Acceleration of Image Classification with Caffe framework using FPGA

Dimitrios Danopoulos
Department of Electrical
and Computer Engineering
NTUA, Athens, Greece

Christoforos Kachris
Institute of Communication and
Computer Systems (ICCS/NTUA)
Athens, Greece

Dimitrios Soudris
Department of Electrical
and Computer Engineering
NTUA, Athens, Greece

***Abstract*—Caffe is a deep learning framework, originally developed at UC Berkeley and widely used in large-scale industrial applications such as vision, speech, and multimedia. It supports many different types of deep learning architectures such as CNNs (convolutional neural networks) geared towards image classification and image recognition. In this paper we develop a platform for the efficient deployment and acceleration of Caffe framework on embedded systems that are based on the Zynq SoC. The most computational intensive part of image classification is the processing of the convolution layers of the deep learning algorithms and more specifically the GEMM (general matrix multiplication) function calls. In the proposed framework, a hardware accelerator has been implemented, validated and optimized using Xilinx SDSoC Development Environment to perform the GEMM function. The accelerator that was developed achieves up to 98× speed-up compared with the simple ARM CPU implementation. The results showed that the mapping of Caffe on the FPGA-based Zynq takes advantage of the low-power, customizable and programmable fabric and ultimately reduces time and power consumption of image classification.**

I. Introduction

The continuing exponential increase of media, IoTs and big data in general requires faster and faster processing speeds while the applications must maintain a low power cost and keep the development time small. Many high performance systems rely on machine learning (ML) algorithms such as image classification, data analytics etc. which are required for embedded and big data applications as well. In this field, Deep Convolution Neural Networks (DNNs) have gained significant traction due to the fact that they offer state-of-the-art accuracies and important flexibility (e.g. by updating the classifier). Although a serious amount of computation is needed to analyze the large amounts of data, the use of multicore systems [1] seems promising but the challenge of reducing the high energy cost and the processing times remains. FPGA implementations on the other hand have seen great advancement as new emerging techniques exploit the FPGA SoCs, such as the Zynq 7000 [2], taking advantage of the high performance hardware accelerators with few power costs while keeping the adaptability of fast prototyping. With the utilization of hardware accelerators the total throughput is increased due to highly parallelizable massive number of multiply-accumulate operations (MACs) that DNN algorithms need and also the energy consumption is decreased. Caffe deep learning framework of UC Berkley [3] has already been officialy implemented and optimized in two different architectures for CPU and GPU only but can be easily configured without hard-coding. This paper presents:

- A modified version of Caffe to effortlessly port it into the ARM (Zynq 7000 based) processor of FPGA.
- A hardware accelerator designed on Xilinx SDSoC Environment that leverages the FPGA architecture to perform the image classification algorithm.
- A CPU-FPGA-based system, a highly heterogeneous all-programmable SoC that supports the Caffe framework and utilizes the hardware accelerator achieving significant speed and power efficiency compared to the ARM Zynq processor.

II. Background

A. Deep Neural Networks

A deep neural network (DNN) is a biologically-inspired programming paradigm which can model complex non-linear relationships like analyzing visual imagery. It behaves similarly to the human brain so as to simulate its densely interconnected brain cells using digital neurons that trigger when they sense certain features. A DNN can achieve the classification without being specifically programmed but with the process of training which includes learning from previous examples and modifying a set of weights from a dataset.

After training, the task is performed on new data through a process called inference which is basically a dot product of the features and weights (ex. Classifying an image). The importance of such algorithms is the accurate results, the speed of computation and the little human effort required because there is no need to hand-craft the features of the model as this is done automatically by the algorithm with great precision. Some of the fundamental layers found in many models are:

- Convolutional Layer: applies a convolution operation to the input by convolving each filter, passing the result to the next layer in order to recognize specific features.
- Pooling Layer: combines the outputs of neuron clusters at one layer into a single neuron in the next layer, thus reducing the sensitivity of the filters. This process can be

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 687628, VINEYARD http://vineyard-h2020.eu

achieved by max pooling which uses the maximum value from each cluster of neurons at the prior layer. Another example is average pooling, which uses the average value from each cluster of neurons combining their results.

- Fully Connected Layer: connects every neuron in one layer to every neuron in another layer. They can be visualized as one dimensional and perform the high level reasoning in the neural network.

B. Caffe Framework

Caffe is a deep learning framework that uses neural networks with multiple hidden layers between the input and output layer and can solve many problems in image recognition, speech recognition and natural language processing. The core of the neural network computation involves going through each layer in the network which can include many different layers and performs the so called forward and backward passes. The forward pass computes the output of each layer given the input for the inference and eventually makes a prediction. The backward pass computes the gradient given the loss for learning. In backward, Caffe reverse-composes the gradient of each layer to compute the gradient of the whole model by automatic differentiation. This is back-propagation and is needed for the training of a model where the network weights are updated. Caffe stores, communicates and manipulates all this information as *blobs*: *blob* is the standard array and unified memory interface for the framework. The details of blob describe how information is stored and communicated in and across layers and nets. The conventional blob dimensions for batches of image data are number $N \times channel\ K \times height\ H \times width\ W$ which is 4D dimensional and vary according to the type and configuration of the layer. In order to perform an image classification through Caffe framework for example, we need to specify the model definition in *prototxt* format, the pre-trained model with the weights in *caffemodel* format, the image mean of the data in *binary proto* format, the classification categories in a text file and the input image to classify. The implementation of image classification was performed using the C++ API of Caffe but a python and matlab API is provided for building as well.

III. Mapping Caffe to ARM

This section describes the steps to port the whole Caffe framework into the Zynq 7000 SoC to run on the ARM core. In order to make the Caffe run on Zynq we had to cross compile the whole framework using the ARM cross compiler that was included in the SDSoC Environment. SDSoC is the IDE-based framework provided by Xilinx that allows the development of HW/SW embedded systems for the Zynq platform. This platform offered the ability to define, integrate and verify the hardware accelerator that would accelerate a specific function that will be described later and generate both the ARM software and FPGA bitstream.

The first step was to edit the *Makefile* of the Caffe framework and change the previous default g++ toolchain where it appeared to the new ARM cross compiler toolchain. Moreover, the default library search paths of Caffe were modified so as to stop loading the x64 libraries so they were set to point on a new custom folder where all the arm libraries that Caffe demanded were placed. The cross compilation of the libraries was manual, meaning that the source files of each library that Caffe required had to be downloaded and then cross compiled one by one using the ARM toolchain. For every library, the appropriate configuration had to be defined (overcoming several complications in some library installations) before the header files and dynamic libraries were produced. These dependencies include libraries such as *opencv, boost, protobuf, hdf5, openblas, glog, leveldb, snappy, gflags, lmdb* etc. Also, the rpath-link was manually set on the Caffe Makefile to point in the custom library folder to make it the same path as the general libraries. This is used because some shared objects that were built, had encoded a wrong path to some executable arguments and the correct shared objects needed at runtime had to be located. Lastly, in the configuration of Makefile, a CPU-only mode was set, the default BLAS folder was changed to point on the custom made folder and OpenBLAS was used which is generally supported on embedded systems.

The image classification was performed with the C++ interface using the SqueezeNet network on ImageNet data because it's light and can avoid memory issues with the Zynq SoC. SqueezeNet is an 18-layer network that has 1.24 million parameters which is only 5 MB of weights in all [4]. The inference was achieved in 1.1 secs on average and produced correct results. Comparing it with the inference on our host Intel i3-3250 CPU which runs in 0.2 secs, we noticed the expected computation time which is about $5\times$ times slower on Zynq as the ARM CPU clock is almost $5\times$ times slower (667 MHz vs $3,5$ GHz). Finally, we performed a full profiling on Caffe framework using the included benchmarking tools. The results showed that the large computation overhead lies on the convolution layers of the model and several studies have explored the acceleration of this layer using Winograd's convolution for example [5]. The following chart illustrates the percentage of execution time in every layer of Caffe's reference model and clearly shows the large computation overhead in the convolution layers of the network which takes approximately 75% of the total time.

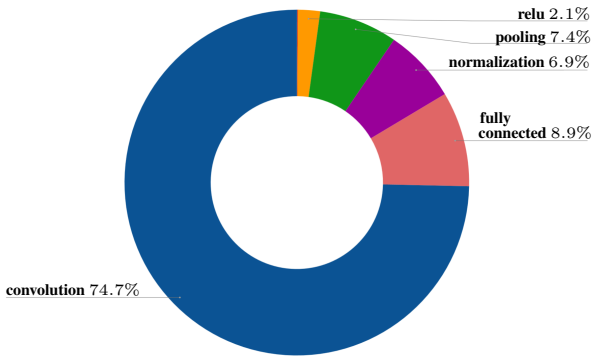

Fig. 1. Execution time per layer with Caffe profiling

IV. The hardware accelerator

After examining the C++ API and especially the convolution layer source files, we concluded that Caffe's strategy for convolution is to reduce the problem to matrix-matrix multiplication and more precisely to the GEMM function. This function which is called in every convolution layer and fully connected layer, is basically MAC (multiply-accumulate) float operations and it is responsible for most computations in a convolutional neural network.

A. Caffe's convolution with GEMM

Caffe does convolution by turning the input from an image, which is effectively a 3D array, into a 2D array which combines all the corresponding patches of the input and treats it like a matrix I_m. Then each kernel is expanded consecutively forming the filter matrix F_m.

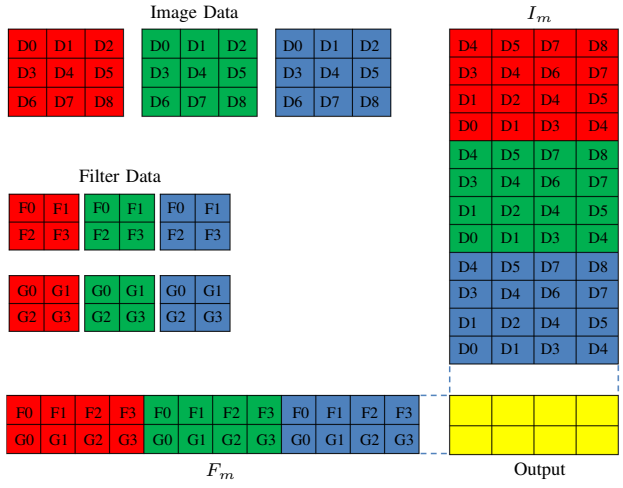

Fig. 2. Convolution lowering to matrix multiplication

As shown in the figure above, many pixels that are included in overlapping kernels will be duplicated in the matrix I_m which seems inefficient, though large matrix to matrix multiplications are highly optimizable and parallelizable [6]. This linear algebra computation in Caffe is done using the GEMM function which is highly-tuned in BLAS libraries. Several studies have examined these optimizations as for example the study on OpenCL GEMM version on FPGA [7].

B. Design of the hardware accelerator

In this section, we present the hardware accelerator that was developed to run on FPGA, implemented with efficiency and speed in mind, aiming to outperform the Caffe's BLAS GEMM function which is done on CPU. In order to accelerate the GEMM function which is basically the matrix equation $C = aAB + bC$ within the PL (Programmable Logic), we had to implement the MAC operations on hardware. We designed a blocked-matrix multiplication algorithm in order to calculate the large matrices throughout the layers of the network which originally could not fit into the FPGA fabric. Our function behaves just like a GEMM function call by calculating a block each time and adding it to the final output. Two kernels were developed, the first was responsible to do the MAC operations of each block (mmult_accel) and the second to add each temporary block to the result block that would be finally added to the output matrix (madd_accel).

C. Optimization strategies

- **Data Access Optimization:** The first set of optimizations that can be made to a hardware function is to improve memory accesses and data transfer rates between the PS and PL. We allocated the matrix blocks in physically contiguous memory so that the compiler could use the most efficient data movers, hence to guarantee a fast streaming communication. What's more, we utilized the block RAM on the FPGA which is physically located near the computation by copying the transferred block matrices on temporary arrays that could fit into the PL on-chip memory, thus allowing one-cycle reads and writes.
- **Pipeline Optimization:** In order to achieve large throughput we had to enable a high degree of fine-grained parallelism in application execution within the PL fabric. We avoided data dependencies and increased the level of parallelism in the hardware implementation of the algorithms of the two kernels. Using appropriate SDSoC pragmas such as the pipeline directive, we constructed a highly parallel and pipelined architecture with minimum latency that performed the MAC operations very efficiently.
- **Dataflow Optimization:** This design technique is expressed at a coarse-grain level. We placed our kernel function calls one after the other so that SDSoC identifies the interactions between the output from the mmult_accel kernel and the input from the madd_accel kernel which are shared. SDSoC ensured the output of the second hardware function started operation as soon as data was available from the first function's output.
- **Cache Optimization:** In order to calculate each submatrix we had to load from DDR memory to the CPU cache the appropriate blocks from matrix A and B each time. With the use of the processor's cache unit we minimized the memory communication overhead by attaching the kernels to the fast cache of ARM CPU.

D. Integration with Caffe

In order to integrate the hardware accelerator with the Caffe framework and port it into the Zynq SoC we had to create it as a shared library instead of an application binary and then link it with the rest of the Caffe framework. Through the SDSoC Development Environment we were able not only to create the dynamic library but also the SD card boot image that our board would boot from. Then all we had to do was replace the Caffe's GEMM function call that runs on CPU with our GEMM function that is accelerated though the PL of the FPGA and run the Makefile with ARM toolchain in order to link the new shared library. As shown in Figure 3,

the whole process starts from the Caffe framework where the user runs the classification command on the host CPU of Zynq SoC. Then, whenever Caffe needs to make a GEMM call, it does not load the previous BLAS GEMM call which would run on CPU but instead it loads our new function (my_gemm). Following, whenever our function needs to speed up MAC operations, it communicates through AXI stream with our kernels (mmult_accel, madd_accel) passing each time the blocked matrices on the PL and sends back the result blocks. So, we efficiently send back and forth information any time needed within the PS and PL in a streaming manner. At the end, Caffe retrieves all the information from our GEMM function which returns the entire calculated array (array C).

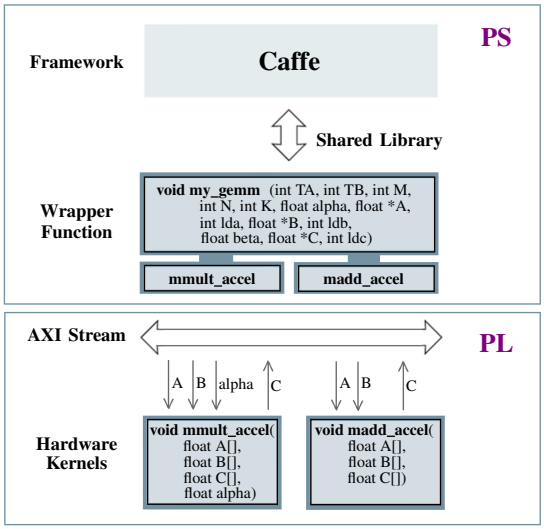

Fig. 3. Execution time per layer with Caffe profiling

V. Performance Evaluation

Our GEMM function was validated and passed several tests on multiple matrix sizes with irregular dimensions and supports alpha and beta scalars to be specified if desired. Also the classification accuracy was tested on Caffe using the CIFAR10 [8] validation set (10000 images) and remained intact at 76%. Moreover, it's worth mentioning that similar studies have explored the use of GPUs with the Caffe framework achieving significant results [9]. Below is the resource utilization of our GEMM function in the FPGA fabric.

TABLE I
Resource utilization for HW functions

Resource	Used	Total	%Utilization
DSP	162	220	73.64
RAM	32	140	22.86
LUT	38816	53200	72.96
FF	20048	106400	18.84

Also the performance of our hardware accelerator was evaluated on Xilinx ZC702 Board compared with the naive implementation of GEMM on ARM. Additionally we compared along with that, the performance of the simple GEMM approach on i3 CPU and the GEMM BLAS function on ARM.

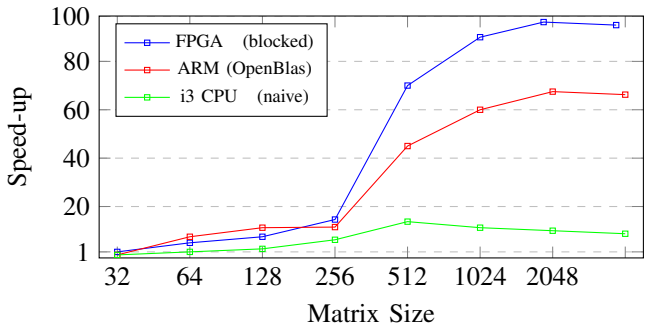

Fig. 4. GEMM function speed-up compared to the ARM version

The actual execution time was tested by measuring the numbers of cycles of the external clock of Zynq 7000 SoC which runs on 667 MHz and we found that our blocked GEMM hardware call reached up to 98× with 128×128 block size on 100 MHz kernel clock.

VI. Conclusion

In this work we presented a framework for implementing DNNs using FPGAs based on the Caffe framework. Hardware accelerators can improve significantly the performance and the energy efficiency of machine learning applications such as image classification. We described a novel scheme for the seamless utilization of the FPGA and the integration of the hardware accelerator with the Caffe Deep Learning framework and obtained successful results in terms of speed and power efficiency. FPGAs may become the platform of choice for accelerating next-generation DNNs and more and more developers are encouraged to take advantage of their benefits.

References

[1] Vivienne Sze, Yu-Hsin Chen, Joel Emer, Amr Suleiman and Zhengdong Zhang. Hardware for Machine Learning: Challenges and Opportunities. In *IEEE Custom Integrated Circuits Conference*, 2017

[2] Zynq-7000 All Programmable SoC. Technical Reference Manual, 2017

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama and T. Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014

[4] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv preprint arXiv:1602.07360,* 2016

[5] Roberto DiCecco, Griffin Lacey, Jasmina Vasiljevic, Paul Chow, Graham Taylor and Shawki Areibi, Caffeinated FPGAs: FPGA Framework For Convolutional Neural Networks, In *Distributed, Parallel, and Cluster Computing (cs.DC)*, 2016

[6] N. Dave, K. Fleming, M. King, M. Pellauer and M. Vijayaraghavan, Hardware Acceleration of Matrix Multiplication on a Xilinx FPGA, In *Formal Methods and Models for Codesign*, May 2007

[7] C. Nugteren. CLBlast: A Tuned OpenCL BLAS Library. *arXiv preprint arXiv:1705.05249*, 2017

[8] Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", 2017

[9] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro and Evan Shelhamer. cuDNN: Efficient Primitives for Deep Learning. *arXiv preprint arXiv:1410.0759*, 2014