IJICTR
International Journal of Information & Communication Technology Research

Volume 15- Number 2 – 2023 (12 -18)

A Partial Method for Calculating CNN Networks Based On Loop Tiling

Ali A.D. Farahani
School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran

Hakem Beitollahi*
School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran
Beitollahi@iust.ac.ir

Mahmood Fathy
School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran

Reza Berangi
School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran

Received: 12 October 2022 – Revised: 1 November 2022 - Accepted: 1 January 2023

Abstract—**Convolutional Neural Networks (CNNs) have been widely deployed in the fields of artificial intelligence and computer vision. In these applications, the CNN part is the most computationally intensive. When these applications are run in an embedded device, the embedded processor can hardly handle the processing. This paper implements loop tiling to explain how one can construct a lightweight, low-power, and efficient CNN hardware accelerator for embedded computing devices. This method breaks a large CNN engine into small CNN engines and calculates them by low hardware resources. Finally, the results of small CNN engines are added and concatenated to construct the large CNN output. Using this method, a small accelerator can be configured to run a wide range of large CNNs. A small accelerator with one layer is designed to evaluate our methodology. Our initial investigations show that based on our methodology, the constructed accelerator can run a modified version of MobileNetV1, 70 times per second.**

Keywords: Convolutional neural networks (CNNs), Hardware Accelerator, Embedded system, Low Power.

Article type: Research Article

Publisher: ICT Research Institute

* Corresponding Author

[DOI: 10.61186/itrc.15.2.12]

I. Introduction

Convolutional neural networks are used in computer vision applications widely [2]. Artificial intelligence and deep learning use CNNs in their applications [3]. CNNs are the most computation-intensive part of all vision networks, such as ResNet, MobileNet, VGG, and AlexNet [1]. The processing of CNNs has two different phases training and inference. Only the inference phase could be run on small embedded processors, and training should be run exclusively on powerful GPUs [3].

Using GPUs is a simple option for inference; however, it is too expensive and power-hungry. Also, powerful GPUs may not be physically available anywhere, and these powerful GPUs provide their computing capability as a cloud service to all devices that require it [1]. However, access to these services has several problems, including extra cost, extra power for wireless or 5G interfaces, continuous bandwidth requirements, delay in sending and receiving responses, network disconnection possibility, Etc. Therefore, there is a solid demand in the semiconductor industry for a tiny co-processor or accelerator to run CNNs efficiently beside small and embedded processors at the edge.

The main contribution of this work is that we convert the loop tiling method into a new partitioning scheme in hardware implementation. Using this scheme in hardware, a small fixed CNN accelerator engine, as a basic unit, can calculate a wide range of large CNNs in the above networks, partially and in a step-by-step manner. This methodology is hardware implementation or dual loop tiling method that was previously used in lots of software implementation of CNN execution in small processors. The designed CNN accelerator is as small as that can be integrated into a low-power embedded processor.

The rest of the paper is organized as follows. Section II discusses the related works. Section III provides a background on CNN networks. Section IV states the problem statement. Section V proposes the partitioning method. Section VI presents the experimental results and finally section VII concludes the paper.

II. Related works

Many academic and industrial kinds of research have been done on convolutional network accelerator hardware. Many of these research projects focus on accelerating convolutional networks on FPGAs. Some accelerators are only designed for a specific convolutional neural network [7]. In [5], the team has developed a software-hardware template that can generate the appropriate hardware convolutional network as well as the software components (if needed) by receiving the Python code [16].

Others focus on faster access to memory and optimal use of memory to reduce the memory bottleneck effect and thus increase the accelerator performance [8]. In [18] an analytical model has been proposed to find the best loop-level optimization configuration, including loop tiling and loop permutation for CNNs on multi-core processors. Loop level optimization (loop tiling and loop permutation) is an essential transformation to reduce data movement and provide an efficient memory architecture in CNNs.

[17] presents an enhanced version of loop-tiling in the software manner. This technique utilizes various combinations of affine and non-affine loop transformations to find the best transformation sequence for minimizing the CNN execution time. To find the best combination, a cost model evaluates the speedup of each sequence of transformations, which would yield.

Most research studies have used Depthwise Separable Convolution networks to reduce hardware size and speed up the execution [7], [9]. In [16], a CNN accelerator with a 14 × 16 processing element (PE) array is designed and then utilized in a loop tiling structure and Ping-Pong operations to efficiently transmit feature maps from the on-chip buffer to the PE array. Moreover, a roofline model is used to explore the best tiling parameters. Additionally, this technique has been implemented on the FPGA.

RASHT [19] is a scalable architecture that resizes PEs to match any layer shape of CNN layers. The main idea behind RASHT is that a CNN network consists of different layers of different sizes. Instead of designing a fixed PE engine for all layers, the engine resizes itself based on the layer it operates on it.

III. Convolutional Neural Networks (CNN)

Almost all CNNs contain from a few ten to a few hundred layers of convolutional [3]. In CNNs, the last layer is converted to a flat matrix and fed to the input of fully connected layers. Then, a fully connected neural network is executed in one or two layers on this data for classification purposes. Finally, a nonlinear function analyzes the outputs of this neural network, and the most likely classes are selected as the network output. Therefore, most of today's AI networks have a CNN network similar to Fig 1. In the first layers, the resolution of feature maps is usually high, but the number of kernels (filters) is relatively low. In the final layers, the number of kernels is between 128-2048, but the resolution is decreased because of several pooling layers.

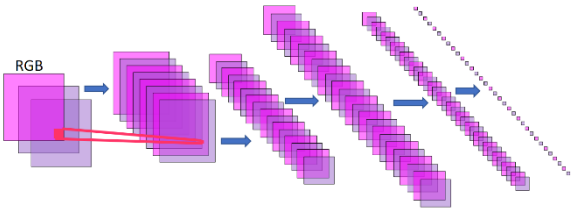

Figure 1. A sample CNN network

CNNs are widely used in vision applications [1], [2], [3]. The input layer (layer 0) in a vision application is an RGB image. The first layer is generated directly by convolving kernels and the input image. Step by step, the other layers of feature maps are generated by convolving kernels with previously generated feature maps. Every convolution operation has several MAC (Multiplications and Accumulate) operations, including some element-by-element multiplications, followed by adding those multiplication products. Besides convolution, there are other layers in CNNs: Mean or

[DOI: 10.61186/itrc.15.2.12]

Max Pooling, ReLU, addition, and normalization, that are widely used in CNNs.

There are many mathematical operations in the convolutional layers, so even for desktop and server processors, executing a CNN is a heavy computational task. For instance, AlexNet, Resnet, and VGG-16 include 724 million, 3.9 billion, and 15.5 billion MAC operations, respectively [11]. There are two common simplifications used in CNNs to reduce the number of MAC operations and decrease the computation load: (1) Compression of CNN models [10] and (2) simplification of CNN models by replacing standard convolution with Depthwise Separable Convolution [6]. In Depthwise Separable Convolution, the convolution operation is divided into two parts with a smart kind of factorization [4]. As a result, this factorization effectively reduces the volume of mathematical operations. Due to the sharing of calculations, the parameters of these calculations (multiplication coefficients and biases) are also shared, reducing the number of network parameters [4]. The compression of CNN models tries to reduce the number of kernels by removing redundant feature maps and, consequently, redundant MAC operations [10].

MobileNetv1 is one of the first vision networks to use depthwise separable convolution. Using this convolution type, the needed operation for each image is reduced to as low as about 1 billion MAC instructions [6]. Another light vision model, ShuffleNet, uses the depthwise technique and has about 0.5 billion MAC instructions [15].

IV. Problem Statement

The methodology of designing a lightweight and low-power CNN accelerator for embedded systems as a general-use case is our primary objective in this paper. To this end, a one-layer CNN accelerator is designed with minimum hardware resources. This CNN accelerator can run only one Depthwise CNN layer at a time.

CNNs have different layers of convolutions. There are several kernels in each layer of the convolution. Usually, the number of kernels is mainly a power of two, i.e., 16, 32, 64, 128, 256, 512, 1024, Etc. Usually, the number of kernels in the first layer is 16 or 32. This number gradually increases layer by layer when we go through the network. Each CNN accelerator has several input feature maps and several output feature maps. Consider a convolutional network whose first layer has 16 kernels and its last layer has 128 kernels; as the CNN accelerator runs the convolutional network, step by step, the CNN accelerator should support 128 feature maps as input and could generate 128 feature maps as output. Otherwise, this CNN accelerator cannot run the last layers. Note that a 128x128 CNN accelerator is a huge CNN accelerator that is not considered to be used in an embedded processor. If a large CNN accelerator is designed, it can run a wide range of CNNs; however, the use of a large CNN accelerator increases the hardware resources and the chip area, power consumption, and cost. This kind of design eventually leads to a significant decrease in the circuit's speed of execution and frequency. Moreover, some parts of the hardware are actually unused in the layers of convolutional networks where the number of kernels is less than the maximum number of kernels predicted for the accelerator. Therefore, the resource usage and the efficiency of the CNN accelerator are decreased.

It is known that in embedded systems, the chip area, size, and power consumption of any accelerator must be small as possible. However, the use of a small CNN accelerator leads to a limited number of input and output feature maps in the CNN accelerator. In this situation, it is impossible to calculate the convolution in that network layer whose number of kernels is greater than the predicted number in the CNN accelerator. Thereby, it leads to a useless accelerator because it cannot run most convolutional networks. Due to the problems stated above, we have used loop unrolling hardware implementation of the loop tiling technique [13, 14, 18].

Loop tiling is used in executing the CNN network over some relatively small processors that can be embedded in chips. This type of execution of CNN networks is called “software implementation” [17]. It means that dedicated hardware for CNN execution is not designed. The small processor executes each layer of CNN. When the execution of the layer is finished, all the result is kept in memory. The result of the previous layer is used as input for the next layer in the convolutional layer. Therefore, memory should have enough capacity to keep at least the two biggest CNN layers in the worth case plus the required parameter in the multiplication of this layer. It is about 1-2 Mbyte in a relatively small CNN network that is designed for mobile and embedded applications.

To execute a CNN on a processor, the cache capacity of the processor is an essential issue that should be considered. Processors could execute over a large amount of data in their cache very quickly (Fomula.1). However, if the required data of execution are out of the cache and in the main memory, then a long delay is unavoidable due to the fetch operation, which slows down the processor.

For a = 0 to A-1:
  For b = 0 to B-1:
    For c = 0 to C-1:
$$X(a,b,c) = \sum_{j=0,k=0}^{2,2} X(a+j, b+k,\ c) * M(a)[k, j]$$

Fomula.1. Original Convolution

The above pseudocode (Formula.1) shows a typical calculation for computing one feature map over previous feature maps in the last layer, where A is the number of feature maps, and B and C are the input image sizes. M represents the parameters of each convolution.

The calculation of each CNN feature map in a new convolution layer in a processor is a nested loop, as shown above. This calculation should be repeated for each feature map in the new convolution layer over all feature maps in the last convolution layer. A convolution layer with 224x224 resolution and 256 feature maps has 800 Kbytes integer data (number) and about 70 Kbytes integer parameters. If the processor cache is about 256 Kbytes, not all the needed data can be placed in the cache, and logically several consequent

[DOI: 10.61186/itrc.15.2.12]

cache misses are happened during the processing because of the capacity cache miss.

To overcome this problem, the above loop is broken down to several small loops with less than 64K bytes and can hold all in the cache, and the amount of cache missed is kept as low as possible to speed up the execution of the CNN network. Each small loop is called a "tile," and the overall technique is "loop tiling." Then the result of each tile is added together and sometimes re-arrayed to build the originally required output, that is, a convolution output.

The Tiling method divides the convolution into several smaller convolutions (Formula 2). In the software implementation, the parameters and data of these small convolutions can be stored in the processor's cache. Finally, based on the shape and size of the original convolution, several small convolutions are combined, and the big (original) convolution is rebuilt.

In Formula 2, it is assumed that a large convolution has the number of feature maps equal to FM_D, and the size of the parameters and data of the convolution is larger than the processor's cache. The processor's cache can only hold the parameters and data of a convolution length L_D. However, by breaking the big convolution and making it smaller by the number of $(1 + FM_D/L_D)$, the smaller convolutions are quickly calculated in the processor.

f or i = 0 to $(\frac{FM_D}{L_D})$:
  for a = 0 to A-1:
    for b = 0 to B-1:
      for c = 0 to $L_D - 1$:
        $X(i, a, b, c) = \sum_{j=0,k=0}^{2,2} X(i, a + j, b + k,\ c) * M_i(a)[k, j]$

Fomula.2. Small Convolution made from Original Convolution.

In a software implementation, it is clear that loop tiling needs more memory for storing tiles for future use but effectively reduces cache miss and keeps the processor at the high-speed execution.

In the hardware implementation, the cache miss problem faces like a limitation of input feature maps and output feature maps in accelerators because of using the small accelerator. It also appears as a limitation in the size of on-chip memory. Therefore, breaking down the CNN to small CNN, like the concept of tiling, could be useful, especially in edge and embedded systems with several limitations in memory, area, power, Etc. This technique (loop tiling) is called CNN partitioning in this paper. By using CNN partitioning, it is possible to use a small-size hardware-based CNN accelerator to run a wide range of CNNs, from small to relatively large CNN.

V. The Partitioning Method

This paper proposes a hardware-based methodology that utilizes a lightweight, simple, and small CNN accelerator engine to execute a wide range of relatively large CNNs. In this way, it is like one design a big CNN accelerator inside an embedded system, but actually, there is a small lightweight accelerator capable of running big CNNs efficiently with less hardware and memory.

Using this engine, any type of CNNs can be calculated in a step-by-step and partial method that is similar to the concept of loop tiling [13, 14]. Our used methodology and the designed CNN accelerator engine can perform depthwise convolutional operations at very good performance and standard convolution but at lower performance. The designed CNN accelerator engine as a basic unit has sixteen feature maps as input and produces sixteen output feature maps. For simplicity, a convolution with sixteen inputs and sixteen outputs is displayed as 16x16. Note that these numbers do not mean image resolution or feature map pixels. This accelerator can perform convolution operations on any image size with any resolution. To minimize the hardware, this accelerator operates on 8-bit integer numbers. Also, the CNN accelerator has 16 fast internal SRAM banks as internal memories for storing the calculated feature maps. Therefore, every output feature map has its own dedicated fast memory bank, and no congestion happens during accessing memory.

The basic-unit CNN engine performs a part of a large convolution layer at each time. Finally, these parts should be added or concatenated, as discussed below. In other words, using this approach, instead of generating all feature maps in one step in a layer, feature maps for that layer are generated and placed in multiple steps. The details of the technique are explained using an example on MobileNet-v1.

V.I MobileNetV1: As an Example

MobileNetV1 is chosen as an example. This network has 16 convolutional layers, as explained in table 1. The first row of the table represents the first convolutional layers. Layer 1 includes three input parts of input image and 32 output feature maps (Conv3x32). Layer 2 includes 16 input feature maps and 32 output feature maps (Conv16x32). Layer 3 includes 32 input feature maps and 32 output feature maps (Conv32x32). The other layers are repeated as shown. The two last layers have 128 output feature maps, as shown in the figure (Conv64x128 and Conv128x128, respectively).

This example shows how a small CNN accelerator (the basic unit) runs a relatively big MobileNetV1 CNN from the first layer to the last layer, step by step. The CNN accelerator stores the results of each layer in the memory banks that are allocated for storing the results of the CNN accelerator. CNN accelerator will use these results as input for the next layer in the next step. In the last layer, all output feature maps that are stored in the memory banks are treated as the output of the CNN.

TABLE I. MobileNet-v1 Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1 Conv pw / s1	3 × 3 × 32 dw 1 × 1 × 32 × 64	112 × 112 × 32 112 × 112 × 32
Conv dw / s2 Conv pw / s1	3 × 3 × 64 dw 1 × 1 × 32 × 128	112 × 112 × 64 56 × 56 × 64
Conv dw / s1 Conv pw / s1	3 × 3 × 128 dw 1 × 1 × 128 × 128	56 × 56 × 128 56 × 56 × 128
Conv dw / s2 Conv pw / s1	3 × 3 × 128 dw 1 × 1 × 128 × 256	56 × 56 × 128 28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw 1 × 1 × 256 × 256	28 × 28 × 256 28 × 28 × 256

[DOI: 10.61186/itrc.15.2.12]

Conv pw / s1			
Conv dw / s2 Conv pw / s1		3 × 3 × 256 dw 1 × 1 × 256 × 512	28 × 28 × 256 14 × 14 × 256
5×	Conv dw / s1 Conv pw / s1	3 × 3 × 512 dw 1 × 1 × 512 × 512	14 × 14 × 512 14 × 14 × 512
Conv dw / s2 Conv pw / s1		3 × 3 × 512 dw 1 × 1 × 512 × 1024	14 × 14 × 512 7 × 7 × 512
Conv dw / s2 Conv pw / s1		3 × 3 × 1024 dw 1 × 1 × 1024 × 1024	7 × 7 × 1024 7 × 7 × 1024
Avg Pool / s1		Pool 7 × 7	7 × 7 × 1024
FC / s1		1024 × 1000	1 × 1 × 1024
Softmax / s1		Classifier	1 × 1 × 1000

An example explains this process. Assume a 16x16 CNN accelerator is available (the basic unit that we have designed already). As discussed above, the MobileNetV1 network has 32 kernels in the first layer, 64 kernels in the second layer, and up to 1024 kernels in the last layers. In the first layer, the first group of feature maps is generated from the input image. As the first CNN layer has 32 kernels in this layer, the CNN accelerator can execute the whole network in two tries or steps, as shown in Fig.2. All 32 generated feature maps from the input image will be stored in the memory banks for using in the next convolution. As there are only 16 memory banks, in the second convolution try, the generated feature maps should be stored again in the 16 memory banks but in different locations.

In the second layer of convolution, the second group of feature maps should be generated from 32 generated feature maps in the previous layer. Because the CNN has 32 input kernels and 32 output kernels in this layer, the CNN accelerator can execute the whole convolution in four tries. A 32×32 convolution is supposed as four 16x16 convolutions, as shown in Fig. 4. Therefore, it is executed as four consequent convolutions. Again the generated feature maps should be stored in the 16 memory banks but different locations (Fig. 3 and also Fig. 4, 5 for next layers).

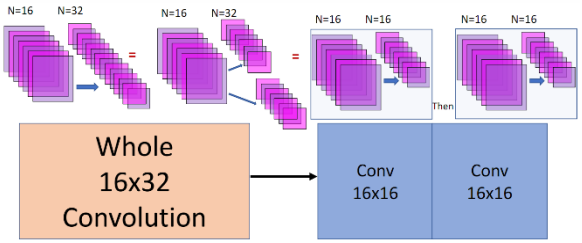

Figure 2. 16x32 convolution in two consequent steps.

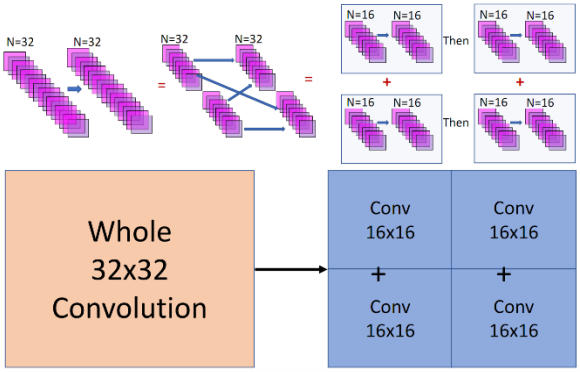

Figure 3. Executing 32x32 convolution in four consequent steps.

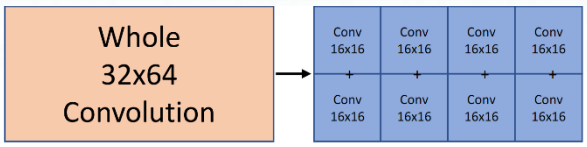

Figure 4. Executing 32x64 convolution in eight consequent steps.

In the third layer of convolution, the third group of 128 feature maps should be generated from 64 generated feature maps in the previous layer. As shown in Fig. 6, a convolution calculation can be done in 32 tries. In this case, the whole convolution is done by 32 runs of CNN accelerator (the basic unit). In addition, 24 ”add” operations should be done to add partial convolution and convert sixteen parts into four.

In the fourth layer of convolution, the group of 128 feature maps should be generated from 128 generated feature maps. As explained in Fig. 7, convolution can be calculated in 64 tries. Fifty-six ”add” operations should be done to add partial convolutions and merge eight parts into four sets of feature maps.

In the fifth layer of convolution, the group of 256 feature maps should be generated from 128 generated feature maps in the previous layer. Similar to what is shown in Fig. 7, a convolution calculation can be done in 128 tries. In this case, the whole convolution is done by sixteen runs of the CNN accelerator. In addition, 112 ”add” operations should be done to add partial convolutions.

This process continues similarly until the network reaches the last layer of CNN. The last layer of CNN is the fourteenth layer, which has 1024 feature maps and a 7x7 resolution of each feature map, as shown in Table.1.

The remaining three layers are average pooling, fully connected, and SoftMax layer. These layers are not computation intensive out of accelerator in CPU with ant major computation load. So, only the core of the CNN network is executed by the accelerator at high speed.

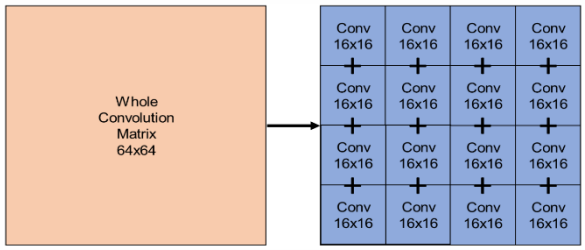

Figure 5. Executing 64x64 convolution in sixteen consequent steps.

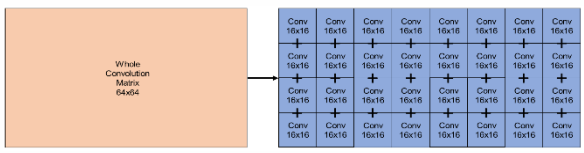

Figure 6. Executing 64x128 convolution in sixteen consequent steps.

[DOI: 10.61186/itrc.15.2.12]

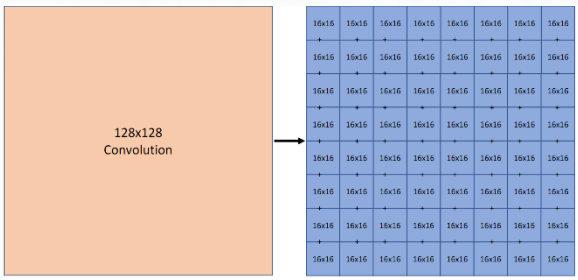

Figure 7. Executing 128x128 convolution in sixty forth consequent steps.

The layers between the fifth and eleventh layers are equal to or similar to those described in the example. For example, in layer ten, the 64 input and 128 output feature maps must be created. This issue is similar to Figure 5. In this layer, the number of convolutions is 32, and the number of additions is 24. If the number of kernels in the last layer reaches 256, then the last layer is calculated similarly to what is shown in Fig. 5, with different steps. Therefore, it is shown visually that this accelerator can partially run each layer of a big CNN network.

Usually, the size of convolutions is duplicated in some layers of CNNs. In rare cases where the size of convolutions is not multiplicand of 16, it is possible to calculate those convolutions by adding some zeroes to the end side of convolutions and fix sizes to true multiplicand of 16 as needed (similar to the concept of padding). According to what has been described earlier, using this accelerator in networks whose dimensions are more than eight times the dimensions of the accelerator will lead to lots of overheads and delays. Therefore, it is better to use 32x32 accelerators for networks with 256 kernels in the last layers in the above case. The occupancy chip area of a 32x32 accelerator that can produce 32 feature maps from 32 input feature maps is about four times more than a 16x16 accelerator. Therefore, a good trade-off should be established between the accelerator chip area and the level of required processing capability.

Our proposed method enhances the performance of the CNN network in terms of resource utilization. This issue reduces the required resources (hardware resources), which are necessary for embedded systems. Moreover, loop tiling reduces memory resources. In other words, we need less memory in the embedded system for processing the CNN network by utilizing the loop tiling.

VI. RESULTS

This paper originally describes the methodology of partial calculation of a relatively big CNN network with a small CNN accelerator. Although the implementation is not the primary goal, a small prototype CNN accelerator is designed for the proof concept. This design is implemented on the ZedBoard for initial evaluation. ZedBoard is an FPGA development board that contains a medium size FPGA from Xilinx (XC7Z020). This ZYNQ FPGA has two built-in ARM Cortex processors. One of these processors is used as the main processor. The main processor is connected to the CNN accelerator using the built-in AXI bus in ZYNQ FPGAs.

Different networks for testing the built-in accelerator and the loop tiling method are considered. These networks are MobileNetV1, MobilenetV2, fd-MobileNet, ShuffleNet, Xception, MobileNetV1-0.5 and MobileNetV1-0.25. These are all networks that are all designed for embedded and mobile processors in terms of network parameters, computing volume, and network bandwidth.

The execution strength of the built accelerator is obtained as frames per second (FPS), which is shown in Figure 8. As can be seen, for most of the selected networks, an average of seventy images are processed in one second, which is beyond the needs of most hidden applications. In the meantime, only the Xception network runs slightly slower than the others. It is necessary to explain that the Xception network is relatively heavy with reasonable accuracy from the point of view of computation. This network is also capable of processing seventeen images per second, which is suitable for most embedded applications.

As most of the previous state-of-the-art papers have used MobileNet, especially MobileNetV1, the results of MobileNetV1 are used to evaluate the proposed method.

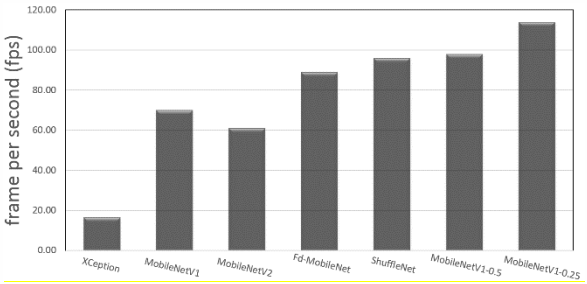

Figure 8. Amount of frame rate per second (FPS) of lightweight CNN models on the proposed accelerator and using loop tiling.

Using this medium-size FPGA, we show how compact and small our design is compared to [5] and [7]. Table.2 shows the initial result of the implementation of an accelerator designed for eight-bit integer data on ZedBoard in comparison with two recently published works [5, 7].

TABLE II. RESULT OVER XC7Z020 FPGA (ZEDBOARD)

FPGA	DSP	BRAM	Power	Clock	Work
XC7Z020	200	112/140	3.1 W	120 MHz	Current
Stratix-V GXA7	256	2330/2560	19.1 W	100 MHz	Ref [5]
XC7VX690T	1027	850 / 1470	9 W	200 MHz	Ref [7]
XC7Z045	224	162/545	5.3 W	200 MHz	Ref [16]

As can be seen, the accelerator designed for the 8-bit integer data is fully embedded in the FPGA chip and takes up almost 91.4% of its resources. The MobileNetV1 is run on this FPGA board about 70 times per second over a 224×224 resolution image at the frequency of 120MHz, which is a good execution rate for a small FPGA at low power of 3.1 Watt. The image has been selected from the ImageNet dataset and has a

[DOI: 10.61186/itrc.15.2.12]

resolution of 224×224. The MobileNetV1 is trained for remote sensing classification application with 21 classes.

VII. Conclusion

This paper uses a new method for executing large convolutions over a small CNN accelerator. We select the MobileNetV1 CNN network as a case study. Our initial implementation shows that the applied method can effectively break up and run MobileNetV1, which has 1024 convolution kernels at its last layers. This paper shows how a large convolutional network is visually broken up and run partially with a lightweight CNN accelerator. This method is useful for low-power and embedded processors that cannot tolerate a huge CNN accelerator.

References

[1] Tianyi, Liu, et al.: 'Implementation of Training Convolutional Neural Networks', arXiv:1506.01195, 2015.

[2] Andrii O. Tarasenko, et al.: 'Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future', J. Cognitive neuroscience, 2020.

[3] Asifullah Khan1, et al.: 'A Survey of the Recent Architectures of Deep Convolutional Neural Networks', Artificial Intelligence Review, DOI: https://doi.org/10.1007/s10462-020-09825-6.

[4] Min Wang, et al.: 'Factorized Convolutional Neural Networks', P. IEEE International conference on Computer Vision Workshops, P.545-553, 2017.

[5] Yufei Ma, et al. 'ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler', Integration, the VLSI Journal, 2018, ELSEVIER, pp14-23.

[6] Andrew G. Howard, et al. 'MobileNet: Efficient Convolutional Neural Networks for Mobile Vision Applications', arXiv:1704.0486,2017.

[7] Xiaocong Lian, et al. 'High-Performance FPGA-Based CNN Accelerator With Block-Floating-Point Arithmetic', IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 27, NO. 8, AUGUST 2019, pp.1874-1885.

[8] Yongming Shen, et al. 'Escher: A CNN Accelerator with Flexible Buffering to Minimize Off-Chip Transfer', Annual IEEE Symposium on Filed-Programmable Custom Computing Machine FCCM, 2017, pp.93-100.

[9] Wei Dinga, et al. 'Designing Efficient Accelerator of Depthwise Separable Convolutional Neural Network on FPGA', Journal of Systems Architecture, ELSEVIER 2019, DOI: https://doi.org/10.1016/j.sysarc.2018.12.008.

[10] Jiang Su, et al. 'Redundancy-reduced MobileNet Acceleration on Reconfigurable Logic For ImageNet Classification', International Symposium on Applied Reconfigurable Computing, 16-28, 2018.

[11] Yu-Hsin chen, et al. 'Efficient Processing of Deep Neural Networks: A Tutorial and Survey', pp. 2295-2329, Proceedings of the IEEE - Vol. 105, No. 12, December 2017.

[12] H. Kopka and P. W. Daly, A Guide to L AT EX, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[13] A. Stoutchinin, et al. `Optimally Scheduling CNN Convolutions for Efficient Memory Access', IEEE Transaction on computer-aided design of integrated circuits and systems, Feb 2019.

[14] H. Sharma, et al. `Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks', ISCA 2018.

[15] X. Zhang, et al. "SuffleNet:An Extremly Efficient Convolutional Neural Network for Moblile devices", 2017, arxiv:1707.01083v2.

[16] Y. Huang, et al. "An efficient loop tiling framework for convolutional neural network inference accelerators june", vol.16, pp.116-123, the Institue of Engineering and Thechnology, IET Circuits Devices Syst, 2022.

[17] M. Merouani, et al. "Progress Report: A Deep Learning Guided Exploration of Affine Unimodular Loop Transformations" IMPACT 2022.

[18] R. Li, et al. "Analytical Characterization and Design Space Exploration for Optimization of CNNs" ASPLOS '21, April 19-23, 2021, Virtual, USA.

[19] P. Darbani, N. Rohbani, H. Beitollahi, P. Lotfi-Kamran " RASHT: A Partially Reconfigurable Architecture for Efficient Implementation of CNNs" IEEE Transactions on very large scale integration (VLSI), Vol. 30, Nr. 7, pp. 860-868, 2022.

Ali A. D. Farahani received his B.Sc. degree in Electronic Engineering from the Iran University of Science and Technology (IUST), Tehran, Iran, in 1999 and his M.Sc. degree from K. N. Toosi University of Technology (KNTU) in 2001. He is currently working toward the Ph.D. degree at the Department of Computer Engineering, IUST. His research interests include Computer Architecture, Hardware Accelerators, and Reconfigurable Computing.

Hakem Beitollahi received his B.Sc. degree in Computer Engineering from the University of Tehran, Tehran, Iran, in 2002, the M.Sc. degree from the Sharif University of Technology, Tehran, in 2005 and the Ph.D. degree from the University of Leuven, Leuven, Belgium, in 2012. He is currently an Assistant Professor and the Head of the Computer Architecture Branch, School of Computer Engineering, Iran University of Science and Technology (IUST), Tehran. His research interests include Real-Time Systems, Reconfigurable Computing, and Hardware Accelerators.

Mahmood Fathy received his B.Sc. degree in Electronics from the Iran University of Science and Technology (IUST), Tehran, Iran, in 1985, the M.Sc. degree in Computer Architecture from Bradford University, Bradford, U.K., in 1987 and the Ph.D. degree in Image Processing Computer Architecture from The University of Manchester, Manchester, U.K., in 1991. He is currently a Full Professor with the School of Computer Engineering, IUST. His research interests include High Performance Computing (HPC), Quality of Service, Hardware Design and Real-Time Image Processing.

Reza Berangi was an Associate Professor at IUS. He is currently retired. His main research interests are Digital Signal Processing, Networking, Deep Learning, Hardware Accelerator and Cloud Computing.

[DOI: 10.61186/itrc.15.2.12]