

# A Resource-Limited Hardware Accelerator for Convolutional Neural Networks in Embedded Vision Applications

Shayan Moini, Bijan Alizadeh, *Senior Member, IEEE*, Mohammad Emad, and Reza Ebrahimpour

**Abstract**—In this brief, we introduce an architecture for accelerating convolution stages in convolutional neural networks (CNNs) implemented in embedded vision systems. The purpose of the architecture is to exploit the inherent parallelism in CNNs to reduce the required bandwidth, resource usage, and power consumption of highly computationally complex convolution operations as required by real-time embedded applications. We also implement the proposed architecture using fixed-point arithmetic on a ZC706 evaluation board that features a Xilinx Zynq-7000 system on-chip, where the embedded ARM processor with high clocking speed is used as the main controller to increase the flexibility and speed. The proposed architecture runs under a frequency of 150 MHz, which leads to 19.2 Giga multiply accumulation operations per second while consuming less than 10 W in power. This is done using only 391 DSP48 modules, which shows significant utilization improvement compared to the state-of-the-art architectures.

**Index Terms**—Convolutional neural network, hardware accelerator, FPGA, embedded systems.

## I. INTRODUCTION

CONVOLUTIONAL Neural Networks (CNN) are a group of deep neural networks which have shown noticeable performance in many learning applications. Their high computational complexity has slowed down their usage in industrial real-world visual pattern recognition and machine learning applications. These days, breakthroughs in computational platforms including Graphical processing Units (GPU) and Application Specific Integrated Circuits (ASIC) has let them be used in real world applications. Microsoft has used FPGA-accelerated CNNs in its databases to improve the Bing search engine [1]. This example, its high power and computational demands, and its available resources, are different from those of embedded applications and are orders of magnitude higher than them. ASIC design [2] is a power efficient way

to accelerate CNNs for embedded systems, however it has very high design and fabrication cost. FPGA acceleration as another low power acceleration method with high configurability is suitable for our case. Numerous works are introduced for FPGA acceleration of CNNs. Zhang *et al.* [3] have taken a parametric method and proposed a roofline performance model of the convolution evaluating various combinations of loop tiling and unrolling and found an optimum solution. Then, they have implemented their architecture using high level synthesis tools. Motamedi *et al.* [4] introduced a new architecture over [3] that exploits the tiling in the convolution and therefore widens the search space. They have also introduced a mathematical model for their architecture to find the parameters for the highest computation to communication ratio, and they have also estimated the performance of their design for a Virtex 7 FPGA under its nominal resources. Rahman *et al.* [5] have introduced ICAN, a 3D computation tile for convolutional neural networks, and to alleviate the low performance due to complex interconnect, they have introduced an input reuse network composed of two dimensional arrays of registers. They have evaluated their architecture targeting a Virtex 7 FPGA and have reached an operating frequency of 160 MHz, however they have synthesized the ICAN and on-chip buffers separately, and the results of the shape adapter as part of the read controller are not included in the synthesis report. Wang *et al.* [6] have introduced DLAU, a power and resource efficient FPGA based CNN accelerator. However they have only discussed the performance of their design compared to other works and have not directly presented it. In this brief, an architecture for accelerating the convolution layer in CNN is presented and implemented on a ZC706 evaluation board featuring a Xilinx Zynq XC7Z045 SoC. The main contributions of this brief are as follows:

- Proposing an architecture to implement the convolution layer in CNN that supports run time modification for image size, kernel size, and other parameters. It makes efficient use of limited off-chip memory bandwidth by loading the necessary data at once and storing it on on-chip memory. In other words, it makes complete use of the high reusability of the input and kernel data in order to limit access to off-chip memory. It has also minimal resource usage compared to similar works and maintains a relatively considerable performance. The architecture has a high performance per DSP ratio which is achieved by utilizing all of the used DSP48 resources during the operations.
- The other benefit of this architecture is its ability to use the embedded ARM processor in the Zynq SoC – the main controller of operations – for other tasks of the host embedded applications simultaneously, as controlling the flow of operations in programmable logic (PL)

Manuscript received February 21, 2017; revised March 29, 2017; accepted April 1, 2017. Date of publication April 4, 2017; date of current version September 25, 2017. This work was supported by the Iran Cognitive Sciences and Technologies Council (COGC) under Grant 2611. This brief was recommended by Associate Editor C.-T. Cheng. (*Corresponding author: Bijan Alizadeh.*)

S. Moini, B. Alizadeh, and M. Emad are with the School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran 14395-515, Iran (e-mail: shayan\_moini@ut.ac.ir; b.alizadeh@ut.ac.ir; m\_emad@ut.ac.ir).

R. Ebrahimpour is with the Department of Computer Engineering, Shahid Rajaei Teacher Training University, Tehran 16785-163, Iran, and also with the School of Cognitive Sciences, Institute for Research in Fundamental Sciences, Tehran 19395-5746, Iran (e-mail: rebrahimpour@rftu.edu).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TCSIL.2017.2690919

1549-7747 © 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.

See [http://www.ieee.org/publications\\_standards/publications/rights/index.html](http://www.ieee.org/publications_standards/publications/rights/index.html) for more information.

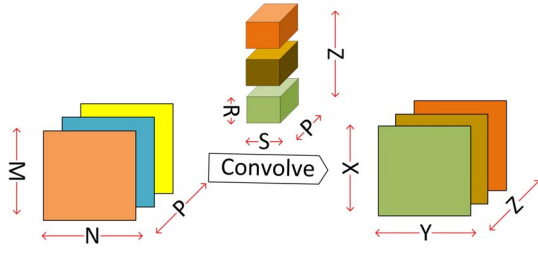


Fig. 1. A typical convolution layer.

TABLE I  
MAC OPERATIONS FOR EACH CONVOLUTION LAYER

Layer	M,N	P	R,S	Z	X,Y	#MAC
1	227	3	11	96	55	105M
2	27	96	5	256	27	224M
3	13	256	3	384	13	150M
4	13	384	3	384	13	112M
5	13	384	3	256	13	74M
Total						703M

is interrupt-driven and takes a very small amount of time and performance load for the processing system (PS) (see Section V).

The rest of this brief is organized as follows. In Section II, an overview of theory of convolutional neural networks is presented. In Section III, the proposed architecture and its acceleration methodology is explained in details. In Section IV, the implementation and experimental results are reported and Section V concludes the brief.

## II. CONVOLUTIONAL NEURAL NETWORKS (CNN)

Convolutional Neural Networks are a group of deep learning artificial neural networks inspired by mammalian visual system. A typical CNN in object recognition consists of a number of layers where each one operates on input feature map from the previous layer. The input of CNN is the target image and the output is an array used to classify the image. The main operations of the CNN are convolution, non-linear function, pooling, and normalization.

The main focus of this brief is on convolution layers where most of the computational load is concentrated. A convolution operation is composed of a set of 3D arrays called kernels where each kernel is applied to all the input feature maps with a 3D convolution operation and a sliding mechanism to generate one output feature map. For instance, as depicted in Fig. 1, let us consider an input consisting of  $P$  feature maps, each with  $M$  pixels in length and  $N$  pixels in width, a set of  $Z$  kernels, each with  $R$  pixels in length and  $S$  pixels in width, and  $P$  pixels in depth - equal to the number of feature maps. There are  $Z$  output feature maps each with length  $X$  and width  $Y$ . The total number of multiply accumulate operations in such a system is equal to  $X \times Y \times Z \times R \times S \times P$ . The pixels in the set of kernels are called weights. The main focus of this brief is on accelerating pre-trained CNNs with the necessary weights stored in off-chip memory, operating at run-time, receiving images as inputs, and delivering output arrays to classify the subject.

### A. Alex-Net

A rather well-known CNN, known as Alex-Net, is proposed in [7] which has been accelerated in many works [3]–[5]. Alex-Net has 5 convolution layers. The number of Multiply and Accumulate (MAC) operations and relevant parameters in

```

For z = 1 : Z
  For x = 1 : X
    For y = 1 : Y
      For r = 1 : R
        For s = 1 : S
          For p = 1 : P
            Out[x][y][z] += Input[r+xt][s+yt][p] * kernel[r][s][p][z];
          
```

Fig. 2. Typical operation in convolution layer.

```

For x = 1 : X
  For y = 1 : Y
    For r = 1 : R
      For s = 1 : S
        For p = 1 : P {
          Out[x][y][z1] += Input[r+xt][s+yt][p] * kernel[r][s][p][z1];
          Out[x][y][z2] += Input[r+xt][s+yt][p] * kernel[r][s][p][z2];
          ...
          Out[x][y][zn] += Input[r+xt][s+yt][p] * kernel[r][s][p][zn];
        }
      
```

Fig. 3. Our parallelisation scheme.

each convolution layer is listed in Table I.  $M$  is an abbreviation for Millions. As shown in Table I, the total operations of convolution layer is equal to more than 700 million MAC operations which constitutes 90% of all the computations in the CNN, making it a suitable candidate for acceleration.

## III. ACCELERATION METHODOLOGY AND ARCHITECTURE

### A. Data Reuse Factor

An operation chosen as a candidate for hardware acceleration needs to have a high data reuse factor – defined as the number of operations that a piece of input data is used for, before it is replaced by the next data. This factor is very small for the pooling layer. For non-linear function and fully connected layers it is equal to 1, meaning that the main advantage of hardware acceleration over single thread CPU – which is parallelizing operations with common data – is of little use in these operations. In convolution layer, this factor is very high and therefore parallelization improves its performance. The first layer of Alex-Net has a data reuse factor of 3025 – in this layer each weight of each kernel is used in all pixels of the corresponding output feature map which counts to  $55 \times 55$  or 3025. Therefore this brief is concentrating on accelerating the convolution layer.

### B. Acceleration Methodology

The goal is to reduce the slow off-chip memory access. Each pixel of input feature map and kernel data is loaded only once and stored on on-chip memory until it is of no use. Fig. 2 shows the typical operation of the convolution layer. For all output feature maps residing in spatial position  $(x,y)$  and a different feature map  $(z)$ , a similar part of input feature map is used for calculations and only the kernels are different. Therefore a number of modules can be devised to receive the common input feature map data and their corresponding kernel to generate the output. This parallelization scheme is used in the present brief and as shown in Fig. 3, at each run, pixels in the same spatial position of different output feature maps are calculated simultaneously. The reason for unrolling the  $Z$  loop in Fig. 3 is that according to Table I, this parameter and the  $P$  parameter stay significantly large as the layers go forward, unlike other parameters that shrink due to the pooling operation and small kernel size. The problem with unrolling  $P$  parameter is that the shared input data would not be present in that case. Therefore the  $Z$  parameter is the best parameter for loop unrolling.

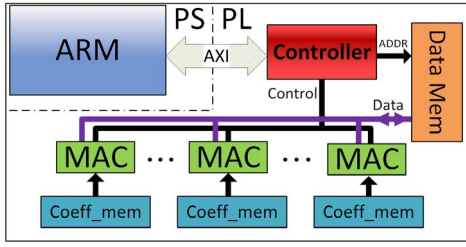


Fig. 4. The proposed architecture at run-time.

### C. Proposed Architecture

Let us consider the operation of the convolution layer when calculating the output pixel in feature map number  $i$  positioned in length  $x$  and width  $y$ . This pixel and its adjacent pixel in another output feature map are calculated simultaneously. The input feature map data is equal and kernel data is different. As shown in Fig. 4, the architecture at run-time is composed of a controller in the PL side, which is connected to a data memory, feeding the data of input feature maps to a group of MAC modules. Each MAC module is used to calculate the pixel for its own output feature map and has a small memory called ‘Coefficient Memory’ that contains the corresponding kernel. The MAC module uses the input data from the data memory and kernels from the coefficient memory to calculate an output pixel during a run. At the same time, the results of the MAC modules are stored in the data memory which is deep enough to store both input and output feature maps for different convolution layers with different parameters and input and output feature map sizes. After all the pixels of an output feature map are calculated, a cycle is finished.

The main controller of the whole system is the ARM processor in the PS side. Since the architecture is used for different types of convolution operations, each with different parameters, the controller in PL side needs to be configured for different kernel and feature map sizes and numbers in run-time. These parameters are fed to the controller by setting certain registers, provided by the ARM processor, and are used by the PL-side controller to calculate necessary addresses. These parameters include the number of input feature maps and kernel sets, their length and width, the number of pixels the window moves after calculating each output pixel called stride, and the address to store output in data memory in order to prevent it from overwriting the previously stored input feature maps— as they are needed for calculating all the output feature maps.

Fig. 5 shows the memory connectivity of off-chip and on-chip memories. The off-chip memory is a 1 Gigabyte DDR SDRAM memory. This memory is accessible to the PL through four AXI high performance ports provided by a DDR memory controller. Each port has an AXI CDMA (Central Direct Memory Access) for transferring data between off-chip and on-chip memory – data and coefficient memories – with minimal load on the ARM. Fig. 6 shows the detailed architecture in PL side. The width of data and coefficient memories is equal to 32 bits. All operations in this work are in Q15 fixed point domain, which is an enough precision at run-time [8], with kernels being 16 bit values with 15 bits for fraction and 1 bit for sign as they are normalized between 1 and -1. The input and output feature maps are 32 bits with 16 integer bits that prevent from overflow. Each row of data memory is used to store one data value, while each row of the coefficient memory is used to store two 16 bit kernel values. Since on-chip memories are true dual port block RAMs, two rows of memory

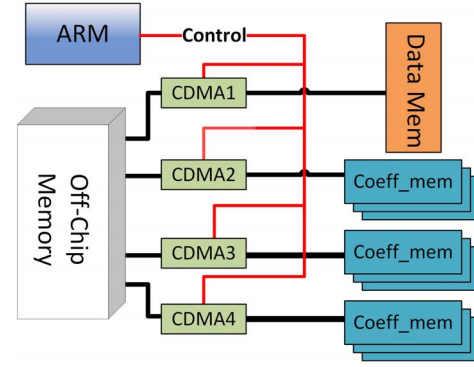


Fig. 5. Overview of off-chip memory connectivity.

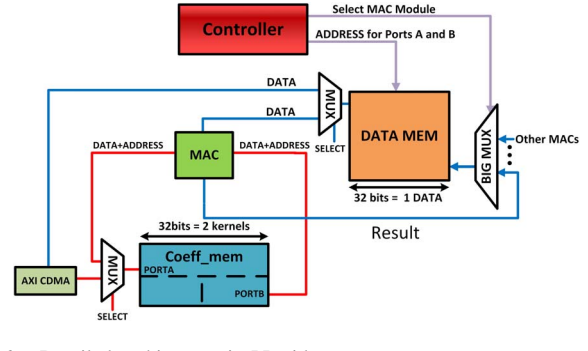


Fig. 6. Detailed architecture in PL side.

can be accessed each time; therefore, by storing four kernels in each coefficient memory, two in the first half and two in the second half of the memory, at each run, a MAC module can generate four output pixels for four different feature maps. As shown in Fig. 6, one port of data memory is multiplexed between AXI CDMA and controller, and also one port of each coefficient memory is multiplexed between AXI CDMA and the corresponding MAC module. This is because of the limited ports connecting these memories. The multiplexer is controlled by the ARM through AXI GPIO. One port of data memory is used to feed pixels of input feature maps to all of coefficient memories with address provided by the controller. Once a run is complete and the result of each MAC module is ready, it is stored in data memory through its second port. Since all the MAC modules are connected to this single port, they are multiplexed by a ‘Big Multiplexer’ which is controlled by the controller and selects the correct MAC modules to store their output one after another. Since feeding of input feature maps’ data and storing of results of each output feature map are done simultaneously, the process is not interrupted until a cycle is completed.

The order of operations is as follows. First, the ARM processor sets the multiplexers to connect the on-chip memories to AXI CDMA, then it programs the CDMA to load the input feature maps’ data to the data memory and the kernel data to coefficient memories. Then, the ARM sets the PL-side controller registers with suitable parameters for the convolution operation in hand and afterwards sets the multiplexers to connect data memory to controller and coefficient memories to the corresponding MAC modules. After that, the MAC modules start calculating the output feature map pixels. After a cycle is finished, the ARM takes control again and sets the multiplexers to connect on-chip memory to off-chip memory and transfers the results of the convolution operation from data memory to off-chip memory. These data transfers are done



TABLE II  
NUMBER OF OPERATIONS AND TIME IN ACCELERATOR

Layer	1	2	3	4	5
Cycle	1M	1.7M	1.2M	1.6M	1.6M
Time (ms)	7.3	11.6	7.8	7.6	3.8
CPU time (ms)	17.3	19.4	8.7	13.3	9.3
#MAC (M)	105	224	150	112	74
Perf.	14.3	19.2	19.2	14.7	19.2

using AXI CDMA's in scatter gather mode where the DMA automatically performs the transfer and interrupts the ARM after the last transaction is completed. Therefore the total load on the ARM processor is interrupt based and minimal during the whole operation.

#### IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

##### A. Implementation

The proposed architecture was coded in Verilog and synthesized and implemented targeting a Zynq 7000 ZC7045 SOC included in the ZC706 evaluation board. Vivado 2015.4 was used as synthesis and physical design tool. The implemented design is fully pipelined and has successfully reached 150 MHz of clock frequency. The final design contains 32 Mac modules that each accompanied with one 18 KB coefficient memory and a Data memory of 1.7 MB sufficient for accelerating all convolutional layers of Alex-Net [7]. Each MAC module contains 12 DSP48 modules used to generate 4 output pixels at each run which leads to a total maximum performance of 19.2 Giga Multiply Accumulation operation per second. The limiting factor for this number of MAC modules is available on-chip memory resources. The design contains four Central DMA's connected to off-chip memory via the high performance AXI interface generating a total throughput of 2.4 GB/s sufficient for the operation of this design. It should be noted that this bandwidth is not used all the time and is only used when one cycle is finished and the result is being transferred from the data memory, and the coefficient memories are being updated. The off-chip memory is free at the run time and can be accessed by the ARM processor for other applications.

##### B. Experimental Results

We have used the implemented architecture to accelerate all the convolution layers of Alex-Net [7]. Details of the layers are mentioned in Table I. The accelerator is composed of 32 MAC-modules each able to calculate 4 output pixels in each run which leads to 128 output feature maps calculated after the whole cycle is finished. Three CDMA's are connected to these coefficient memories, 2 of them each connected to 11 coefficient memories and the third connected to 10 coefficient memories. The final CDMA is dedicated to data memory and is used to load input feature maps' data and also collect the results after a cycle is completed. For layer 1 which has 96 output feature maps, one cycle is sufficient. The term run is used for each time an output pixel is calculated and the term cycle is used for each time an output feature map comprising of numerous pixels is calculated. For layer 2, there are 256 output feature maps where each half of them uses one half of input feature maps, so this layer needs two cycles to calculate output feature maps and between the two cycles the input feature maps needs to be refreshed. For layer 3, we have 384 output feature maps which are calculated in 3 cycles. Layer 4 has 384 output feature maps, however each half of these feature maps uses half of input feature maps, therefore after calculating 192 feature maps during 2 cycles, input

data needs to be refreshed, so a total of 4 cycles is needed. Finally layer 5 needs 2 cycles for a total of 256 output feature maps. Table II shows the number of clock cycles (*#Cycle*), run time in milliseconds (*Time*), million number of MAC operations (*#MAC*), and the performance for each layer. The final row (*Perf.*) shows the performance in terms of Giga MACs per second. As can be seen, a maximum performance of 19.2 Giga MACs per seconds can be achieved. Note that *CPU time* row shows the amount of time taken for an Intel Core™ i5 M540 CPU to run each layer which will be discussed later.

##### C. Discussion

One important point to be note here is that the above mentioned performance is achieved using only 391 DSP blocks. While other architectures may achieve better performance with the cost of using more DSP blocks. In order to have a fair comparison between the proposed architecture and existing ones, we have defined *Perf/DSP* as the mean share of each DSP48 block in the total performance calculated by (1).

$$Perf/DSP = Performance(GOPS) \div DSP Utilization \quad (1)$$

The *DSP Util*, *LUT Util*, and *BRAM Util* rows of Table III show DSP, look up table, and Block Memory utilization of our and others' implementations. The Block RAM usage of our design is equal to 100%. This is because our design has made use of all on-chip memory available and also a part of Distributed memory available for data memory and coefficient memory in order to limit slower access to DDR off-chip memory. However, our design uses a considerably fewer number of on-chip block memories compared to other implementations which shows a smart conservation of memory resources. This is less true for the utilization of LUTs where Rahman *et al.* [5] have shown less utilization than ours. However the on-chip memory is a less abundant resource than LUT (LUT consumption is 9.2% and BRAM consumption is 52.7% in [5]). In Table III, *Perf/DSP* row shows the mean share of each DSP48 block in performance. The results show significant performance to DSP ratio improvement compared to [3] and [5]. Consequently, our architecture achieves more performance using fewer DSP48 blocks which enables the design to be implemented on low-end platforms with less resources. At the same time, it improves the timing by decreasing the congestion resulting from a high number of DSP48 blocks and therefore increasing the operating frequency, and finally the total power consumption of the architecture is decreased due to a fewer number of DSP48 blocks and block memories running at the same time. Another advantage of this architecture, and FPGA based systems in general, is their low power consumption. The power consumption of the system was calculated to be 10 watts at run-time using Xilinx Power Estimator (XPE). Also *CPU time* row in Table II shows the amount of time taken for a PC running Intel Core™ i5 M540 CPU to run each layer. As Shown, our architecture has a better performance compared to the PC, while consuming much less energy. The CPU has an idle power consumption of 45 watts. Another platform which has implemented Alex-Net, GPU platform with 235 watts of power consumption [9] is too power hungry to be considered for embedded vision systems. This is less true for ASIC platform where the custom design, decreases power consumption, however the high design and deployment cost, time, and effort, makes ASIC an unfavorable platform for embedded systems.

TABLE III  
PERFORMANCE AND PERFORMANCE TO DSP RATIO COMPARISON WITH PREVIOUS WORKS

Architecture	PACT2010[10]	ISCA2010[11]	ICCD2013[12]	ISFPGA2015[3]	DATE2016[5]	Our Work
Precision	Fixed point	48bit fixed	Fixed point	32bit float	32bit fixed point	Q15
Frequency	125MHz	200MHz	150MHz	100MHz	160MHz	150MHz
FPGA Chip	SX240T	SC240T	VLX240T	VX485T	VX485T	XC7Z045
CNN Size	0.53 GMAC	0.26 GMAC	2.74 GMAC	1.33 GFLOPS	1.33 GFLOP	1.33 GFLOPS
Performance	7 GOPs	16 GOPs	17 GOPs	61.62 GFLOPS	80.78 GFLOPS	38.4 GOPs
DSP Util.	N/A	N/A	N/A	2240	2688	391
Perf/DSP	N/A	N/A	N/A	0.027	0.030	0.098
LUT Util	N/A	N/A	N/A	186251	28000	77442
BRAM Util	N/A	N/A	N/A	1086	1024	545

Sufficient off-chip memory bandwidth is another bottleneck in embedded systems. The proposed architecture connects to off-chip DDR memory during loading the image and kernel data or storing the results which makes a small fraction of total time. It also uses 2.4 GB/s of bandwidth during loading and storing data which is a reasonable bandwidth for an embedded system (DDR off-chip memory can support up to 6.4 GB/s of bandwidth). The main ARM controller can use the extra feasible bandwidth for other operations, or make use of the whole bandwidth when the accelerator is running on the PL side and off-chip memory is free.

According to Table II, the total time it takes the accelerator to execute all the five convolution layers of the Alex-Net is equal to 38 milliseconds. This time should be added to the time it takes to load and store data which is less than 5 milliseconds which results in 43 milliseconds. Considering 10 images per second as a suitable criterion for real-time embedded vision applications, this architecture is a suitable accelerator for a complete CNN based pattern recognition algorithm (Alex-Net as an example) whose main application runs in the ARM processor and the extra time (57 milliseconds left of the whole 100 milliseconds) is used to calculate the rest of the less computationally expensive operations for CNN (pooling, normalization, non-linear function, fully-connected) either on ARM as software, or accelerated on PL side (explained in future work section).

Comparing the frequency of this brief to that of [3] and [5] must be done considering their Virtex 7 FPGA platform which is a high end device with better timing and switching characteristics than our Zynq 7000 platform. Furthermore, Rahman *et al.* [5] have not included the shape adapter - which is implemented in read controller - in the synthesis report where otherwise, could degrade their timing results. As it is shown in Table III, our implementation shows the highest performance to DSP ratio ( $Perf/DSP$ ) which is due to the non-stop use of DSP modules at run-time. It is worth noting that if our resource bottleneck which is on-chip memory is compensated by replacing the platform with Virtex 7 whose on-chip memory resources is triple that of our ZYNQ 7000 SoC, it would surpass the previous works in the performance. Note that, for [4], the implementation results are not available and therefore it is not compared to this brief. For [10]–[12] DSP utilization is not reported (Not Applicable: N/A), however their performance is far below ours. Performance is converted from GMACs to GOPs by multiplying by 2, since each MAC operation consists of an addition and a multiplication operation. Our system has a performance of 38.4 GOPs. The low power consumption of the system at run-time, calculated to be less than 10 watts, is suitable for various power limited embedded applications.

## V. CONCLUSION

In this brief, a hardware accelerator for the convolution layer of CNN was proposed. The accelerator makes complete use

of high data reuse factor of input feature maps and kernels and minimizes access to off-chip memory. The resource utilization is minimal compared to other works. The architecture maintains dominant performance to DSP ratio by using all the instantiated DSP modules during the whole span of run time. The architecture was implemented on a ZC706 board with a frequency of 150 MHz and Memory Bandwidth of 2.4 GB/s and a power consumption of less than 10 watts. The performance reaches to 38.4 GOPs that combined with low power consumption and moderate bandwidth and resource usage, is a suitable candidate for embedded applications where the ARM processor can run other processes at the run-time. Our future work is to evaluate the ARM processor against custom hardware accelerator in other operations of CNN to find the best candidate for their implementations and finally run the complete flow of CNN on a SoC in real-time as required by embedded vision systems.

## REFERENCES

- [1] A. Putnam *et al.*, “A reconfigurable fabric for accelerating large-scale datacenter services,” *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 13–24, 2014.
- [2] A. Pullini *et al.*, “A heterogeneous multi-core system-on-chip for energy efficient brain inspired computing,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, to be published.
- [3] C. Zhang *et al.*, “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” in *Proc. ACM/SIGDA Int. Symp. Field-Programm. Gate Arrays*, Monterey, CA, USA, 2015, pp. 161–170.
- [4] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, “Design space exploration of fpga-based deep convolutional neural networks,” in *Proc. Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2016, pp. 575–580.
- [5] A. Rahman, J. Lee, and K. Choi, “Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array,” in *Proc. Design Autom. Test Europe (DATE)*, Dresden, Germany, 2016, pp. 1393–1398.
- [6] C. Wang *et al.*, “DLAU: A scalable deep learning accelerator unit on FPGA,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 36, no. 3, pp. 513–517, Mar. 2017.
- [7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.
- [8] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.
- [9] K. Ovtcharov *et al.*, “Accelerating deep convolutional neural networks using specialized hardware,” Microsoft Research Whitepaper, 2015.
- [10] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar, and H. P. Graf, “A programmable parallel accelerator for learning and classification,” in *Proc. Int. Conf. Parallel Archit. Compilation Tech.*, Vienna, Austria, 2010, pp. 273–283.
- [11] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, “A dynamically configurable coprocessor for convolutional neural networks,” in *Proc. Annu. Int. Symp. Comput. Archit. (ISCA)*, St.-Malo, France, 2010, pp. 247–257.
- [12] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, “Memory-centric accelerator design for convolutional neural networks,” in *Proc. Int. Conf. Comput. Design (ICCD)*, Asheville, NC, USA, 2013, pp. 13–19.