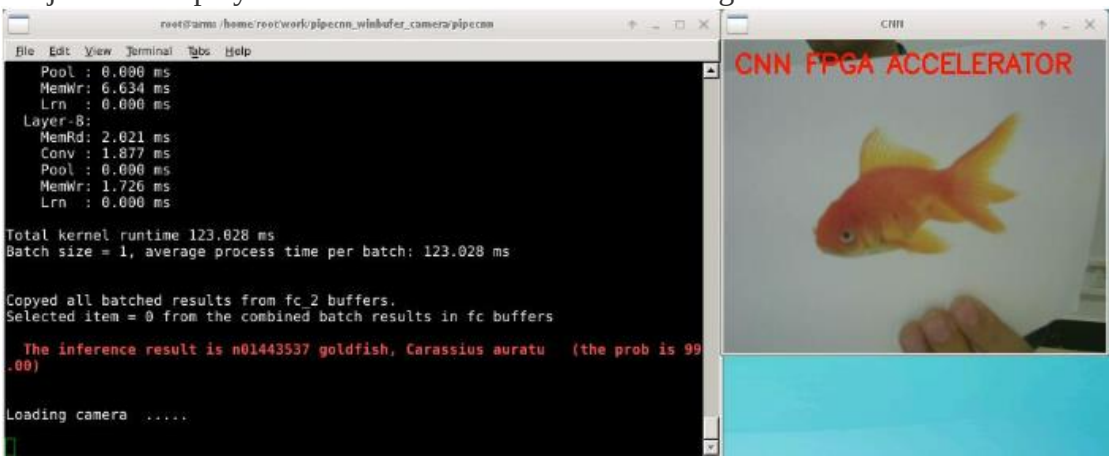

Fig.5: The result of object classification acceleration experiment via camera

In part 3, In order to prove that our acceleration system is a universal model, in this section we use the VGG-Net network for face recognition experiments as shown in Fig.6.

Fig.6：The result of face recognition acceleration experiment based on VGG-Net

In part 4, Finally, we apply our proposed accelerator in the object detection. We use Faster R-CNN to detect the target and draw it out as shown in Fig.7.

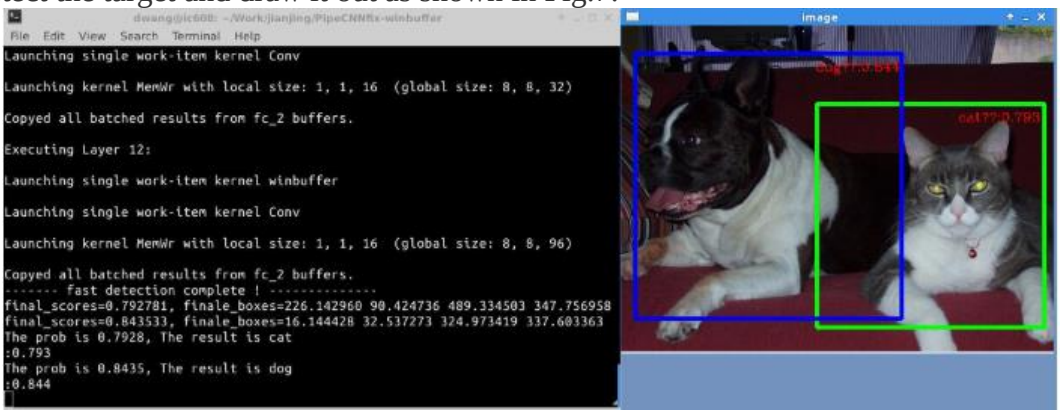

Fig.7：The result of object detection acceleration experiment based on Faster-RCNN

V. Performance metrics / goals

5.1 Throughput Optimization

To further improve the throughput of the convolution kernel, data vectorization and parallel CUs are introduced. As shown in Fig.8, the input features and weights at same the position (x, y) from adjacent feature maps are grouped as one vectorized input. The size of the vectorized data is controlled by the design parameter VEC_SIZE. The vectorized data streams are fetched by the MemRD kernel and send to multiple CUs in the Conv. kernel by OpenCL Channels as the colored line shows. The number of parallel CUs used is controlled by another parameter CU_NUM. Simply changing the value of the parameters VEC_SIZE and CU_NUM, the implemented design can achieve scalable performance and hardware cost without the need of modifying the kernel code. In the final design, two 8×8 multipliers were also grouped and mapped into one DSP block by manually inserting Altera's IP blocks in the kernel code to improve the efficiency of the pipeline.

5.2 Bandwidth Optimization

To relieve the pressure on external memory bandwidth, we introduce a sliding-windowbased data buffering scheme. As shown in Fig.7, the filter stride S of the convolution window is usually smaller than the filter size K(in most cases, S =1). Therefore, a large portion of data can be reused during the convolution computation. To exploiting data reuse, the MemRD kernel fetches a window of data that covers the area of FT_NUM of convolution filters each time, and caches the data in the on-chip buffers. And then, for successive convolution filtering operations, feature-map data and weight are repeatedly loaded from local memories avoiding access of external memory. To demonstrate the effectiveness of this scheme, we profiled the DDR memory bandwidth of implementations with different values of FT_NUM on different FPGA platforms. The average bandwidth reductions achieved reached up to 50%.

The proposed DCNN accelerator was implemented on Altera Cyclone-V SoC-FPGA based DE10-NANO board. The Cyclone-V SoC-FPGA consists of 110K logic elements(LEs),There are also an ARM Cortex-A9 dual-core processor. The OpenCL kernel codes were compiled by using Altera OpenCL SDK v16.0. A Perl script was designed to quickly perform the first-step compilation for multiple rounds with different VEC_SIZE and CU _NUM settings. For a given device, the proposed architecture achieves the maximum throughput when the convolution kernel utilizes the highest ratio of DSP resources (we estimate that the bandwidth is sufficient). Therefore, the design space was automatically explored and the one that maximized the DSP utilization for the convolution kernel was selected to complete the second-step compilation.

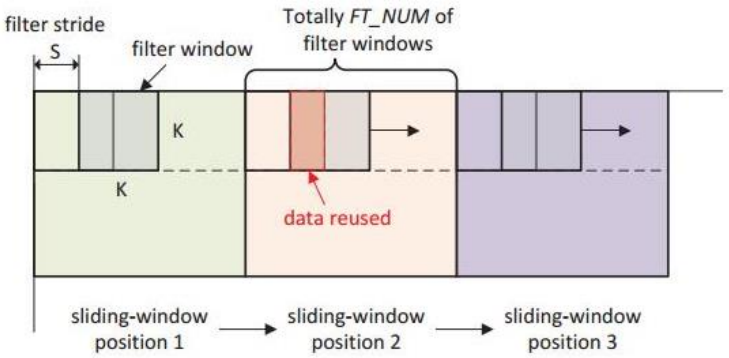

Fig.8: Sliding-window-based data buffering scheme

The final implemented design uses the setting of VEC_SIZE=8 and CU_NUM=16. Fig.9 reports the hardware resource brakedown for the five OpenCL kernels. The Conv.kernel consumes 64 DSP blocks, which is 79% of the total resource. The two data movers use a small amount of DSPs for address calculation. It can be observed from the data that our design successfully maximizes the DSP utilization while only consumes a small portion of the logic and on-chip memory resources. To measure the performance, the ImageNet database was used and a number of five hundred pictures were processed on the test board. The average processing time was used as the final score. For AlexNet, the achieved classification speed is 120 ms per image (i.e., 8.3 img/s).

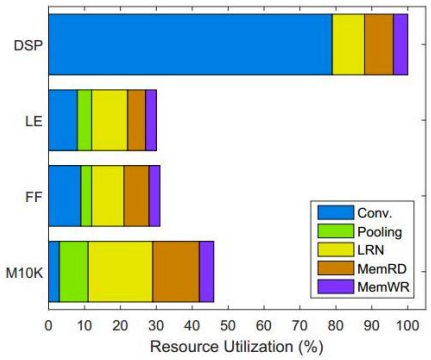

Fig.9: Resource utilization of each kernel for AlexNet model

VI. Design Method

6.1 Convolution Kernel

The convolution operation is essentially a 3-dimensional (3-D) multiply-accumulate (MAC) operation . In our design, we propose to implement by using a HLS-friendly 1-D convolution structure which flattened the 3-D convolution as follow:

$$D_o(f_o) = \sum_{f_i=1}^{C_l \times K \times K} W_l(f_o, f_i) \cdot D_i(f_i)$$

In this way, nested-loops can be avoided in kernel code, and an efficient convolution pipeline structure consisted of a multiplier-adder tree with a delayed buffer is generated by the compiler as Fig. 10 shows. When an appropriate buffer depth is selected, the proposed structure can be efficiently pipelined by the OpenCL compiler with an initial interval of only one clock cycle. Each convolution pipeline constitutes a compute unit (CU) and the kernel consists of multiple CUs to perform parallel convolutions.

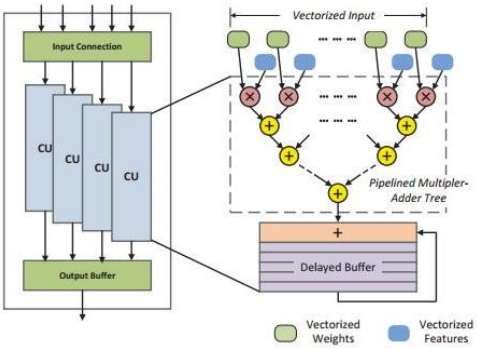

Fig.10: The hardware architecture of the convolution kernel.

6.2 Data Mover Kernels

Two multi-mode 3-D NDRange kernels are designed to fetch/store data from/to the global memory for the computation pipelines. Data and work-item mapping schemes are illustrated in Fig. 11.

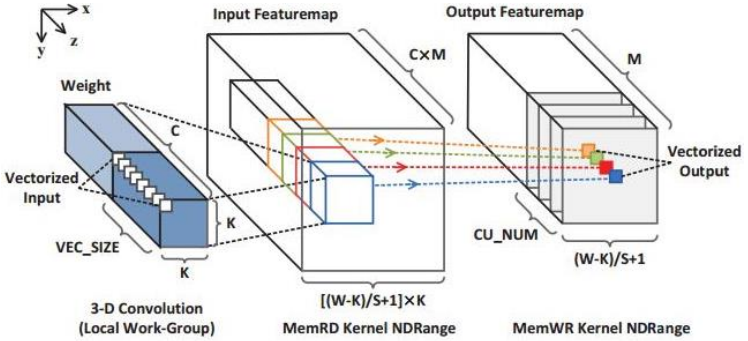

Fig.11. Data and work-item mapping scheme of the data mover kernels

6.3 Pooling Kernel

In our design, the output data stream of the convolution kernel are directly subsampled by the Pooling kernel before writing back to global memory. A line-bufferbased hardware structure is proposed for the pooling kernel as shown in Fig. 12. The kernel first reads data of the same feature maps in a line-by-line manner from the Channels and then stores them in a group of L line buffers. After all buffers are fully filled up, a pooling window of K'×K' feature map data are read out and send to the next stage with pooling logics. In DCNNs, two pooling schemes, i.e., max-pooling and average-pooling, are widely used. Therefore, the pooling logic modules support either selecting the maximum or computing the average value of the (L+1) inputs. The kernel can also be by-passed by setting a control register when a pooling layer is not required after the convolution layer.our pooling kernel design can significantly reduce unnecessary kernel-memory data transfer, and save the limited global memory bandwidth for convolution and FC layers when implemented on embedded FPGAs.

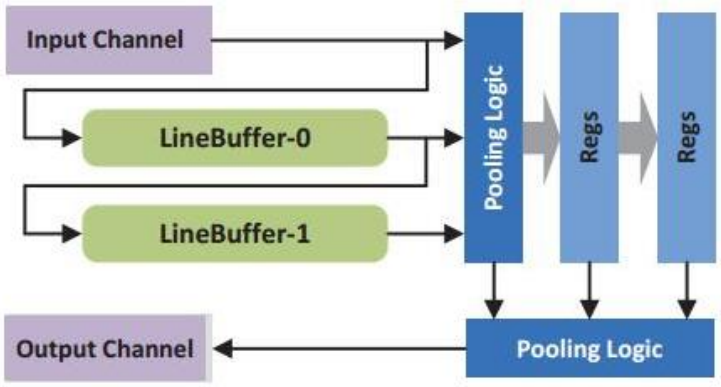

Fig. 12. Line buffer-based hardware architecture of the pooling kernel (L = 2)

6.4 LRN Kernel

Due to the wide data range required by the square and exponent operations, the LRN kernel uses floating-point data format. A piece-wise linear approximation scheme is adopted to

implement the core exponent function of the LRN kernel. We improve this scheme by introducing a new lookup table segmentation scheme to reduce the hardware costs . In this new method, the function evaluation range is divided by using a variable interval of 2^{-nx} , where x is the input index and n is an integer that controls the approximation accuracy. The approach avoids complicated table addressing logic by directly operates on the exponent of the input. The hardware parameter Shift_Bit is determined by the segmentation parameter n. For AlexNet model, a maximum approximation error of 0.5% can be achieved by setting n = 2.

VII. Conclusion

7.1 The introduce of DE10-NANO

Table.1 The introduce of DE10-NANO

FPGA Device	Intel CycloneV SE 5CSEBA6U23I7N
ARM Processor	Dual Cortex-A9
Logic Elements	110k
SDRAM	1GB DDR3
SRAM	64KB
Etherent	Gigabit
FPGA Expansion	40-Pin GPIO x2

7.2 Comparision with software accelerator on mobile CPU/GPU

Table.2 compares the performance and power consumption of the proposed design with software-based DCNN accelerators on mobile CPU and GPU. AlexNet model was used as a benchmark for all platforms. We report the power consumption by two categories, i.e., effective power and system power. The effective power refers to the actual power spent on DCNN computation and is calculated by subtracting the standby power from the total system power. To measure the power consumption of the proposed design, an external power meter was directly connected with the power supply of the DE10-NANO board. The average power consumption of the board was measured as 1.6W after Linux system boot, and as 2.1W while performing image classification. The results show that the proposed design achieves 170× and 4× speedup compared with the software accelerator on mobile CPU and GPU, respectively. The power consumption of the our design is very similar to the mobile GPU.

Table.2 the performance and power consumption on mobile CPU,GPU and FPGA

Platform	Frequency	Time	Effective power	System power
ARM Cortex A57 CPU	1.9GHz	20,767ms	2.4W	4.1W
Mali-T760 GPU	700MHz	482ms	0.52W	2.3W
Cyclone V SoC-FPGA	140MHz	110ms	0.5W	2.1W

7.3 Conclusion

This work presents a resource-efficient OpenCL-based FPGA accelerator for deep convolutional neural networks. A new architecture of deeply pipelined kernels with special data reuse and task mapping schemes are proposed. When implemented on a Cyclone-V SoC-FPGA, the proposed design achieved 4× performance improvement over start-of-the-art software accelerator on mobile GPU. The evaluation board only consumes 2.1W power while running AlexNet DCNN, which is similar to the GPU platform. Therefore, the proposed design is suitable for power-aware embedded applications like wearable devices, autonomous robots, microUAVs

and IoT systems. As the DSP resources on low-power embedded FPGAs are commonly limited, future works can be conducted to develop multiplier-free hardware architectures by using binarized DCNN model.

VIII. References

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton and et al., "ImageNet classification with deep convolutional neural networks," in Proc. Neural Information Processing Systems (NIPS'12), 2012.
[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv:1409.1556, 2014.
[3] C. Zhang, P. Li, G. Sun, Y. Guan, B. J. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15), 2015.
[4] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. F. Ma, S. Vrudhula, J. S.Seo, and Y. Cao, "Throughput-Optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'16), 2016.
[5] J. Qiu, J. Wang, S. Yao and et al., "Going deeper with embedded FPGA platform for convolutional neural network," in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'16), 2016.
[6] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie and X. Zhou, "DLAU: a scalable deep learning accelerator unit on FPGA," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016.
[7] https://github.com/doonny/PipeCNN.