IP Core for Efficient Zero-Run Length Compression of CNN Feature Maps

Andrea Erdeljan, Bogdan Vukobratović, and Rastislav Struharik, *Member, IEEE*

***Abstract* — Convolutional Neural Networks (CNNs) are becoming a fundamental tool for machine learning. High performance and energy efficiency are of great importance for deployments of CNNs in many embedded applications. Energy consumption during CNN processing is dominated by memory access and since large networks do not fit in on-chip storage, they require expensive DRAM access. This paper introduces an universal Output Stream Manager (OSM) which can be used to compress and format data coming from a CNN accelerator and reduce external memory access. The OSM exploits the sparsity of data and implements two Zero-Run Length encoding algorithms and can be easily reconfigured to optimize usage for different CNN layers.**

***Keywords* — Convolutional Neural Networks, FPGA, computer vision, zero run-length encoding, Zynq-7000, SystemVerilog.**

I. Introduction

In recent years, convolutional neural networks (CNNs) have become the dominant neural-network architecture for solving problems in various domains including image recognition, speech processing and natural language processing.

However, the inference phase in CNNs presents significant computational and memory challenges, especially with the use of deeper neural networks and larger inputs. The inference phase typically requires several billion multiply accumulate operations (MAC) per image. Furthermore, CNNs have a large number of network parameters and during layer processing, CNN uses the input feature map which is the output of the previous CNN layer or the input image, and produces an output feature map. Storage space for both the network parameters and the feature map data movement can become large, which makes it difficult to deploy CNNs in various embedded applications where the size of the memory is of high concern. For example, for VGG-16 CNN [1], which has more than 138 million network parameters, approximately 276 MB of storage space is needed if 16-bit number representation is used. Furthermore, even though sizes of input and output feature maps depend on the sizes of CNN layers, largest input feature and output feature maps in case of the VGG-16 are both more than 6 MB large. Compared to the size of the memory required to store VGG-16 parameters, size of the required feature map memory is only 2% of the required parameter memory size. However, since VGG-16 CNN is composed of 16 different layers, required feature map data movement to process one input image reaches 60 MB [2].

The other issue is energy consumption. Running large neural networks requires a lot of memory bandwidth to fetch the weights and a lot of computation to do dot products which consumes considerable energy. Since the majority of embedded systems are battery constrained, deploying CNNs is proving to be a big challenge in these applications. Energy consumption is dominated by memory access. Under 45nm CMOS technology, a 32 bit floating point add consumes 0.9pJ, a 32bit SRAM cache access takes 5pJ, while a 32bit DRAM memory access takes 640pJ, which is 3 orders of magnitude of an add operation, as shown in [3]. Large networks do not fit in on-chip storage and hence require the more costly DRAM accesses. For example, running the AlexNet CNN [4], one of the smaller CNNs with 800 million network connections, at 30 frames per second, would require (30Hz)(0.8G)(640pJ) = 15.36 W just for making necessary DRAM accesses.

For the above mentioned resons, multi-core CPUs and GPUs are becoming inadequate for this computation in embedded systems. Due to this, many FPGA based accelerators are beeing developed to improve the performance and energy efficiency of CNNs [2], [5]-[13].

To further improve energy efficiency, data statistics of CNN can be explored to reduce DRAM accesses using feature map compression, which is the most energy consuming data movement per access in every CNN accelerator. Since many layers use the rectified linear unit (ReLU) as the non-linear operation, all negative-valued results become zero. The resulting feature maps are often sparse. While the number of zeros in the feature maps depends on the input data to the CNN, it tends to increase with deep layers. For example in VGG-16, almost 48% of the input feature map values of CONV1_2 layer are zeros on average, and it goes up to around 88% at CONV5_3 layer. In AlexNet, almost 40% of input feature map values of CONV2 are zeros on average, and it goes up to around 75% at CONV5 [6]. This property can be used to decrease

Paper received April 24, 2018; revised June 25, 2018; accepted June 26, 2018. Date of publication July 31, 2018. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Vujo Drndarević.

This paper is a revised and expanded version of the paper presented at the 25th Telecommunications Forum TELFOR 2017 [19].

This work is partially supported by the following grant: „Innovative electronic components and systems based on inorganic and organic technologies embedded in consumer goods and products“, pr. num. TR32016, Serbian Ministry of Education and Science.

Andrea M. Erdeljan, University of Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad (email: andrea.erdeljan@uns.ac.rs).

Bogdan Z. Vukobratović, Frobas GmbH, Gebrüder-Eicher-Ring 45, Forstern, Germany, email: bogdan.vukobratovic@gmail.com).

Rastislav J.R. Struharik, University of Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad (phone +381-21-485-2537, (email: rasti@uns.ac.rs).

DRAM access by using a compressed encoding for zero values. In Zero Run-Length (ZRL) encoding consecutive runs of zeros are represented by the single run-length value, instead of original sequence of zeros. Since all the feature maps are stored in the compressed form in the DRAM, more data can fit in on-chip RAM and reduce expensive DRAM access as well as save memory space.

In this paper, we introduce an universal Output Stream Manager (OSM) which can be used to accept output feature map data from the CNN accelerator, further process it, format and stream the processed output feature map data to the external DRAM memory using standardized AXI Full protocol. The implemented IP supports two ZRL algorithms as well as various modes of operation enabling quick reconfiguration for different CNN layers. The core was implemented using SystemVerilog hardware description and verification language and it is technology independent. Data processing performance of IP core is tested using field programmable gate array (FPGA) implementation of the core.

The implemented core is universal and can be easily used with any CNN accelerator, for example [2]. In addition to the OSM, an Input Stream Manager should be used to decompress the compressed data, coming from the external DRAM, and route it to the accelerator, an example can be seen in [14].

The rest of the paper is organized as follows: Section 2 gives an overview of the implemented compression algorithms as well as their comparison and performance in different CNN layers. Section 3 contains a description of the implementation, while the utilizaton and performance results are given in Section 4. The conclusion is given in Section 5.

II. Compression algorithms

The OSM supports two variants of the ZRL compression algorithm. The two algorithms compress runs of zeros, but use different data formats to store the compressed data. Both are based on two elements: non-zero values (NZV) and the encoding map. This map can either be a non-zero index value (NZIV) or a zero-interval (ZI).

The NZV vector is of variable length and holds all the non-zero values from the input data. The NZIV or ZI vectors hold information about the positions of the non-zero values in the uncompressed data stream and are used to reconstruct the input data. The algorithms sequentially produce two vectors: NZV followed by NZIV or ZI. The compression schemes are illustrated in the following subsections.

A. Non-zero index values

The NZIV vector is based on using N_w-bit bit-vectors where each bit position indicates the presence, or absence, of a non-zero value in the uncompressed data stream. Each location set to 1 is an indication that there is a non-zero valued symbol at that location in the input data stream. In other words:

$$NZIV\ (i) = \begin{cases} 1, \text{if } input\_data[i] \neq 0 \\ 0, \text{otherwise} \end{cases} \quad (1)$$

The algorithm operation is illustrated in Fig. 1.

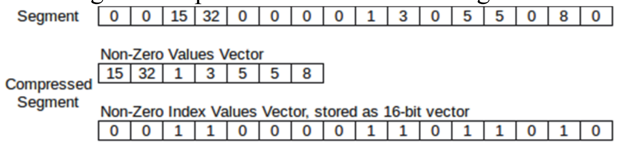

Fig. 1. Illustration of the ZRL NZIV algorithm.

B. Zero interval

The ZI vector, on the other hand, is based on encoding relative positions of non-zero values, using N_p-bit words. It has the same number of elements as the NZV vector. Each non-zero value has a corresponding zero-interval value. The ZI values are determined by the number of zero values seen before the non-zero one ie. it counts the distance between the two non-zero values.

An illustration is given in Fig. 2.

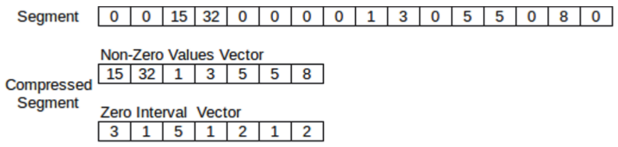

Fig. 2. Illustration of the ZRL ZI algorithm.

C. Criteria for choosing an algorithm

The two implemented algorithms offer different compressed data overheads, depending on the compression ratio of the input data stream. In case when compression ratio is small, NZIV offers a better solution, while ZI is superior in case when compression ratio is high. Cross-point between using these two algorithms depends on the actual values for N_p (size of the ZI word) and Sparsity (SP), where the Sparsity is defined as the ratio between the uncompressed data block size, N, and the number of non-zero values in uncompressed data block, N_{nz}, $SP = N/N_{nz}$. However, if the Sparsity is close to 1 the overhead will be too big and in these cases using compression gives no advantage. This also depends on the actual number of points in the uncompressed data stream, N_t. The criterion for choosing an algorithm is as follows:

```
If N/(Nnz+Nt) ≤ 1:
  Do not use compression
Else:
  If Np < SP: use NZIV
  If Np > SP: use ZI
```

For example, if we use N_p = 8 bits to encode the relative position of the next non-zero value in the input data stream, then the cross-point between using two algorithms is equal to sparsity value of 8, meaning that using NZIV gives a better end result until the compressed data stream size is larger than the 12.5% of the uncompressed data stream size.

Sparsity of feature maps in convolutional and pooling layers are usually smaller than 8 [10], and in these layers NZIV algorithm should be used. In the case of fully-connected layers, SPs are usually bigger than 8, meaning the ZI algorithm should be used to encode compressed data.

Fig. 3 shows the cross-points between the algorithms based on the percentage of zeros in the data stream assuming that the word width is 16 bits. The figure shows three functions: NZIV, ZI when Np = 8 and ZI when Np = 16. When the percentage of zeros in the data is low, the

NZIV algorithm produces a higher percentage of DRAM access reduction. The value of Np also greatly influences the results.

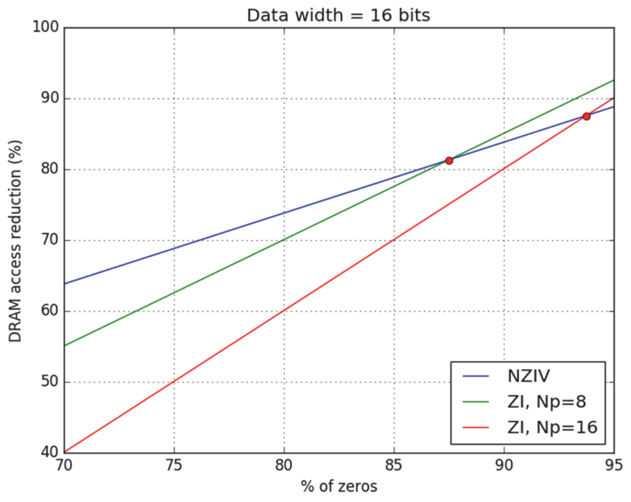

Fig. 3. Comparison of algorithms.

III. Implementation

CNN convolution layers can have a very large number of kernels, so many accelerators need to cycle through the input feature map several times each time getting a part of the output feature map. Since these parts can be non uniform in length, a lot of storage space can be wasted by always padding the location to some predefined maximum value. The padding can be ommited if we store the length of each part so the NZV values in the next cycle can be stored starting from the previous location and thus saving space. In these cases, the Number of Non-Zero Values (NNZV) is stored and the next address is generated using this information.

A block diagram of the developed architecture is shown in Fig. 4. There are four AXI4-Full interfaces - two master interfaces which are used for memory access, one for NZV and NZIV/ZI vectors and another for NNZV storage, and two slave interfaces which are used for input data and configuration. Apart from these AXI interfaces, all other interfaces are a generic Data Transfer Interface (DTI) [16]. All internal modules use this interface. DTI interface is a point-to-point interface between two modules - a master and a slave. It constists of four signals: „data“ (data bus for transferring data from master to slave, whose width can be chosen arbitrarily), „valid“ (a control signal generated by the master to indicate that it has placed some data on the data bus for the slave to read), „ready“ (a control signal generated by the slave to acknowledge that it is ready to accept potential data sent by the master), and „eot“ (end of transaction; control signal, generated by the master, to indicate that the current data transfer transaction is complete). Using this interface simlifies module connections and communications by enabling easy flow control synchronization via its handshake protocol.

The Data Compression module of the OSM is presented in Fig. 5. It can be configured to use either the NZIV or the ZI compression scheme or no compression. The configuration input controls the multiplexers and demultiplexers which control the route that is taken. Zero Compare block analyses the input data. It outputs only non-zero values and gives a one bit indication wheter the data was zero or not. That indication is then used in the NZIV and ZI code blocks where the code word is generated. FIFO is used to buffer the data while the NZV vector is being sent. The NNZV counter passes the data while counting the number of NZV points. This information is needed for NNZV and address generation. The number of non-zero words is counted for each transaction separately to ensure correct address generation. Transactions are separated by the „eot“ signal. Finally, a Round Robin block is used to send the NZV vector and NZIV/ZI vectors in turn. Optionally a width converter can be used to prepare the data to the correct format.

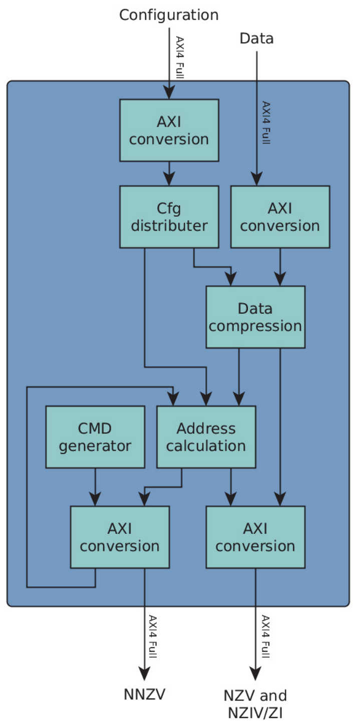

Fig. 4. OSM block diagram.

In parallel with the data, an address is also generated in the address calculation module shown in Fig. 6. The input configuration consists of: the NZV base address and the NZIV/ZI base address which represent the offset used for generating addresses, the size of the sector which is the size of the assigned memory, the number of sectors or how many addresses need to be generated and wheter or not to use NNZV. The generated NNZV sent to the output is the sum of the previous NNZV, which is read from memory, and the current NZV count, which is sent from the data compression module. The NNZV values are always read and written in order, so a simple counter can be used for generating the commands. As for the address generator block, it is used to generate the NZV and NZIV/ZI addresses in turn. This is necessary to ensure correct pairing with the data. The NZV address depends on the previous number of non-zero values, which is read from memory. The addresses are generated by the following pseudocode (NNZV is used only when the corresponding configuration bit is set):

```
repeat (section number)
  nzv address = cnt + NZV base (+NNZV)
  nziv/zi address = cnt + NZIV/ZI base
  cnt += section size
```

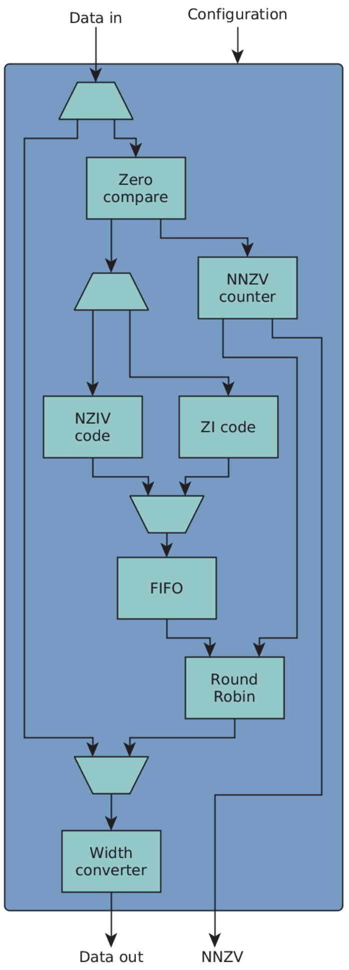

Fig. 5. Data Compression block diagram.

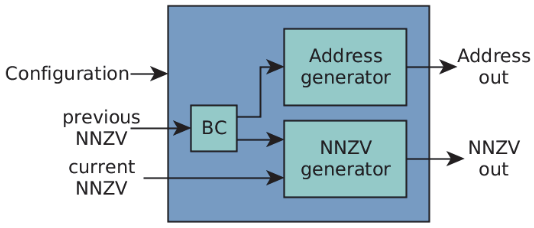

Fig. 6. Address calculation block diagram.

IV. Experiments

Implementation of developed IP core was done using Xilinx's Vivado tool with default settings, which means a balanced performance and area utilization. Target FPGA device for the implementation was Zynq-7020. The most interesting implementation results, regarding used hardware resources, are presented in Table 1. The measurements were done using a 16-bit number representation and 64-bit AXI interface.

Test system for implemented IP core is shown in Fig. 7. System is based on ARM Cortex-A9 processor, present in the Zynq-7020 FPGA. Used data storage was DDR3 memory module on Zedboard. IP core was successfully tested with the system clock set to 83.3 MHz.

Table 1: Required hardware resources.

Resource	Utilization	Utilization (%)
LUT	5532	10.4%
LUTRAM	485	2.79%
Slice regs.	6301	5.92%
BRAM	6	4.29%

Table 2: Breakdown of the five CONV layers in AlexNet.

Layer	% of zeros	Compression Type	DRAM Accesses Reduction (%)
conv1	0.01%	None	0
conv2	38.7%	NZIV	32.45
conv3	72.5%	NZIV	66.25
conv4	79.3%	NZIV	73.05
conv5	77.6%	NZIV	71.35

Table 3: Breakdown of the 13 CONV layers in VGG-16.

Layer	% of zeros	Compression Type	DRAM Accesses Reduction (%)
conv1-1	1.6%	None	0
conv1-2	47.7%	NZIV	41.45
conv2-1	24.8%	NZIV	18.55
conv2-2	38.7%	NZIV	32.45
conv3-1	39.7%	NZIV	33.45
conv3-2	58.1%	NZIV	51.85
conv3-3	58.7%	NZIV	52.45
conv4-1	64.3%	NZIV	58.05
conv4-2	74.7%	NZIV	68.45
conv4-3	85.4%	NZIV	79.15
conv5-1	79.4%	NZIV	73.15
conv5-2	87.4%	NZIV	81.15
conv5-3	88.5%	ZI	82.75

OSM performance is analysed on five publicly available and well-known CNNs: AlexNet [4] and VGG-16, VGG-19 [1], GoogLeNet [17] and MobileNet [18]. The Compression Ratio is defined as the ratio between the uncompressed size and compressed size. The compressed size includes both the NZV size and the NZIV/ZI overhead. Based on the analysis shown in Section 2, we found the optimal compression algorithm for different CNN layers and calculated the DRAM access reduction using N_p=8. The results for AlexNet are shown in Table 2, while Table 3 shows the results for VGG-16. Table 4 presents the results for MobileNet, Table 5 for VGG-19 and Table 6 for GoogleLeNet. The DRAM access reduction can go up to 85% in some layers.

V. Conclusion

In this paper we presented a hardware implementation of the Output Stream Manager which can be added to any CNN accelerator core in order to compress the output data and minimize memory access.

ZRL compression compresses only the runs of zeros within the feature maps. Future improvements of the core could further reduce the size of feature maps, by using a more elaborate compression algorithms, like the Huffman encoding.

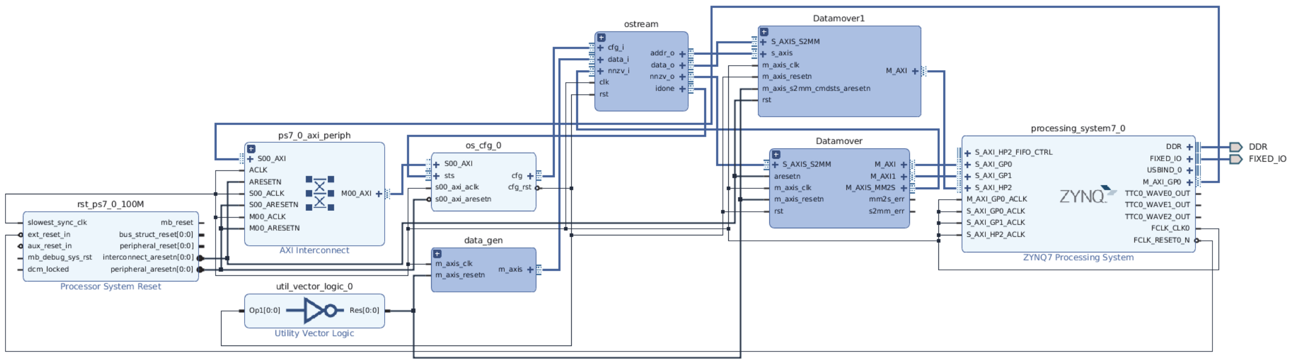

Fig. 7. Test System.

TABLE 4: BREAKDOWN OF THE CONV AND POOL LAYERS IN MOBILENET.

Layer	% of zeros	Compression Type	DRAM Accesses Reduction (%)
conv1	58.21%	NZIV	51.96
conv_dw_1	39.58%	NZIV	33.33
conv_pw_1	35.32%	NZIV	29.07
conv_dw_2	23.18%	NZIV	16.93
conv_pw_2	30.88%	NZIV	24.63
conv_dw_3	36.61%	NZIV	30.36
conv_pw_3	44.77%	NZIV	38.52
conv_dw_4	17.91%	NZIV	11.66
conv_pw_4	18.45%	NZIV	12.20
conv_dw_5	44.84%	NZIV	38.59
conv_pw_5	47.18%	NZIV	40.93
conv_dw_6	25.98%	NZIV	19.73
conv_pw_6	24.34%	NZIV	18.09
conv_dw_7	50.48%	NZIV	44.23
conv_pw_7	35.14%	NZIV	28.89
conv_dw_8	47.80%	NZIV	41.55
conv_pw_8	42.38%	NZIV	36.13
conv_dw_9	44.97%	NZIV	38.72
conv_pw_9	41.70%	NZIV	35.45
conv_dw_10	42.28%	NZIV	36.03
conv_pw_10	51.57%	NZIV	45.32
conv_dw_11	44.16%	NZIV	37.91
conv_pw_11	60.59%	NZIV	54.34
conv_dw_12	15.91%	NZIV	9.66
conv_pw_12	61.91%	NZIV	55.66
conv_dw_13	49.65%	NZIV	43.40
conv_pw_13	83.64%	NZIV	77.39
Global average pool	27.34%	NZIV	21.09

TABLE 5: BREAKDOWN OF THE CONV AND POOL LAYERS IN VGG-19.

Layer	% of zeros	Compression Type	DRAM Accesses Reduction (%)
conv1-1	47%	NZIV	41
pool1	20%	NZIV	14
conv2-1	38%	NZIV	32
pool2	36%	NZIV	30
conv3-1	54%	NZIV	48
conv3-2	43%	NZIV	37
conv3-3	25%	NZIV	19
pool3	25%	NZIV	19
conv4-1	47%	NZIV	41
conv4-2	62%	NZIV	56
conv4-3	69%	NZIV	63
pool4	74%	NZIV	68
conv5-1	74%	NZIV	68
conv5-2	79%	NZIV	73
conv5-3	81%	NZIV	75
pool5	88%	ZI	83

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," CoRR, vol. abs/1409.1556, 2014.

[2] R. Struharik and B. Vukobratović, "Aiscale - a coarse grained reconfigurable cnn hardware accelerator," in 2017 IEEE East-West Design Test Symposium (EWDTS-2017), Oct 2017.

[3] M. Horowitz, Energy table for 45nm process. https://sites.google.com/site/seecproject.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Commun. ACM, vol. 60, pp. 84–90, 2012.

[5] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," SIGARCH Comput. Archit. News, vol. 45, pp. 27–40, June 2017.

[6] Y. Shen, M. Ferdman, and P. A. Milder, "Escher: A cnn accelerator with flexible buffering to minimize off-chip transfer," 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 93–100, 2017.

[7] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energyefficient reconfigurable accelerator for deep convolutional neural networks," IEEE Journal of Solid-State Circuits, vol. 52, pp. 127–138, Jan 2017.

[8] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. B. K. Vrudhula, J. sun Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in FPGA, 2016.

[9] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, “Going deeper with embedded fpga platform for convolutional neural network,” in FPGA, 2016.

[10] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, and Y. Xu, “Throughputoptimized fpga accelerator for deep convolutional neural networks,” TRETS, vol. 10, pp. 17:1–17:23, 2017.

[11] Peemen, Maurice, Arnaud A. A. Setio, Bart Mesman and Henk Corporaal. “Memory-centric accelerator design for Convolutional Neural Networks.” 2013 IEEE 31st International Conference on Computer Design (ICCD) (2013): 13-19.

[12] Zhang, Chen, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao and Jason Cong. “Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks.” FPGA (2015).

[13] Reagen, Brandon, Paul N. Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei and David M. Brooks. “Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators.” 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA) (2016): 267-278.

[14] Rakanovic, Damjan, Andrea Erdeljan, Vuk Vranjkovic, Bogdan Vukobratovic, Predrag Teodorovic and Rastislav J. R. Struharik. “Reducing off-chip memory traffic in deep CNNs using stick buffer cache.” 2017 25th Telecommunication Forum (TELFOR) (2017): 1-4.

[15] Han, Song, Huizi Mao, and William J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding", arXiv preprint arXiv:1510.00149, 2015

[16] R. Struharik and B. Vukobratović, “Data Transfer Interface Specification“, Internal Report, FTN 2017.

[17] Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich. “Going deeper with convolutions.” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 1-9.

[18] Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto and Hartwig Adam. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.” CoRR abs/1704.04861 (2017): n. pag.

[19] A. Erdeljan, B. Vukobratović and R. Struharik, "IP core for efficient zero-run length compression of CNN feature maps," 2017 25th Telecommunication Forum (TELFOR), Belgrade, 2017, pp. 1-4.

TABLE 6: BREAKDOWN OF THE POOL, CONV AND INCEPTION LAYERS IN GOOGLENET.

Layer	**% of zeros**	**Compression Type**	**DRAM Accesses Reduction (%)**
pool1	17%	NZIV	11%
conv2	53%	NZIV	47%
pool2	47%	NZIV	41%
inception3-A/1x1	51%	NZIV	45%
inception3-A/3x3	50%	NZIV	44%
pool3	67%	NZIV	61%
inception4-A/1x1	81%	NZIV	75%
inception4-A/3x3	90%	ZI	85%
inception4-A/pool	33%	NZIV	28%
inception4-B/1x1	50%	NZIV	44%
inception4-B/3x3	66%	NZIV	60%
inception4-B/pool	60%	NZIV	54%
inception4-C/1x1	57%	NZIV	51%
inception4-C/3x3	73%	NZIV	67%
inception4-C/pool	29%	NZIV	23%
inception4-D/1x1	80%	NZIV	74%
inception4-D/3x3	85%	NZIV	79%
inception4-D/pool	41%	NZIV	35%
inception4-E/1x1	87%	NZIV	81%
inception4-E/3x3	87%	NZIV	81%
inception4-E/pool	64%	NZIV	58%
inception5-A/1x1	71%	NZIV	65%
inception5-A/3x3	80%	NZIV	74%
inception5-A/pool	41%	NZIV	35%
inception5-B/1x1	85%	NZIV	79%
inception5-B/3x3	83%	NZIV	77%
inception5-B/pool	56%	NZIV	50%