Hybrid Dot-Product Calculation for Convolutional Neural Networks in FPGA

Mário Véstias INESC-ID, ISEL, Instituto Politécnico de Lisboa
mvestias@deetc.isel.pt
Rui Policarpo Duarte, José T. de Sousa, Horácio Neto
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
rui.duarte@tecnico.ulisboa.pt, jose.desousa@inesc-id.pt, hcn@inesc-id.pt

***Abstract*—Convolutional Neural Networks (CNN) are quite useful in edge devices for security, surveillance, and many others. Running CNNs in embedded devices is a design challenge since these models require high computing power and large memory storage. Data quantization is an optimization technique applied to CNN to reduce the computing and memory requirements. The method reduces the number of bits used to represent weights and activations, which consequently reduces the size of operands and of the memory. The method is more effective if hybrid quantization is considered in which data in different layers may have different bit widths. This article proposes a new hardware module to calculate dot-products of CNNs with hybrid quantization. The module improves the implementation of CNNs in low density FPGAs, where the same module runs dot-products of different layers with different data quantizations. We show implementation results in ZYNQ7020 and compare with state-of-the-art works. Improvements in area and performance are achieved with the new proposed module.**

***Index Terms*—Deep learning, Convolutional Neural Network, Embedded computing, Hybrid quantization, Field-Programmable Gate Array.**

I. Introduction

Many machine learning applications, like image classification [1], car driving assistance [2], robotics [3] and game playing [4] are now based on deep learning. Deep neural networks is a deep learning method based on artificial neural networks with multiple layers. Among the several types of deep neural networks, our focus is on convolutional neural networks (CNN) which are mainly used for image analysis.

Fully connected feed forward networks can be applied to image classification. However, the high number of neurons associated with an image make this solution impractical. The CNN replaces the fully connected layers of the first layers by convolutional layers, which reduces the number of weights and permits increasing the depth of the network. Each convolutional layer receives a stack of 2D input feature maps (IFM) an runs a convolution with a 3D filter to generate an output feature map (OFM). Many filters can be applied to the same set of IFMs generating as much OFMs.

Many deep-learning applications are migrating from the cloud to the edge where the computing platform is limited in performance, memory and energy. For future deployment of CNN in edge devices, it is important to improve and optimize CNN models to reduce computing, memory and energy requirements.

In this paper, we propose an hardware arithmetic module for hybrid dot-product calculation for CNNs in FPGA, with an emphasis on low density FPGAs for edge computing. This module improves the area and performance of dot products with support for dynamic resize of weights. These are used to implement a configurable architecture for layer execution of CNNs.

The paper is organized as follows. Section II provides an overview of CNNs and their sources of parallelism. In sections III we describe the state of art on FPGA implementations of CNNs. Section IV describes the baseline architecture and section V describes the proposed hybrid architecture. Section VI shows the results and how they compare to previous works. Section VII concludes the paper.

II. Related Work

FPGAs are increasingly being used for CNN inference since they can be reconfigured to adapt to each CNN model offering good performance and energy efficiency.

Some of the first implementations of large CNN models [5], [6] consider a general hardware core for convolution that can execute different convolutional layers with different shapes. While flexible, the performance efficiency of the solution depends on the size of the convolution window. In [6] the authors tried to overcome this inefficiency implementing convolutions as matrix multiplications by rearranging the IFMs. The solution is more general, but the overhead of memory accesses and execution times associated with the rearrangement of the IFMs is large. [7] also adopt an accelerator for matrix multiplication. The overhead of the conversion of the IFM to a matrix was eliminated at the cost of using dedicated hardware units to do the conversion.

A few authors considered low density FPGAs as the target device. In [9] small CNNs are implemented in a ZYNQ XC7Z020 with a performance of 13 GOPs with 16 bit fixed-point data. In [10] the same FPGA is used to implemented big CNN models, like VGG16, with data represented with 8 bits achieving performances of 84 GOPs. In [11] a configurable architecture is proposed to implement large CNN in low density FPGAs, achieving a peak performance close to 400 GOPs in a ZYNQ XC7Z020 with data represented with 8-bits.

In this work we consider the architectural approach proposed in [11] that targets low density FPGAs. The architecture

1946-1488/19/$31.00 ©2019 IEEE
DOI 10.1109/FPL.2019.00062

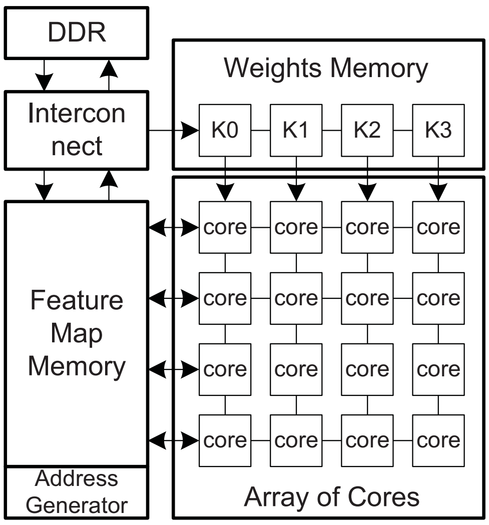

Fig. 1. Block diagram of the baseline architecture

uses a general configurable hardware module that executes one layer at a time with 8-bit fixed-point representation of activations and weights. In this paper, we modify this architecture to support hybrid quantization with a new configurable module for hybrid dot-product calculation. The architecture extended with the new module for hybrid dot-product calculation has around $2.5\times$ better performance than the baseline architecture for about the same occupation of FPGA resources.

III. Baseline Architecture for CNN

The baseline architecture consists of an array of processing cores to calculate dot-products and a memory to store OFMs (see Figure 1).

The execution of both convolutional and fully connected layers works the same way because we transform the 3D convolutions in linear dot-products identical to those used in FC layers, to be explained above.

The architecture executes one layer at a time and is configured with the features of each layer (number of kernels, size of kernels, size of IFM and size of OFM). The image or the IFMs of a layer are stored in the feature map memory (FMM) and the kernels are stored in the kernels memory with each kernel stored in a different memory. Initially, the image is stored into the FMM and kernels are stored in the kernels memory. Then, multiple activations (multiple ports are used to read multiple activations at the same time) are broadcasted to the cores that calculate dot-products between a vector of activations and a vector of weights. The result is sent back to the FMM. Before being stored in this memory, the activation function (ReLU) is applied to the result of the convolution to produce an activation of the OFM. For layers followed by pooling, the activation is stored in a local memory to wait for the other members of the pooling window. The process repeats until finishing the convolution between the image and the kernels. After that, the next kernels are loaded from memory and the process repeats until running all kernels of a layer.

IV. Hybrid Architecture for CNN

The baseline architecture considers a fixed 8-bit width for both activations and weights. This work modifies the architecture to support the execution of layers with two different weight sizes (8 and 2). The size of the activations is still 8-bits and constant for all layers since the accuracy of a CNN is more sensible to the size of activations.

Each core of the original architecture receives eight activations and eight weights of a single kernel in parallel (a total of 64 bits each). The core of the hybrid architecture receives also 64 bits of activations and 64 bits of weights, that is, the dot-product parallelism is still eight since there are only eight different activations available in each cycle. However, in a layer whose weights are represented with less bits, 64 bits of weights can represent more than eight weights. Using the 64 bits to represent more than eight weights of the same kernel is useless since there are only eight activations available. The solution is to use them to represent weights of different kernels. For example, with 2 bit-weights, we can represent a total of eight weights of four different kernels in a total of 64 bits.

The hybrid core proposed in this work is able to calculate dot-products with different data sizes and multiple dot-products. For example, considering a CNN with some layers having 8-bit activations and weights, and other layers having 8-bit activations and 2-bit weights. The hybrid core supports the parallel execution of one dot-product between eight 8-bit activations and eight 8-bit weights and four dot-products between eight 8-bit activations and eight 2-bit weights.

The hybrid core proposed in this work supports the execution of two different $activation \times weight$ representations: 8×8 and 8×2 (represented as 8:82). The solution for the hybrid calculation consists on implementing 8×2 dot-products, which are added as partial products of a multiplication case the layer requires 8×8 dot-products.

Let's consider eight 8-bit activations, $A_0, A_1, ..., A_7$, that are read in parallel from the activations memory in each clock cycle. Weights are read from the kernel memory. For 2-bit weights, for groups of eight 2-bit weights are read in parallel, $W_{00}, W_{01}, ..., W_{07}$, $W_{10}, W_{11}, ..., W_{17}$, $W_{20}, W_{21}, ..., W_{27}$ and $W_{30}, W_{31}, ..., W_{37}$. In this case, four dot products, DP_0, DP_1, DP_2 and DP_3 are calculated as:

$$DPj = \sum_{i=0}^{i=7} A_i \times W_{ji} \quad (1)$$

with four multiply-accumulate units. Each dot-product is implemented with cascaded MACC units (see Figure 2).

For 8-bit weights the same unit is used to determine the dot-product, $A \cdot W$, between the eight activations and eight 8-bit weights, $W_0, W_1, ..., W_7$. To achieve this, we set $W_k = W_{3k}W_{2k}W_{1k}W_{0k}$, where $k = 0, 1, 2..., 7$ and then add the four dot products described above as follows:

$$A \cdot W = DP3 \times 2^6 + DP2 \times 2^4 + DP1 \times 2^2 + DP0 \quad (2)$$

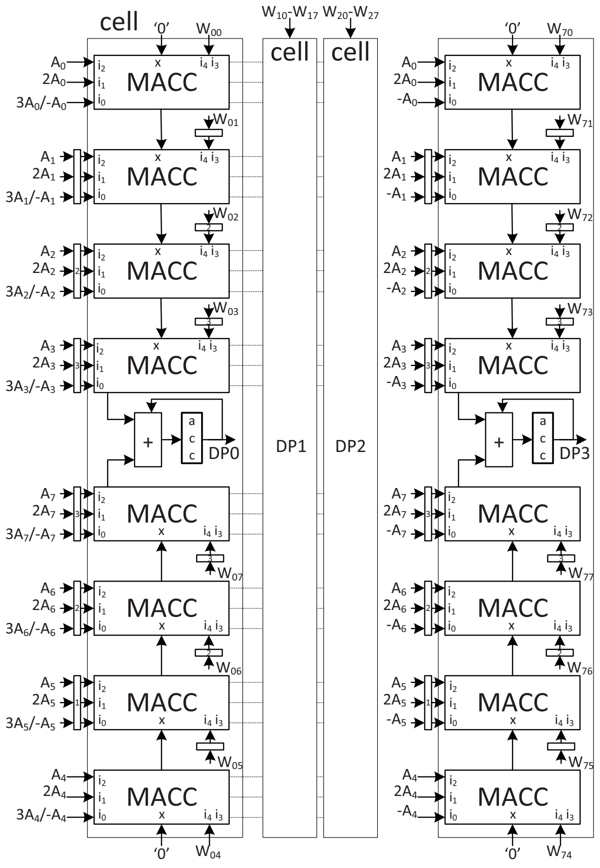

Fig. 2. Architecture of the core for 8- and 2-bit weights

Equation 2 is executed only after calculating the complete convolution. Therefore, it is implemented outside the core before each input port of the feature map memory followed by the ReLU activation function.

Each MACC unit used in the core implements the multiplication between one activation and one 2-bit weight and accumulates with the output of the previous MACC. The 2-bit weights have different meanings depending on the weight size of the layer. If the weight size is two bits then it represents numbers $\in -1, 0, 1$. Otherwise, if the weight size is 8 bits, the 2-bit weights are partials of the 8-bit weight and represent numbers 0, 1, 2, 3, except for the most significant 2-bit which represent numbers -1, 0, 1. So, given the 2-bit weights, the MACC unit selectively adds the previous MACC output with a multiple of the activation, that is, 0, A, 2A, 3A or -A. Multiples 3A and -A are calculated outside the MACC and selectively sent to the MAC according to the layer configuration. The implementation of the MACC is illustrated in figure 3.

With a chain of LUTs and the carry chain we can implement the MACC unit. Each output bit, O_{i+1}, is the sum of an input bit, x_i and another bit defined by a function of five variables i_4, i_3, i_2, i_1, i_0. This is done with a single level of LUTs [15] with f given by:

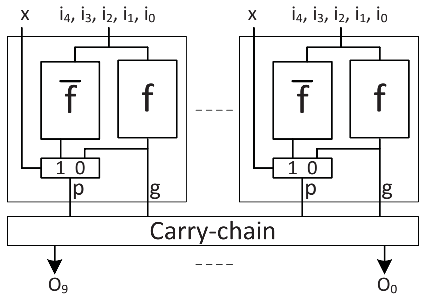

Fig. 3. Architecture of the MACC unit

$$f_i = \overline{i_4}i_3i_0 + i_4\overline{i_3}i_1 + i_4i_3i_2$$

where inputs i_n are according to the inputs illustrated in figure 2.

V. Results

The extended architecture with the proposed hybrid core was implemented with Vivado 2017.3 in the ZedBoard with a ZYNQ XC7Z020 with a working frequency of 200 MHz, except the HP interfaces to external memory that work at a frequency of 150 MHz.

We have mapped a trained AlexNet with 8-bit activations and two different weight configurations. One configuration uses 8-bit weights for all layers (54,4 % top-1 accuracy). The other configuration uses 2-bit weights except for the first and last layers (51,0 % top-1 accuracy). The overall resource utilization is shown in table I.

TABLE I
Area occupation for the baseline and hybrid architectures

Size	#cores	batch	MACCs/core	LUT	DSP	BRAM
Arq. 8:88	128	4	8	44281	220	132
Arq. 8:82	96	6	32	43052	192	124

We have improved the baseline architecture by increasing the batch size. The number of cores decreases with the weight size reduction since the hybrid cores occupy more resources. However, the number of MACCs per core is higher with the hybrid cores.

We have mapped AlexNet in the proposed architectures with hybrid cores and compared the performance of both architectures (see table II).

TABLE II
Area occupation of the baseline and hybrid architectures

Size	Conv (ms)	FC (ms)	images/s	GOPs
Arq. 8:88	3.61	3.59	139	201
Arq. 8:82	1.82	1.02	352	510

The architecture with hybrid 2-bit weights (8:82) improves the image throughput of the baseline architecture by $2.5\times$. We

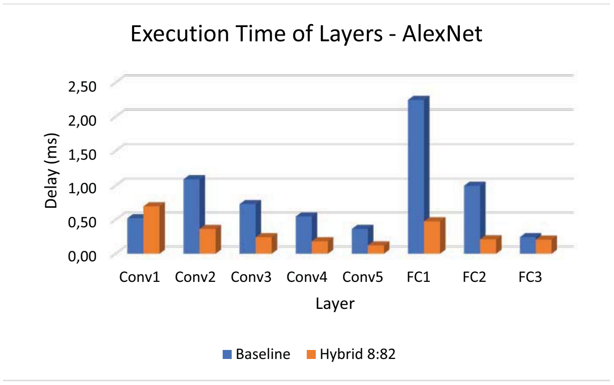

Fig. 4. Execution time of layers in each architecture

have determined the average processing times of each layer of the architectures (see figure 4)

The first layer uses always 8-bit weights in all architectures. Therefore, the baseline architecture runs this layer faster since it has more cores. The other convolutional layers decrease with the decrease of weight size, as expected. In the fully connected layers we observe the big difference between the baseline and the hybrid architectures caused by the transfer times of weights from external memory to the weight memories. Only the last layer has similar execution times. the small difference is due to the small batch size of the baseline architecture.

VI. Conclusions

The hybrid architecture proposed in this work support the execution of layers with different weight sizes. With 25% more resources per core, the performance of the architecture running AlexNet increases by about $2.5\times$.

The architecture was designed to run CNN models in low cost FPGAs. However, its scalability permits to increase the number of cores so that the performance can also be improved.

We are now planing to consider two separate modules for layer execution, so that, for example, convolutional layers can be optimized independently of fully connected layers..

Acknowledgment

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019 and was also supported by project IPL/IDI&CA/2018/LiteCNN/ISEL through Instituto Politécnico de Lisboa.

References

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015.

[2] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ser. ICCV '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 2722–2730.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, Jan. 2016.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[5] 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 *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 26–35.

[6] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 16–25.

[7] Y. Qiao, J. Shen, T. Xiao, Q. Yang, M. Wen, and C. Zhang, "Fpga-accelerated deep convolutional neural networks for high throughput and energy efficiency," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 20, pp. e3850–n/a, 2017, e3850 cpe.3850.

[8] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, and Y. Xu, "Throughput-optimized fpga accelerator for deep convolutional neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 10, no. 3, pp. 17:1–17:23, Jul. 2017.

[9] S. I. Venieris and C. Bouganis, "fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–17, 2018.

[10] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, Jan 2018.

[11] M. Véstias, R. P. Duarte, J. T. d. Sousa, and H. Neto, "Lite-cnn: A high-performance architecture to execute cnns in low density fpgas," in *Proceedings of the 28th International Conference on Field Programmable Logic and Applications*, 2018.

[12] Y. Ma, N. Suda, Y. Cao, J. s. Seo, and S. Vrudhula, "Scalable and modularized rtl compilation of convolutional neural networks onto fpga," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–8.

[13] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," in *Proceedings of the 4th International Conference on Learning Representations*, 2016.

[14] J. Wang, Q. Lou, X. Zhang, C. Zhu, Y. Lin, and D. Chen., "Design flow of accelerating hybrid extremely low bit-width neural network in embedded fpga." in *28th International Conference on Field-Programmable Logic and Applications*, 2018.

[15] E. G. Walters, "Array multipliers for high throughput in xilinx fpgas with 6-input luts," *Computers*, vol. 5, no. 4, 2016. [Online]. Available: http://www.mdpi.com/2073-431X/5/4/20

[16] S. I. Venieris and C. S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2016, pp. 40–47.

[17] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, Jan 2018.

[18] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. ACM, 2017, pp. 15–24. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021741