FPGA-Based Traffic-Sign Classification

Cesar Augusto Pereira de Jesus Gouveia
cesar.gouveia@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

June 2019

Abstract

The objective of this work is to develop an efficient traffic sign classification hardware/software embedded system on a SoC-FPGA platform.

Advanced Driver Assistant Systems are a crucial component on self-driving cars. One of the main tasks of these systems is to recognize traffic signs. Modern recognition systems are typically based on Convolutional Neural Networks. Because of the safety-critical nature of these systems, the image processing task must be performed in real-time.

The SoC FPGA system developed uses a Convolutional Neural Network to perform the recognition. The network classifies 43 traffic signals, on the German Traffic Sign Recognition Benchmark, with a final accuracy of 97.2 % (after quantization) and is composed by two Convolutional and two Fully Connected layers. A dedicated multi-processor was developed to accelerate the Convolutional layers, which consume 93 % of the total execution time. The Fully connected layers are computed in software.

The proposed solution, implemented on a Zynq 7020 device, can achieve a classification rate of 8 classification/s using only one processing element, which is five times faster than a software-only solution, executing on an ARM processor embedded on the same device. The architecture is fully scalable (up to 36 Processing Elements can be implemented in a Zynq 7020 device), and with only four processing elements it is possible to meet real-time objectives (approximately $60ms$ per classification) using only 17 % of the available resources.

Keywords: Traffic Sign Recognition, Convolutional Neural Networks, Hardware/Software Co-design, Systems on Chip, Custom Hardware Design

1. Introduction

One of the more relevant tasks of ADAS is to recognize traffic signs in real-world environments. A TSR algorithm is composed of two main phases: detection (one or more traffic signs are localized in the input image) and classification (the sign is classified as belonging to one of the pre-defined traffic sign classes) [8]. This work focus on the classification phase.

One of the main challenges for the TSR system is real-time operation. The system must provide the correct sign information even at a high travelling speed. Considering an information-only system, a velocity of 180 km/h, a distance from the camera to the sign of 30 m, the overall execution time (detection + classification) must not exceed 600 ms.

CNNs are growing in size and complexity, with tens of millions of parameters and computations, which can represent a challenge, even for modern CPUs. FPGAs can provide a power-efficient embedded platform to execute such networks [6, 7, 17].

2. Background

2.1. Artificial Neural Networks

Artificial neural networks (ANN) are computational models that mimic the neurons and its interconnections present in the biological neural networks of the brain.

An artificial neuron, also called perceptron (Fig. 1), represents the basic unit of artificial neural networks. The main elements of artificial neurons are:

- Input nodes $(x_1, x_2, ..., x_n)$;
- Weighted connections $(w_1, w_2, ..., w_n)$);
- Activation function (Φ);

After collecting all synaptic input, an inner product function of the input and weight vectors (u) is created, such that:

$$u = \sum_{i=0}^{n} w_i x_i \tag{1}$$

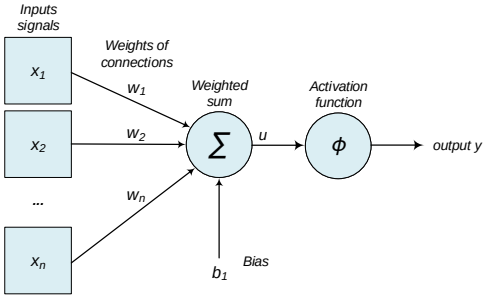

Figure 1: Perceptron.

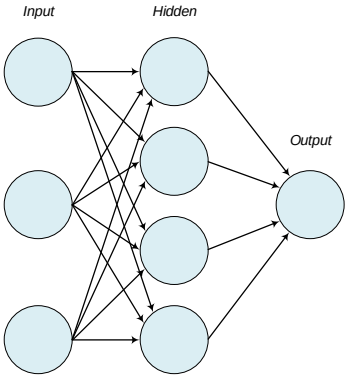

Figure 2: Artificial neural network, adapted from [12].

The most commonly used activation functions, $y = \Phi(u)$ are:

- Sigmoid:

$$\Phi(u) = \frac{1}{1 + e^{-u}} \tag{2}$$

- Tanh:

$$\Phi(u) = tanh(u) = \frac{sinh(u)}{cosh(u)} \tag{3}$$

- Rectified Linear Units (ReLU):

$$ReLU(x) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ otherwise} \end{cases} \tag{4}$$

A simple ANN is composed by an input layer (leftmost layer), a hidden layer (middle) and an output layer (rightmost), as represented in Fig. 2. An application for this network, for example, could be identifying whether a traffic sign image is a stop signal or not. A common approach is feeding the input neurons with the image pixels intensities scaled between 0 and 1. The output layer is just a single neuron and corresponds to the prediction value. If the output value is greater than 0.5 the input image is a stop signal, otherwise not.

The network above has only one hidden layer, however today's networks usually have multiple hidden layers, and are thus designated as Deep Neural Networks (DNN).

2.2. Convolution Neural Networks

In recent years, Convolution Neural Networks (CNN) have emerged as the most effective image classification architecture because they are able to take advantage of the specific spatial structure of the (bidimensional) images.

A spatial convolution is computed by convolving a source image with a small matrix filter mask (convolution kernel), which produces an output modified image (filtered). Each output (neuron) is only connected to a small block of the input image, and therefore the number of connections in a convolutional layer is much smaller than that of a fully connected layer. As such, convolutional networks are also faster to train, which allows the efficient implementation of deeper, many-layer networks.

Fig. 3 shows a general CNN structure, which is typically composed of convolution, pooling (subsampling) and fully connected (FC) layers.

Each convolution layer performs N convolutions of size K×K on each of the C input channels and outputs N feature maps. This procedure is followed by a non-linear activation function Φ and an added bias term b. Pooling (sub-sampling) layers may be inserted after a convolution layer to gradually reduce the spatial size of the feature maps and the computation load of the network. The process of down-sampling is done by selecting the maximum pixel value from a given mask, usually a 2×2 kernel size and a stride of 2.

The final stage of a CNN is typically a FC layer, responsible for the classification task.

2.3. DNN Frameworks

There are several frameworks for building DNNs, four of the most popular are: Caffe, TensorFlow, Torch and Theano. Since our objective is to use a pre-trained neural network and implement it in the FPGA, we are using Caffe which provides well-documented examples for the main CNN architectures, and a fast way for train, test and deploy models it has been our choice.

Caffe is a deep learning framework developed and maintained by Berkeley Vision and Learning Center (BVLC). It's written in C++, provides a python interface and supports cuDNN v5 for GPU acceleration [13].

The Caffe's method for storing and communicating data is through 4-dimensional arrays named blobs. Blobs are responsible for the dataflow in the network in forward and backward passes, and offer a unified memory interface for holding batches of images, parameters and parameters updates [10].

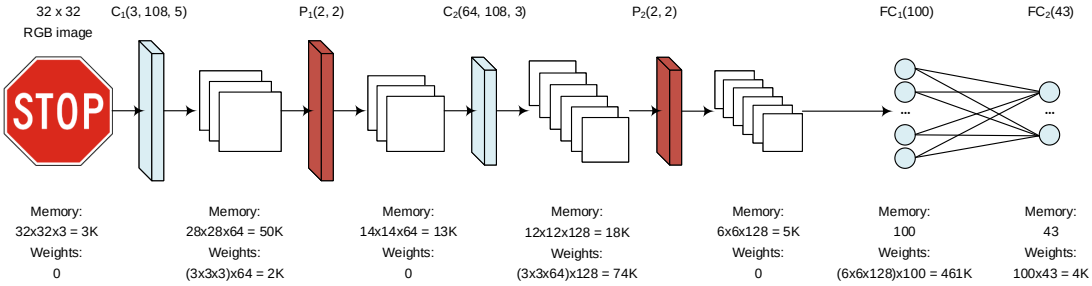

Figure 3: Example of a CNN topology with 2 convolutional layers (C1, C2), 2 pooling layers (P1, P2) and 2 FC layers (FC1, FC2).

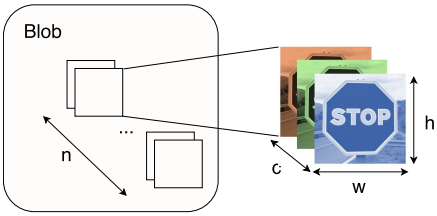

Figure 4: 4D tensor representing a set of n images with c channels and $w \times h$ size.

Figure 5: 43 GTSRB traffic sign classes (from [18]).

Fig. 4 presents a commonly approach representation of a 4D tensor (Blob) with the following parameters: image ID, channels, height and width. A Caffe layer takes one or more blobs as input and produces one or more blobs as output [10].

Ristretto is an extension of Caffe, which allows to train, test and finetune with fixed-point networks, by re-implementing Caffe-layers and simulating reduced word width arithmetic. Ristretto takes a trained model as input, and automatically produces a condensed network version, by performs multiple quantization-accuracy tests, in order to find an optimum ratio between compression rate and network's accuracy.

2.4. Fixed-point arithmetic

Fixed-point format can be used to represent fractional values after the range of the values has been studied and evaluated. The Q notation is the fixed-point format used to represent such values in this work.

A Q format value is represented the fixed-point bit format as $Q[QI].[QF]$, where QI corresponds to the number of integer bits and QF the fractional bits. The word lenght can be obtained by adding both parts ($WL = QI + QF$).

Negative values for QI represent extended resolution for fractional only numbers, as in [14]. E.g. for an unsigned number, a negative QI presents the number of leading fractional zeros.

3. TSR CNNs approaches

The German Traffic Sign Recognition Benchmark (GTSRB) is a benchmark dataset comprising a set of signals from the German traffic sign system, and including:

1. 43 traffic sign classes;
2. around 39000 training images;
3. around 12000 testing images.

Each image of the dataset, in PPM format, represents only one traffic sign. Images are not always squared and their size varies between 15×15 to 250×250 pixels [18]. The traffic signs have a simple shape, size and colour (Fig. 5), in compliance with the European Union traffic system standards and regulations, which makes it the most commonly used benchmark to evaluate TSR systems. All architectures analyzed in this section were evaluated with the GTSRB dataset.

The choice of architecture is a major factor in the recognition of traffic signals using CNNs. [5] tested 6 different convolutional network models, with different input sizes and number of convolutional layers, and consequently, different number of parameters. An architecture with 3 convolutional and 3 FC layers yielded the best results for the set of experiments performed, indicating that after a certain

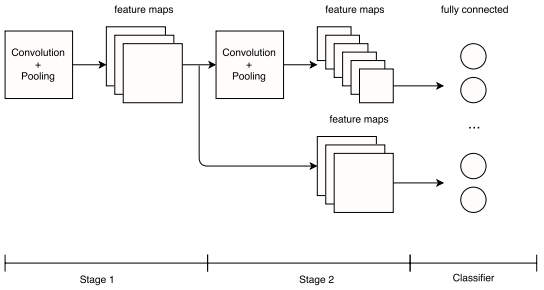

Figure 6: Multi-scale CNN, adapted from [15].

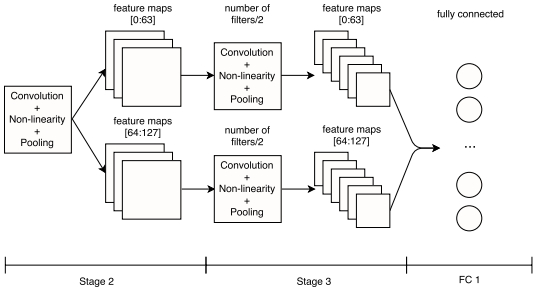

Figure 8: Two half-sized convolution layers (adapted from [9]).

number of convolutional layers the result does not improve.

Sermanet [15] and IDSIA [2, 3] teams introduced CNNs in the GTSRB competition. [15] presented a multi-scale CNN architecture, with 1,437,443 parameters. The structure combines the features maps obtained from stage one and two, and feeds them both into the first FC layer, as described in Fig. 6. The authors suggest that this type of structure benefits from combining both global and local features. A recognition rate of 98.97% was first reached, and after a few improvements on the network's architecture, they achieved a new record of 99.17%. Their final version used a 2-layer classifier with 100 hidden units, instead of the previous single-layer classifier, and used a single input channel Y (ignored color information) instead of a 3-channel input with the full image color space YUV.

[2] implemented a multi-column deep neural network (MCDNN), that used a committee of 25 CNNs (Fig. 6) to combine DNNs trained on differently preprocessed data. This architecture won the final competition phase of GTSRB 2011 with a recognition rate of 99.46% (98.16% for the single CNN architecture). It had 1,543,443 parameters, per CNN, half of which were from the FC layers.

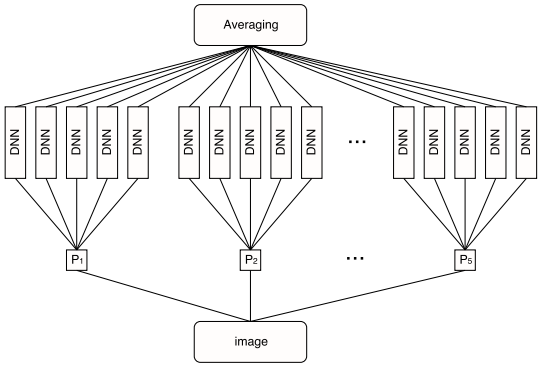

Figure 7: MCDNN, adapted from [2].

[11] used a similar multi-column architecture, with 20 CNNs, and a lower number of parameters per CNN, 1,162,284. achieving an accuracy of 98.82%. They achieved an overall accuracy of 99.65% (compared to 98.82% for the single CNN architecture).

The activation functions used in [15] and [2] were the rectified sigmoid and scaled tanh, respectively, while [11] used ReLUs. [9] used parametric ReLUs (PReLU):

$$PReLU(x) = \begin{cases} x \text{ if } x > 0 \\ ax \text{ otherwise} \end{cases} \quad (5)$$

and divided one of the convolutional layers into two individual blocks (Fig. 8), such that the inputs from stage 2 are divided into two equal parts and feed into two individual convolution-non linearity-pooling units from stage 3. Since the number of filters in each block is halved the same happens to the number of parameters of that stage.

[16] proposed a network architecture with a value 2 of stride for the convolutional layers, and an exponential linear unit (ELU) as activation function. Using a stride of 2 combines the properties of subsampling, usually implemented in the pooling layer, and makes them inbuilt in the convolutional layers. The ELU function is another variant of Equation 4:

$$ELU(x) = \begin{cases} x \text{ if } x > 0 \\ exp(x) - 1 \text{ otherwise} \end{cases} \quad (6)$$

Their TSR implementation shows a combined recognition rate of 99.94% for the tasks of detection and classification (the detection part was done using the Generalized Hough Transform - GHT), but their classifier considered only 16 traffic sign classes.

Table 1 shows the accuracy achieved by each of the referred networks, while Table 2 resumes the main characteristics of the architectures analyzed.

4. FPGA-based CNNs

This section presents some approaches for implementing design efficient hardware for CNN acceleration on FPGAs. In general, works about CNN FPGA accelerators can implement all layers of a

Table 1: Overview by date.

Accuracy	Year	Team	Method	N of Classes
99.55%	2016	[9]	CNN	43
98.82%	2014	[11]	CNN	43
99.65%	2014	[11]	MC of 20 CNN	43
98.16%	2012	[2]	CNN	43
99.46%	2012	[2]	MC of 25 CNN	43
99.17%	2011	[15]	Multi-Scale CNN	43

Table 2: Metrics.

Metrics	Sermanet [15]	IDSIA [2]	Jin et al. [11]	Aghdam et al. [9] [5]
Input size	1x32x32	3x48x48	3x47x47	1x44x44
Color Space	grayscale (Y)	RGB	RGB	Grayscale
N of CONV Layers	3	3	3	3
Filter Sizes	-	4-7	3-5	3-5
N of filters	108	100-250	70-180	64-128
Stride	-	1	1	1
Activation Function	rectified sigmoid	scaled tanh	ReLU	PReLU
POOL type	-	MAX	MAX	MAX
N of FC Layers	2	2	2	3
Hidden neurons	100	300	200	300+300
Output neurons	43	43	43	43
N of weights	1,437,791	1,543,443	1,162,284	-

CNN in hardware [7], or just the convolutional layers, leaving the fully connected to be handled by software [1].

The work in [7] proposes an architecture based on a array processing elements (PE) that processes each convolution layer independently. All the network parameters and the intermediate results of each layer are loaded/stored from/to external memory, as the on-chip buffers are not large enough to store all the required data. The size of the processing array is parameterizable, such that there are n parallel PEs where n is the number of output maps for the executing layer, and each of the PEs has m convolution engines, where m is the number of input maps.

The PEs are able to exploit parallelism at three levels,

1. PE-level: each PE produces one output feature map;
2. convolution-level: each PE has a specific (parallel) convolution engine for each of the input maps;
3. kernel-level: each kernel-weight is computed in parallel in each convolution engine.

The precision of the data representation is also parameterizable to 8-bit or 16-bit. The authors compared the accuracy of different data representations and concluded that 8- and 16-bit are adequate representations for the applications/architectures they evaluated. The pooling component of the PE can be programmed (using a simple multiplexer) to be (or not) part of the datapath. All the layers of the CNN are accelerated.

For kernel-level computation they also use the common two line sized buffers, since the convolution kernel size is 3×3 (Fig. 9).

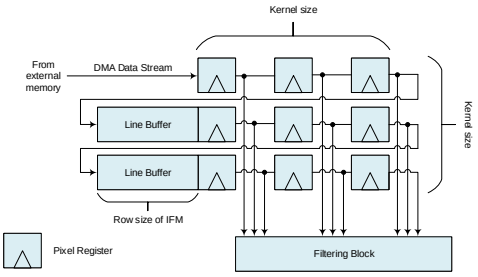

Figure 9: Kernel-level computation for a 3×3 size.

They also use a reduced precision approach to minimize the size of the data and hardware operators. They store all the intermediate results on-chip, using a factorization scheme to reduce the in-

termediate memory requirements, which may limit the complexity of the networks than can be implemented. Their approach focus only on the acceleration of the convolutional layers, since it is the most computationally intensive and corresponds to 90% of the CNN execution time, for a typical neural network implementation [4].

5.
Evaluation of network structure

This section describes how the GTSRB dataset was prepared for training, the training flow used and the training results (accuracy) obtained.

5.1. Network Training

The network training is divided into 2 phases, the dataset preparation and the training phase. In order to prepare the two network models for training, a sequence of 4 steps was followed, for each of them. First, the input dataset was prepared before training: splitted, augmented and preprocessed. Then, the network was trained using Caffe and 32-bit floating-point values, for both weights and activations. To find the optimal reduced-precision configuration for the weights and activations, the network was quantized using Ristretto. Finally, the networks were finetuned, that is, retrained using the reduced-precision values obtained from quantization.

5.2. Dataset preparation

The dataset preparation starts by dividing the initial dataset into a training dataset and a test dataset. Both train and test sets were obtained from the same distribution, so the test set was used as the validation set. The next step was to produce two augmented train datasets. Each dataset was augmented by upsampling all classes to 3000 samples, in order to standardize the contributions of each class, providing a total of 129000 input images per dataset. The classes are upsampled by performing a series of image transformations recommended by each author model [11, 15]. The Jin augmented dataset was produced using random samples from each class and performing three random transformations for each sample: random translation within 10% of the image size, random rotation between -5 and 5 and random scaling using a factor between 0.9 and 1.1. The Sermanet augmented dataset follows a similar approach, as samples are randomly perturbed in position ([-2,2] pixels) and the rotation varies between -15 and 15. In the last step, all images were preprocessed, including the augmented train dataset and the test set. The images were down-sampled or up-sampled to 47x47, for the Jin's model, or 32x32 pixels for Sermanet. Then, a histogram equalization was performed for both models. At the end, the final images are in RGB color space and mapped to [0, 1] pixel range.

5.3. Quantization Analysis

A quantization analysis was performed using Ristretto to find the fixed-point configuration that produces the best tradeoff between accuracy and resources consumed. The accuracy of the networks was evaluated to different word sizes of the weights and activations.

Table 3 compares the amount of resources (in KB) for the top-5 accuracy configurations, considering only the memory resources related to the maximum activations per stage, and the weights as a ping-pong buffer twice the size of the largest filter in each network, such that

$$Total_{memory} = Max_{Activations} + (Max_{kernel} \times 2). \quad (7)$$

Comparing the two architectures with greater precision, the one with the lowest number of resources consumed corresponds to the 8-16 Sermanet configuration. Given the results, the network chosen for implementation corresponds to the Sermanet CNN with Q-1.7 configuration for weights and Q8.8 for activations. The fixed-point networks were fine-tuned in Ristretto, to evaluate if an additional accuracy improvement could be obtained, but no further gain was found.

6. CNN Hardware Architecture

This section describes the architecture of the hardware accelerator that implements the convolutional stages of the Sermanet CNN. The Processing Elements are explored, by varying the number of MACs within each one, in order to understand which configuration(s) provide the best tradeoff between performance and device resources.

7. Convolutional Accelerator

The Convolutional Accelerator is responsible for computing stages 1 and 2 of the Sermanet network, and concatenate the OFMs from both stages. The block diagram in Figure 10 gives an overview of the proposed accelerator. Each PE implements the following operations: convolution, activation function (ReLU) and subsampling (max-pooling). All the input and output images are stored on-chip, while weights and biases are stored on the external memory and read as needed. The weights are stored in a ping pong buffer, in order to maximize the bandwidth, and the biases are loaded to local registers in the PE. The activations are stored in the shared image buffers A and B, and broadcasted to all PEs as M words of 16-bit, where M is the number of MACs in each PE. On the first stage, RAM A acts as an input buffer and B as an output. They change roles in the second stage where RAM B feeds the PE Cluster with the IFM and RAM A receives the OFM. All biases from both stages are stored in a

Table 3: Memory requirements of Top-5 accuracy configurations.

Configuration (x-y)	Accuracy	Memory [KB]
8-16 (Sermanet)	97.2%	$5.4 + 24 \times 2 = 53.4$
16-16 (Sermanet)	97.2%	$5.4 \times 2 + 24 \times 2 = 58.8$
8-8 (Jin)	96.3%	$2 + 50 = 52$
8-16 (Jin)	96.5%	$2 + 50 \times 2 = 102$
16-16 (Jin)	96.5%	$2 \times 2 + 50 \times 2 = 104$

Note: x-y, where x is the word length (in bits) for weights and y for activations.

ROM and read as needed. The control Unit is responsible for generating the addresses for accessing the shared memories A and B, and for the local weight memories of the PE. The control unit also generates 4 control signals that inform the system of the end of processing a single pixel, a pooling kernel, a filter and a full convolutional layer.

Two levels of parallelism are implemented in the PE cluster: PE-level and pixel-level. All PE share the same IFMs, which are broadcasted to the different PE in parallel. The PE cluster, calculates different OFMs in parallel, using a different filter in each PE. Within each PE, different MAC units calculate partial pixels on different IFM-weights combinations.

The PE implements convolution over depth, where depth corresponds to the number of input images or IFM. This method facilitates control, as the pixels are accessed in memory in consecutive positions.

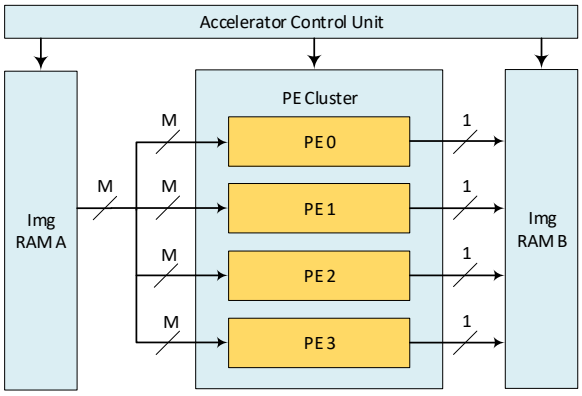

Figure 10: Accelerator architecture.

7.1. Processing Element

The PE module implements the convolutional, activation and subsampling layers (max-pool). Each PE is composed by: a local RAM memory, used to store the weights related to each OFM; a MAC Cluster (one or more MAC units in parallel), which multiplies a set of weights with a set of pixels from the current IFM; an adder cascade to sum all the partial results from the MAC cluster; and a BRP unit, which is responsible for adding the bias term, applying the activation function (ReLU) and subsampling (max-pooling) the resulting pixels. Finally, the output from the PE is rounded to 16 bits (Q8.8).

7.2. PE Configuration

The PE architecture was explored by analyzing different PE configurations with 1, 2, 4, 8 and 16 MACs, mapping the more complex Stage 2 of the network. Only configurations with a power of 2 number of MACs were considered. For the following tests a base architecture of 1 PE which maps stage 2 was used. These configurations were analyzed in terms of utilization and timing. An estimation of the utilization is calculated for multi-PE clusters normalized to produce 16 MACs/cycle each. When a multi-PE Cluster is used the RAM A and B, and the accelerator control addresses are shared between all PEs and the processing resources are multiplied by each PE. Figure 11 shows how the percentage of the two most critical resources (BRAMs and DSPs) varies for the different configurations. The results also show that the PE Cluster configuration that requires the lowest number of BRAMs is a PE cluster with 4 PEs of 4 MACs.

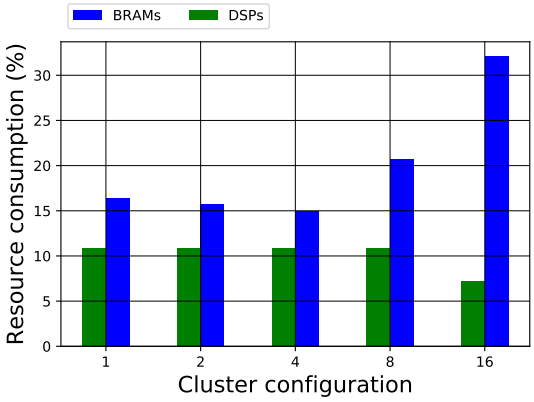

Figure 11: BRAMs and DSPs consumption for different PE configurations.

8. Traffic Sign Recognizer

This section describes the Hardware/Software architecture proposed and developed for the traffic sign recognizer and evaluates the results obtained. The experimental results are presented and evaluated (hardware resources and performance analysis) for a 4-PE architecture.

8.1. Hardware/Software Architecture

The traffic sign recognizer is composed by two convolutional and two FC stages. The system is organized so that the convolutional stages are mapped using the dedicated hardware accelerator and the FC stages in software, using one ARM processor.

The input images and weight filters are read by the DMA from the DDR memory and provided to the accelerator. This DMA block is also responsible for receiving the OFMs from the final stage and writing them back to the DDR memory. All DMA read transactions are issued by the master ARM core, using the AXI4-Lite interface using the pooling mode. Multiple data transfers are performed in order to send the filters as needed. In the second stage, multiple filters of 2800 elements must be sent. Due to the large amounts of data involved, the AXI4-Stream interface is used to provide fastest data transfers between the custom AXI4 Stream based accelerator and the external memory. The HP AXI4-Stream port is configured for 64-bit and each stream element is composed of 4 16-bit pixels (for the input image or the IFMs) and 8 8-bit weights (for the filters of each stage). The hardware block directly accesses the external memory using one DMA block. The DMA commands are controlled by the software running on the ARM.

8.2. 4-PE Architecture

An architecture with 4 PEs was implemented...

Table 4 shows the resource utilization of a 4-PE architecture and its main sub-blocks. The PE cluster is composed by 4 PEs, so the resources relative to a single PE are replicated.

Table 5 shows the total execution time for the software-only solution (O2 optimizations), and two hardware/software solutions (O2 optimizations) for 1-PE and 4-PE. The 4-PE hardware/software application performs a traffic sign classification in 61 ms, which is approximately 9 times faster compared to a software-only application. The convolutional stages now represent only 40% of the total execution time of the classification algorithm.

Table 4 shows the resource utilization of a 4-PE architecture and its main sub-blocks. The global controller has slightly increased the number of resource usage due to the additional logic implemented in the FSM (Figure **??**). The PE cluster is composed by 4 PE, so the resources relative to a single PE are replicated. Moreover, a one-hot shift-register was added to the PE cluster. RAM A and B had a small resource utilization increase due to their extended write port widths.

Table 5 shows the total execution time for the software-only solution (O2 optimizations), and two hardware/software solutions (O2 optimizations) for 1-PE and 4-PE. The 4-PE hardware/software application performs a traffic sign classification in 61 ms, which is approximately 9 times faster compared to a software-only application. The convolutional stages now represent only 40% of the total execution time of the classification algorithm.

9. Conclusion

The work developed during this master's thesis focused on the investigation and development of an efficient TSR hardware/software embedded system on a SoC-FPGA platform. The proposed solution achieves a classification rate of 8 classification/s using only one processing element, which is five times faster than a software-only solution, executing on an ARM processor embedded on the same device. The architecture is fully scalable (up to 36 Processing Elements can be implemented in a Zynq 7020 device), and with only four PEs it is possible to meet real-time objectives (approximately 60 ms per classification) using only 17 % of the available resources.

Several CNNs for TSR were analyzed in terms of accuracy and memory requirements. The training dataset (GTSRB) was augmented and standardized by upsampling each class using a set of image transformations (e.g. rotation, translation). A quantization analysis was performed on both networks in order to find the fixed-point configuration that produces the best trade-off between accuracy and memory resources. The Sermanet network provided the best results (97.20%) and the best configuration found was 8 bits (Q1.7) for the weights and 16 bits (Q8.8) for the activations.

In this thesis, a hardware/software architecture for executing the TSR CNN algorithm was proposed and demonstrated on a Xilinx Zynq-7020 device. The solution takes advantage of the pre-built ARM CPU as well as of the available FPGA resources. The computation is distributed among software and hardware domains, such that the most computationally demanding task of the TSR CNN algorithm was implemented in the hardware domain: the convolutional stages. The remaining tasks are implemented in software. The computational is parallelized among several PEs: the PEs calculate different OFMs in parallel and each PE calculates 4 partial pixels (each PE has 4 MACs) on different IFM-weights combinations.

An architecture with 4 PE of 4 MAC was then implemented which obtains 16 classifications per sec-

Table 4: Resource usage for the main sub-blocks of the 4-PE convolutional accelerator.

Resource	Global Controller	Accelerator Controller	PE cluster	IFM/OFMs RAMs	Convolutional Accelerator
LUTs	401(< 1%)	290(< 1%)	449(< 1%)	76(< 1%)	1234(2%)
FFs	42(< 1%)	206(< 1%)	323(< 1%)	4(< 1%)	653(< 1%)
BRAMs	0(0%)	0(0%)	8(6%)	13(9%)	21(15%)
DSPs	0(0%)	0(0%)	24(11%)	0(0%)	24(11%)

Note: The resource utilization percentage is calculated for one Zynq-7020 device.

Table 5: Execution Times.

Stage	Layer	SW-only (O2) [ms]	HW+SW (O2) for 4 PEs [ms]
2*STAGE 1	CONV1-108	68	25
	POOL1	2	
2*STAGE 2	CONV2-108	412	
	POOL2	0.2	
CONCAT		0.2	
Execution Time for the Convolutional Stages		482	25
STAGE 3	FC1-100	36	36
STAGE 4	FC2-43	0.06	0.06
SOFTMAX		0.02	0.02
Total Execution Time		518	61

ond using only 17% of the target device resources. This represents 1/9 of the total execution time for both detection and classification, leaving a margin of 500 ms to carry out the detection.

Acknowledgements

I would first like to thank my supervisor Prof. Horcio Neto for all the helpful insight, guidance and support through the development of this thesis.

I want to thank Prof. Rui Duarte and Mrio Vestias for their encouragement and insightful comments.

Also, I would like to thank INESC-ID for providing me the tools to develop the experimental work for this thesis.

A big and heartfelt thank you to my family, my girlfriend and my colleagues for all the support given to me during the course of this work.

References

[1] K. Abdelouahab, M. Pelcat, J. Serot, C. Bourrasset, J.-C. Quinton, and F. Berry. Hardware Automated Dataflow Deployment of CNNs. (June), 2017.

[2] D. Cire\csan, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32(January 23, 2012):333–338, 2012.

[3] D. Cires, U. Meier, and J. Masci. A Committee of Neural Networks for Traffic Sign Classification.pdf. 1(1):1918–1921, 2011.

[4] J. Cong and B. Xiao. Minimizing Computation in Convolutional Neural Networks. pages 1–12, 2014.

[5] J. Fulco, A. Devkar, A. Krishnan, G. Slavin, and C. Morato. Empirical Evaluation of Convolutional Neural Networks Prediction Time in Classifying German Traffic Signs. (Vehits):260–267, 2017.

[6] S. Ghaffari and S. Sharifian. FPGA-based convolutional neural network accelerator design using high level synthesize. *2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS)*, pages 1–6, 2016.

[7] 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*, 0070(c):1–1, 2017.

[8] H. Habibi Aghdam and E. Jahani Heravi. *Guide to Convolutional Neural Networks*. 2017.

[9] H. Habibi Aghdam, E. Jahani Heravi, and D. Puig. A practical approach for detection

and classification of traffic signs using Convolutional Neural Networks. *Robotics and Autonomous Systems*, 84:97–112, 2016.

[10] 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.

[11] J. Jin, K. Fu, and C. Zhang. Traffic sign recognition with hinge loss trained convolutional neural networks. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):1991–2000, 2014.

[12] M. A. Nielsen. Neural Networks and Deep Learning. In *Neural Networks and Deep Learning*, pages 875–936. Determination Press, 2015.

[13] NVIDIA. Deep Learning Frameworks — NVIDIA Developer.

[14] E. Oberstar. Fixed-point representation fractional math. 1, 01 2007.

[15] P. Sermanet and Y. Lecun. Traffic sign recognition with multi-scale convolutional networks. *Proceedings of the International Joint Conference on Neural Networks*, pages 2809–2813, 2011.

[16] A. Shustanov and P. Yakimov. CNN Design for Real-Time Traffic Sign Recognition. *Procedia Engineering*, 201:718–725, 2017.

[17] A. Solazzo Matr, M. Domenico Santambrogio, and G. Carlo Durelli. An automated design framework for FPGA-based hardware accelerators of Convolutional Neural Networks. 2016.

[18] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition. *Neural Networks*, 32:323–332, 2012.