1 2 9 0

UNIVERSIDADE Đ
COIMBRA

Bruno Gonçalves Oliveira

EXPLORING ENERGY EFFICIENT OBJECT CLASSIFICATION ON RECONFIGURABLE LOGIC

VOLUME 1

Dissertation submitted to the Department of Electrical and Computer Engineering of the Faculty of Science and Technology of the University of Coimbra in partial fulfilment of the requirements for the Degree of Master Science

July 2019

1 2 9 0

UNIVERSIDADE Đ COIMBRA

BRUNO GONÇALVES OLIVEIRA

EXPLORING ENERGY EFFICIENT OBJECT CLASSIFICATION ON RECONFIGURABLE LOGIC

Thesis submitted to the

University of Coimbra for the degree of

Master in Electrical and Computer Engineering

Supervisor:

Prof. Doctor Jorge Nuno de Almeida e Sousa Almada Lobo

Coimbra, 2019

This work was developped in collaboration with:

University of Coimbra

Department of Electrical and Computer Engineering

Institute of Systems and Robotics

Dedication

Dirijo desde já, um agradecimento ao meu orientador, o Professor Doutor Jorge Lobo do Departamento de Engenharia Eletrotécnica e Computadores da Faculdade de Ciências e Tecnologias da Universidade de Coimbra, pela constante transferência de conhecimento e mentoria. À minha família, não só pelo apoio incondicional ao longo de todos estes anos, mas também por me fazerem acreditar nas minhas capacidades. À minha mãe, à pessoa mais importante da minha vida, queria expressar a minha eterna gratidão e reforçar que todas as minhas conquistas também são dela. Gostaria ainda de deixar uma nota de motivação ao meu irmão Rodrigo, para que quando entrar no mesmo curso, consiga um registo académico de excelência e nunca pereça durante as dificuldades que lhe advém. À minha namorada Carla, quero enaltecer o seu enorme apreço, a sua constante motivação e perseverança que no decorrer dos anos sempre me presentiou. Ao pessoal do laboratório, gostaria de agradecer o bom ambiente de trabalho vivido e a entre-ajuda existente. A todos os amigos que me acompanharam desde sempre, um obrigado especial por tornarem os momentos que vivi nesta academia inesquecíveis.

"All we have to decide is what to do with the time that is given to us."
– Gandalf, from Lord of the Rings: The Fellowship of the Ring.

Acknowledgments

To the Institute of Systems and Robotics(ISR), my most sincere gratefulness for providing the necessary resources to conduct this research work.

I would also like to thank Sundance Multiprocessor Technology, Ltd for giving me access to the VCS-1. The gratitude also ranges Pedro Machado, from the Nottingham Trent University, for support on the hardware usage.

My final thanks, goes to the creators of the BNN-Pynq Framework for providing the scientific community with a very useful developing tool. It was a good starting point for this work.

Abstract

Object classification is a problem with great relevance in computer vision since it can integrate a wide range of target applications, such as agriculture and security. At present, there are a set of solutions that solve this problem, the most successful relying on neural networks. Among the best known are GoogleNet, AlexNet and YOLO. However, the underlying processing requires high performing computational platforms such as GPUs, CPU clusters, or custom ASICs. Apart from ASICs, that have a high cost but are less generic, they are typically high power and not well suited for embedded systems. However, there has been some progress in low power approaches, driven in part by the smartphone and tablet market, and heterogeneous platforms are now available that explore a mix of architectures (CPUs and GPUs) with reconfigurable logic (FPGAs). In this work, we propose implementations of lightweight convolutional neural networks in hybrid platforms, thoroughly exploring the design space, the classification performance and the power efficiency. The underlying algorithm is analysed, and key components for concurrent and parallel computation identified. Mappings of this to the heterogeneous platform will be explored, ranging from a baseline CPU implementation to a full custom implementation maximising the use of the available resources. A set of metrics is considered for the evaluation of the different configurations. In the end, we achieved object classifiers with different characteristics running in two low-power devices. Analyses performed on the implementations supported the reliability of compression a convolutional neural network to fit on the target device, through the reduction of the precision of its calculations.

Resumo

A classificação de objetos é um problema com grande relevância em visão por computador, uma vez que pode ser integrada em um enorme conjunto de aplicações alvo, tais como agricultura e segurança. No presente, existe um conjunto de soluções que resolvem este problema, sendo que, as que possuem maior sucesso, dependem de redes neuronais. As mais conhecidas são o GoogleNet, a AlexNet e o YOLO. No entanto, o seu processamento subjacente requer plataformas de alta performance tais como GPUs, clusters de CPUs ou ASICs customizados. Excluindo os ASICs, que tem um elevado custo, mas são menos genéricos, elas tipicamente têm um elevado consumo energético e não são adequadas a sistemas embebidos. No entanto, tem havido progresso em abordagens de baixo consumo devido em parte ao mercado dos smartphones e tablets, estando disponíveis mixes de arquiteturas (CPUs e GPUs) com lógica reconfigurável (FPGAs). Neste trabalho, propomos uma série de implementações de redes neuronais quantizadas em plataformas híbridas, explorando completamente o espaço de design, a performance de classificação e a eficiência energética. O algoritmo subjacente é analisado, e os componentes chave para computação concorrente e paralela identificados. O mapeamento na plataforma foi explorado, desde a implementação CPU base até uma completamente customizada que maximiza o uso dos recursos disponíveis. Um conjunto de métricas é considerado para a avaliação das diferentes configurações. No final, conseguimos classificadores de objetos com diferentes características a correr em dois dispositivos de baixo consumo. As análises realizadas às implementações suportaram a fiabilidade da compressão de redes neuronais de convolução para caber nos dispostivos alvo, através da redução da precisão dos seus parâmetros.

List of Figures

List of Tables

Contents

1

Introduction

1.1 Motivation

In recent years, a wide range of solutions has proved to be suitable for applications that depend on object classification due to the development of faster processing units and reliability in the precision levels. Currently, the branch of deep learning, has generated agreement between researchers as being the most effective, being possible to obtain precisions in the order of 85% [16] in some validation datasets. In fact, several aspects, as for example, the availability of labelled data sets and powerful GPUs capable of process trainings with those batches of images in practical times, are contributing to the highlight of this branch. Methods such as the unsupervised clustering have not had so much focus due to the dependence of other models to perform the classification of the objects [2].

The domain of Deep Learning, Convolutional Neural Networks (CNNs), makes use of sets of filters, commonly referred as layers, in order to extract the features of the input before proceeding with the classification. Increasing the number of layers to store more object features, and thus get more precision, has problems associated. Among them, the amount of space occupied in the storage unit by the network, the latency in accessing data and the number of devices capable of processing them in real time. Therefore, to overcome these restrictions, sophisticated devices like high-end GPUs or CPUs clusters are needed. To sum up, the current state of the art in image processing algorithms has impressive results but relies on substantial computation resources. Thus, embedded computer vision applications have been limited due to low power and performance constraints.

One of the solutions to put this type of algorithms in embedded platforms is to perform a compression of the neural network parameters. A wide range of techniques such as prunning [15], quantization [27], binarization [21] or a combination of a few of them are presenting good results in reducing the size, while maintaining acceptable losses of accuracy. Yet, in the particular case of final user critical applications, it becomes necessary to analyse the trade-off between size and precision before launching, to avoid risky situations.

In terms of hardware, new platforms that can accommodate these complex networks are arising. Pushed by mobile consumer market devices, there is now a wider spectrum of low power mixed solutions. These devices combine traditional architectures with Field Programmables Gate Arrays (FPGAs) enabling high-performance embedded heterogeneous platforms. A design space exploration can be done to optimise the efficiency of the algorithms targeting the best performance or the best low power performance.

1.2 Objectives

The objective of this work is to have low-power implementations of convolutional neural networks (CNNs) suitable for image classification on embedded systems. To pursue this we will focus on heterogeneous platforms that combine low-power CPU+GPUs+FPGA. This will allow the exploration of a diverse solutions space and define trade-offs, to achieve low-power embedded solutions for image classification.

1.3 Related Work

The constant growth of CNNs complexity increased the level of accuracy in object classification. Consequently, there has been a limitation in the range of devices capable of storing the networks. For that reason, the scientific community has been trying to reduce the size occupied by CNNs. Of all possible methods, the related works that converge with this are the ones that address parameter binarisation and quantisation.

In [6], it was demonstrated that high precision parameters are redundant in achieving acceptable values of confidence in object recognition. With only the quantisation of the fully-connected layers, they lost 1% of accuracy rating when reducing the network size in around 20%. These results were also supported by [22] when they explored CNNs for mobile devices.

An even greater compression can be obtained through the binarisation of the network weights and activations. In [19] they manage to reduce size while improving the speed of computing the convolution layers. In the final, they achieved a network compression in the order of 30%. Two state-of-the-art object classifiers were used as a test-bed of the binarisation. The accuracy losses verified were about 20% higher than the quantised versions. However, the space occupied by binarised networks makes them more suitable for use in embedded devices.

Researchers had to use compression techniques to place well-known object detectors in embedded devices, for example, a lightweight version of YOLOv2 [18] was implemented on a similar target platform used in our work. They used a binarised Convolutional Neural Network(CNN) [21] to extract features and Support Vector Regression Machines(SVR) [5] for location and classification. The overall system was feasible, but limited due to compromises. The resulting accuracy might not suit some critical applications. Above all, we could observe the versatility of one of the target devices our work in the processing of complex networks.

Another attempt to apply the YOLO in embedded systems was done by [10] using a NVIDIA Jetson-Tx2. They also needed to perform optimisation techniques, namely Tucker decomposition [26] and 16-bit quantization. However, the version implemented is already outdated due to the launch of new ones with more classes and precision.

CNNs network size is not the only factor that restricts the range of target platforms. The high workload that this type of algorithms imposes on the processing units leads to a large energy consumption. This can be a limiting factor not only on battery power embedded systems, but also on big data centres where the total energy requirements become prohibitive. Therefore power efficiency of CNNs computations has been a hot topic of research.

In [25], the authors explored FPGA implementations of CNNs to achieve better energy efficiency. Two CNN based object detectors were used to explore the design space of a cluster of FPGAs. While they show significant gains in energy efficiency, the power level of the more efficient solution is well suited for datacenters and not to embedded solutions. Nevertheless, it has been shown that it is possible to minimise the energy consumption of complex CNNs using custom circuits.

Recent heterogeneous platforms provide low power consumption and parallelisation of critical parts. In [23], they exploit CNN implementations for the mobile phone market, where there is a demand for applications that make little drainage on the batteries. A simple object detector was implemented as a use case, obtaining a considerably low consumption. The computational performance obtained was very limited, restricting its use by complex CNNs. Yet, an object detector was implemented successfully, thus supporting the development of this type of algorithms on low-power devices.

The series of works presented shows that there are trade-offs in the development of CNNs between computational performance, size, power consumption and precision. For each application, it is necessary to find a balance between these metrics.

Despite the growing demand for CNNs to use on embedded applications, there are still tremendous difficulties in developing the algorithms for this type of devices. To overcome this problem, in [21], they have created a framework that streamlines the development of CNNs for embedded heterogeneous systems. The tool receives binary Caffe models [8] as an input and then automatically generates bitstreams for the FPGA. Thus, it is possible for people without hardware knowledge to be able to develop CNNs on low-power solutions. Since Caffe can be a difficult tool for beginners to handle, a new branch of the framework was created, the BNN-Pynq [21]. This new framework allows to explore some examples of neural networks, perform trainings with quantised weights and synthesise prefabricated overlays for target devices like the Xilinx Pynq or Xilinx ultra96 family boards. However, it is not clear how to create new overlays for networks with different configurations from the examples. This need to understand hardware in order to create new implementations distances software engineers from this tool, something that was not its purpose. Another limitation is that it only allows the generation of solutions for two specific target devices, a small range compared to the number of heterogeneous platforms with reconfigurable logic on the market.

While there are limitations to the BNN-Pynq Framework, it is a good starting point and we will use in our work, and build upon it to expand to more target platforms, in our pursuit to explore the solution space for low power solutions for CNNs on embedded systems.

1.4 Key Contributions

In summary, the key contributions are:

- Creation of a test bed Convolutional Neural Network;
- CNN training with 4 different types of weight and activation quantisations to classify among 10 object classes;
- Mappings of the networks on the Sundance VCS-1, a device that contains the zu4EV as processing unit, and on the Pynq-Z2, a platform composed by the Zynq-7020.
- Comparison of the test bed CNN with different bit quantisations through metrics of performance, power consumption, resources usage and accuracy;
- Benchmark the performance of the CPUs of the different target devices;

1.5 Overview of the Dissertation

Chapter 1 presented the motivation of this work, the main objectives, the related works and key contributions to the scientific community from this dissertation. Then, in chapter 2, it is explained the theoretical knowledge behind the work developed, more specifically the operations involved with CNNs and the heterogeneous devices used. In chapter 3, it is explored the work carried out and its details. In 4, the obtained results are evidenced. The analysis and discussion of the results is done in chapter 5. At the end, we present the conclusions and possible future works that can come out from this one.

2

Background on Convolutional Neural Networks and Heterogeneous Devices

Given the depth and extensiveness of this work, it is necessary to introduce some concepts. Thus, the following sections explain the operations involved in convolutional neural networks, the tools used to develop low-power solutions and the devices used in this work.

2.1 Convolutional Neural Networks Operations

Convolutional Neural Networks(CNNs) were proprosed in late ’80s by [13], for an application that performed character recognition in zip codes. More recently, classifications of images with more complex objects were obtained at low computing times in [12]. These networks are composed by several types of layers, each one of them having specific functions such as feature extraction, scaling or classification. In this work, we will only put in the hardware accelerators the convolutional layers, max-pool layers and the fully connected layers, so for that reason, we will only make a brief explanation of those. Further details of their operations can be found in [7]. Figure 2.1 shows as an example of a CNN that classifies among 3 objects classes, composed of 3 convolutional layers, two of max-pool and two fully-connected.

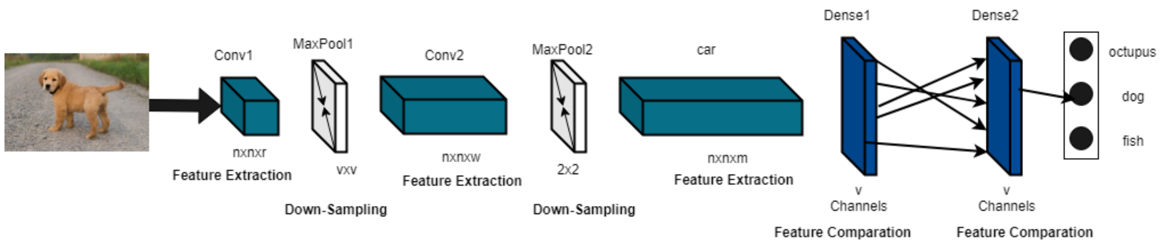

Figure 2.1: Example of a convolution neural network.

2.1.1 Convolutional Layers

All convolution layers are composed by a determined number of filters, with the same width and height. In other words, a layer can be observed in three dimensions, first dimension for the width, second dimension for the height and the third for the number of filters. The set of values that compose each point of this volume are obtained by training the network. The filters that make up the layers slide along the input to extract the features needed for the object classification. Each output pixel is obtained by the formula:

$$O_n(x,y) = f_{act}\Bigg(\sum_{c=0}^{C-1}\sum_{h=-\frac{H-1}{2}}^{\frac{H-1}{2}}\sum_{w=-\frac{W-1}{2}}^{\frac{W-1}{2}} I(x+w,y+h,c)K_n(w,h)\Bigg), \tag{2.1}$$

O_n(x,y) is the output pixel obtained by the convolution of the n_{th} filter of the layer, K_n, with the input image I. The number of channels in the input is represented by C, WxH are the filter dimensions and f$_{act}$ corresponds to the activation function. More details of this operation are shown in figure 2.2.

The convolution with stride occurs like the one previously seen, except for the displacement of the filter. In this case, the filter does not slide contiguously, as show in 2.3.

In the first iteration, represented by the red colour, the first pixel is processed. In the following iteration, represented by the green colour, the next pixel to the first one processed is ignored and the next to it is computed. Thus, in the case depicted the convolution window is moved two by two pixels, reducing the size of the output frame because of the ignored pixels.

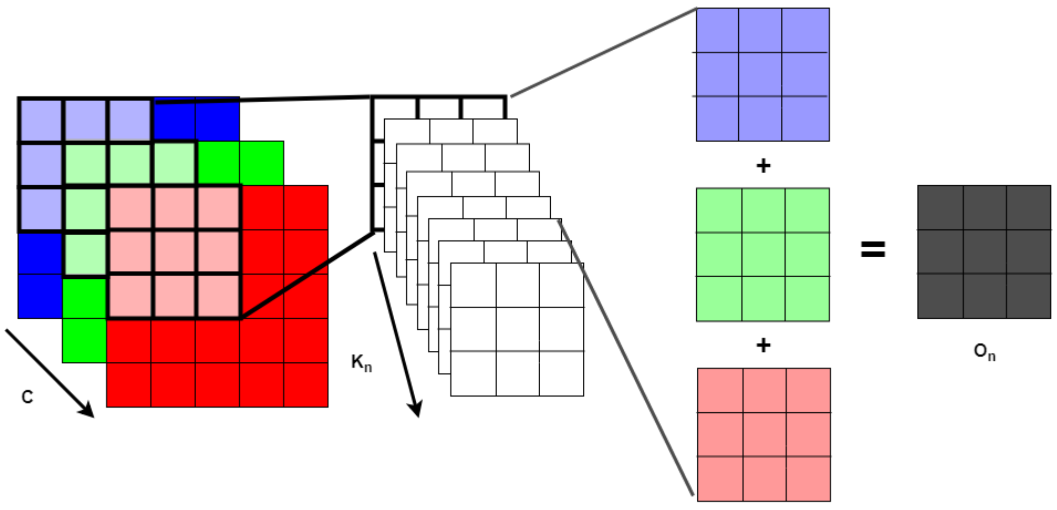

Figure 2.2: Convolution operation between an input with C channels and a layer with K_n filters. Represented the output of one of the filters.

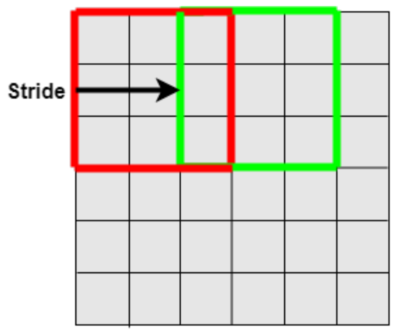

Figure 2.3: Convolution operation with stride. The filter slides according to the corresponding stride passed as input. In this example, from two by two pixels.

Convolution operations are an efficient method for the extraction of features in an input. However, [3] evidence that convolution operations occupy over 90% of total system computation. This leads us to focus attention on accelerating convolution layer operations in hardware.

2.1.2 Max-pool Layer

A window slides over an input, with a specific stride, selecting the largest value of the samples for output. Output size depends on factors such as stride and border ignorance. Figure 2.4 shows the max-pooling of an 8x8 input frame by a 2x2 filter with a stride of 2.

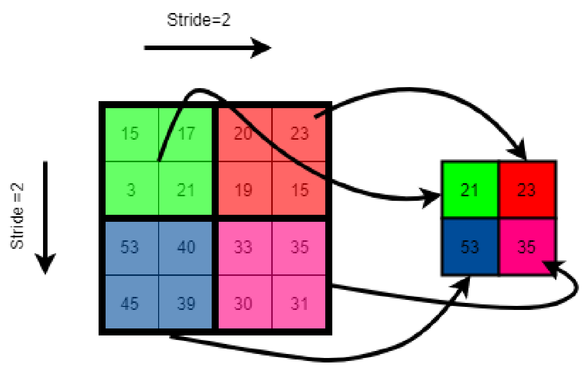

Figure 2.4: Maxpool operation with stride of 2 for an 8x8 input. It outputs a 4x4 frame.

2.1.3 Fully Connected Layer

Features of the objects are obtained after an input image passes through pooling and convolution layers. It remains to make a comparison with the features of the classes that the network was trained to classify. This operation is performed by the fully connected layer.

2.2 Training Convolution Neural Networks

Important to mention that this research does not seek to improve the networks training process. But, there is a need for correct weights to test the implementations, so the training in [4] was followed. In this section, we will discuss the parameters that shaped the training performed.

An epoch is the number of times that a set of samples propagate forward and backwards through the network.

The batch size specifies the number of samples that propagate through the network in each forward and backward iteration during the training phase. The amount of images that compose the data set needs to be divisible by the batch number, in order to avoid problems. This parameter also has a dependence on the training device since larger batches require more memory to accommodate the samples.

The loss function evaluates the current solution results against the true values.

Learning rate restricts how much the network weights change according to the loss function. For example, small values in this parameter cause fewer changes in weights over the epochs. This variable is represented by alpha and in our model of training is not static throughout the epochs. Therefore, it will suffer a decay according with a rule.

2.3 Hardware Development

2.3.1 Vivado High-Level Synthesis

The inability to depend on the clock frequency to decrease the computation time of the programs triggered the manufacturers' interest in launching products with parallel processing. Thus, to take advantage of the devices, it is necessary to structure the programs by the existing resources. Vivado HLS allows the creation of custom functions through languages such as C and C++, to facilitate access to the heterogeneous components and interfaces of Xilinx devices. In addition, it also contains several libraries with widely used functions.

2.3.2 Xilinx Modules

Xilinx provides their users with functions that optimise access to the resources of its FPGAs. They are called IPs and allow to substantially reduce the development time of solutions. There are several types of it, each with different purposes. In any case, we will only address those that were used in the final implementation.

Processor System Reset Module IP provides specific reset signals for each system module. Therefore, it is very useful to control modules with different activity times.

The Processing System 7 component makes a logical connection between the on-chip and off-chip parts of the device. Thus, allowing to map solutions between devices with traditional architectures and reconfigurable logic.

Axi Interconnect performs the exchange of information between master and slave memory-mapped devices.

2.4 Quantized Neural Networks (QNNs) on Reconfigurable Logic

As said in 1.1, the state of the art in image classification uses CNNs with millions of floating point parameters. However, in [6] it has been demonstrated that high precision in these elements is redundant and its elimination does not affect significantly the classification. So, the quantisation of these parameters can be achieved through trade-offs between precision and size of the weights/activations, opening the spectrum of target platforms for these solutions, ranging from traditional architectures to full custom designs.

In the scope of hardware designs, despite the advantages described above, there is still a huge difficulty to implement CNNs on this type of devices by developers. Thus, because of this lack of solutions to assist the developers in this branch of deep learning that targets embedded solutions, the BNN-Pynq framework [21] was created. This engine supports the mapping of quantised convolution layers, pooling layers and dense layers in hardware specific solutions. It was built on top of the Vivado HLS, thus making use of its functions. Fig. 2.1 shows the possibles quantisations of the layers weights and activations in the Framework. The abbreviations next to the quantisations are used throughout the work to refer to those bit specifications.

Nomenclature	**Weight**	**Activation**
w1a1	1-bit	1-bit
w1a2	1-bit	2-bit
w2a1	2-bit	1-bit
w2a2	2-bit	2-bit

Table 2.1: Possible combinations of quantisations between the layer weights and activations in the BNN-Pynq Framework [21].

2.4.1 Data Flow

The configured logic circuit uses a tailored engine for each layer to meet its requirements, instead of an entire network processing structure with control mechanisms. These two CNN implementation philosophies contain quite a few differences. First of all, the pipelined system allows the entry of a new image as soon as the first engine finishes processing the previous image. On the other hand, a fixed system only allows one image to be processed at a time, obtaining lower throughput than the previous mentioned architecture. Another difference, resides in the number of processing units of the layers, being a suited number for each layer in the pipelined architecture and a generic number in the fixed architecture.

In the adopted approach, different layer processing units communicate through data streams. The convolution parameters have its parameters stored on the on-chip side of the device to reduce the latency in memory accesses. Figure 2.5 shows the interactions between the different parts of the device and intra-engine operations.

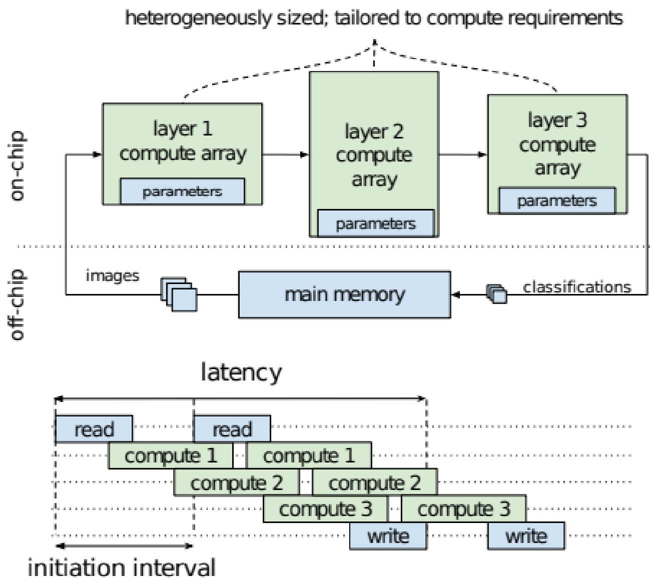

Figure 2.5: Heterogeneous streaming [21].

2.4.2 Layer-Specific Processing Engine

The previously spoken layer processing units are referred to as the Matrix-Vector-Threshold Unit(MVTU). Each one of them contains an input and output buffer, an array of processing elements(PEs) and its respective number of Simple Instruction Multiple Data(SIMDs). The developer defines the number of PE and SIMDs in function of the desired throughput. The PEs are made up of a datapath, all of which receive the same amount of data and flow control signals. The figure 2.6 exemplifies the operation of the MVTUs.

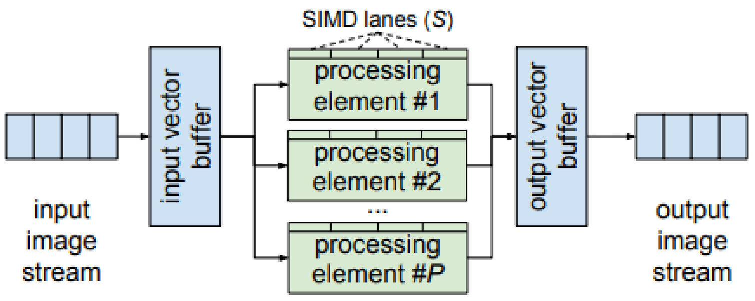

Figure 2.6: An array of PEs that constitute an engine doing the processing of SIMDs [21].

2.4.3 Methodology for Developing a Classifier

A set of actions need to be followed in order to generate a model of a classifier for the device that contains an FPGA, using the BNN-Pynq Framework. Figure 2.7 contains the work flow and the parts involved in this process. First, the convolution network is trained with the target bit quantisation in Theano [20]. Then, the ”finnthesizer” partitions the parameters according to the number of PEs and SIMDs defined. The architecture is then meticulously detailed through the HLS pragmas and thus producing the input to the synthesizer. In the end, the Xilinx Vivado tool uses the FINN hardware library and generates the bitfile for the heterogeneous device.

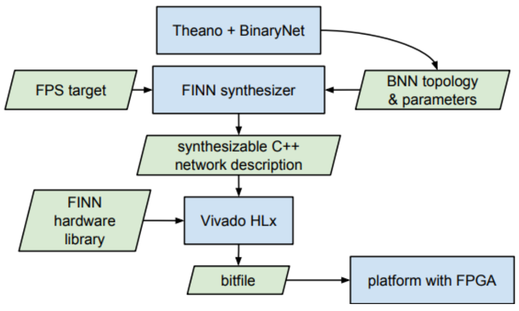

Figure 2.7: Generating a FPGA accelerator from a trained BNN [21].

2.5 Target Devices

Image processing has a strong dependence with the resources available in the target device. Therefore, in order to exploit the space of possible solutions in object classification, heterogeneous platforms are required. They combine different architectures with reconfigurable logic through fast communication lines, allowing to avoid bottlenecks in memory accesses. The great flexibility of heterogeneous devices in the mapping of applications is the key reason they are gaining great relevance in the market. For instance, Intel expects until 2020 to have 30% of FPGAs in its data centres [1].

Xilinx has products like the Zynq UltraScale + MPSoC and Zynq-7000 SoC families that have enough resources to overcome the restrictions imposed by CNNs algorithms. So, in this work, we are going to target the Pynq-Z2 and the Sundance VCS-1 boards as these devices are low power and heterogeneous. Although these boards have FPGAs that come from very close families, there is some divergence between its resources. A set of metrics served to compare the results obtained in each of the devices and to recognise which is the most versatile for embedded applications.

2.5.1 Pynq-Z2 Board

This device contains the heterogeneous ZYNQ XC7Z020-1CLG400C which is part of the Zynq-7020 SoC family. It has a dual-core Cortex-A9 processor with maximum 2Ghz of clock speed, 512MB of DDR3 memory and around 85K of logic cells. The communication between the on-chip and the off-chip modules is performed by 4 high performance AXI3 Slave ports. Figures 2.8 and 2.9 shows more details of the main components of the processing device and the device, respectively.

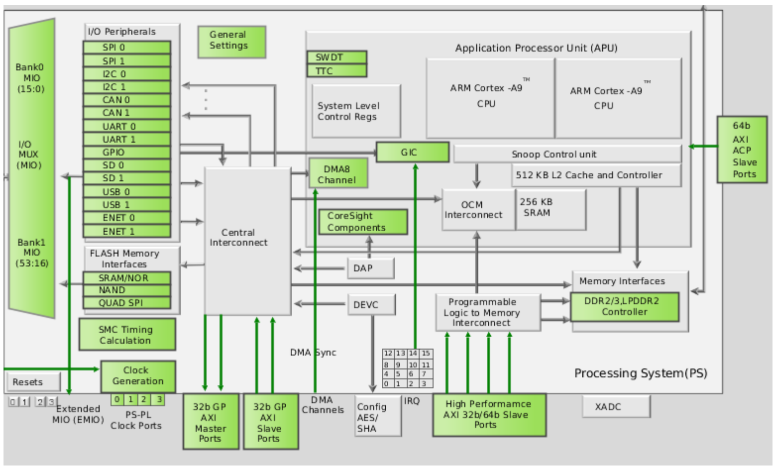

Figure 2.8: Main Components of the ZYNQ XC7Z020-1CLG400C Heterogeneous Device [24].

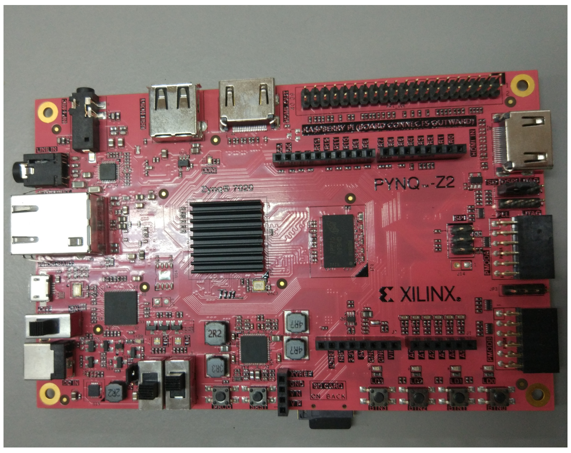

Figure 2.9: Pynq-Z2 Board.

2.5.2 Sundace VCS-1

The Sundance VCS-1 contains a Zynq UltraScale 4EV FPGA (zu4EV) with 192K logic cells, a quad-core Arm Cortex-A53 with a clock frequency up to 1,5Ghz, a dual-core Arm Cortex-R5 with a clock frequency up to 600 Mhz and a GPU Mali-400 with two cores. This range of different computational units allows applications to benefit from different performances and power consumptions depending on the resources in use. The power consumption ranges between 2-24 W, which enables to fine tune optimal implementations for embedded applications.

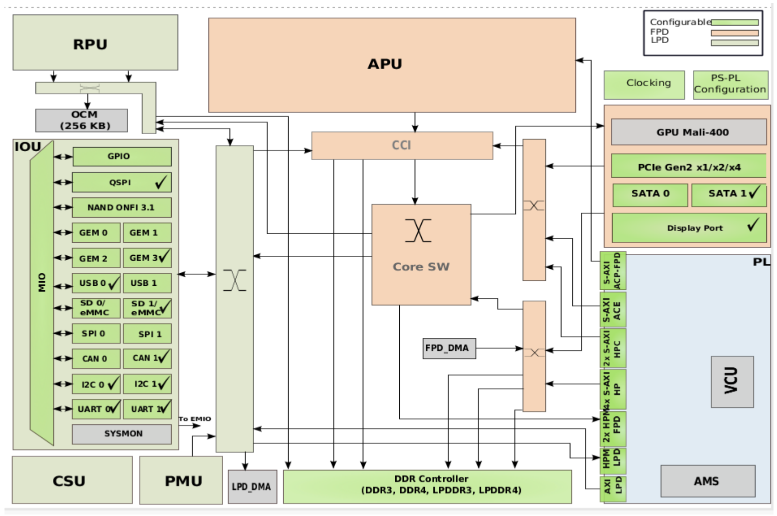

Figure 2.10: Main components of the Zynq heterogeneous device [24].

An external power consumption meter is connected between the heterogeneous processing device and the power supply for collecting consumption profiles. This tool provides useful data for the development of applications with low power requirements. In the figure 2.11, is shown the connections between the different devices and their interfaces with the development tools.

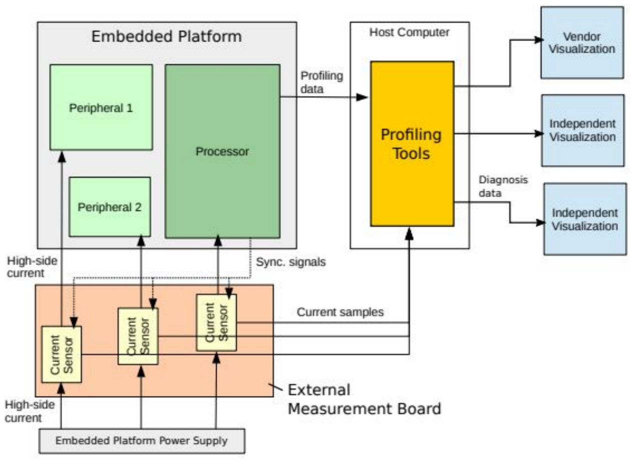

Figure 2.11: Physical connections and interfaces between devices[17].

In brief, the Zynq MPSoC is a deeply versatile device capable of achieve high-performance using specialised, customised and optimised combination of traditional methods. Figure 2.12 shows an image of the device.

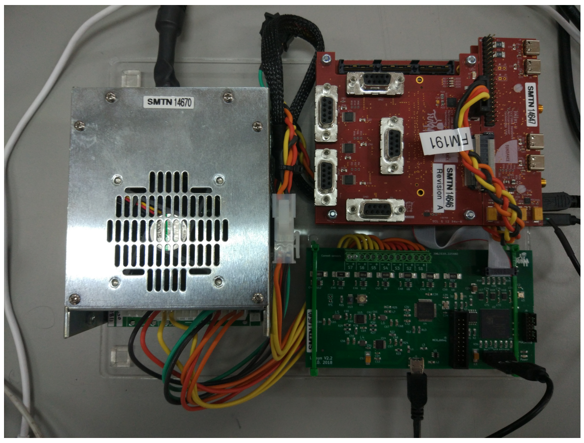

Figure 2.12: Sundance VCS-1 board.

2.6 Power Profiles

Previously it was seen that the Sundance VCS-1 allows to collect the power consumed by the processing units, but it remains unclear how the developer will access the profile logs. The Tulipp tool chain [9] will be used as the interface of the external consumer meter. Thus, this analysis tool facilitates to perceive each resource of the target device being used and to proceed with different mappings of an application in order to meet the low power requirements. Figure 2.13 shows the example of a consumption profile extracted in a computer vision application with multiple filters.

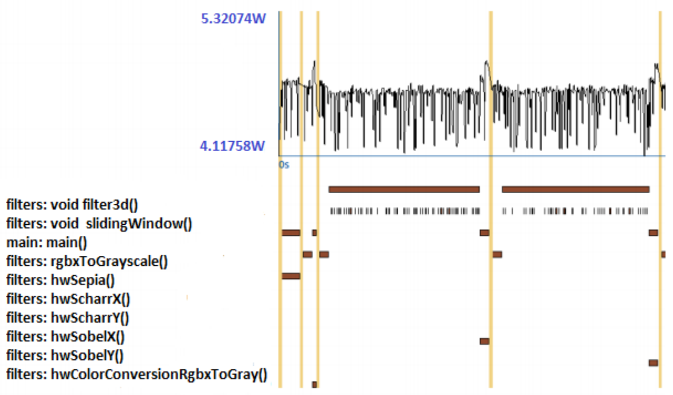

Figure 2.13: Consumption profile of an computer vision application.

3

Our Implementation of CNNs on Low Power Devices

In this chapter, we will cover the work developed starting on the definition of the testbed CNN, followed by the training of its multiple quantisations and, in the end, the synthesis of the designs to place on the target devices.

3.1 CNN Configuration

As a starting point, the layers that composed the test bed network used to benchmark the devices were defined. Table 3.1 shows the details of each layer of the network.

Type of Layer	Input size	Output Size	Number of Filters	Filter Size	Pad	Stride	Ignore Border
Convolution 1	32x32	30x30	32	3x3	0	1	True
Convolution 2	30x30	28x28	64	3x3	0	1	True
Maxpool 3	28x28	14x14	————	2x2	0	1	False
Convolution 5	14x14	12x12	128	3x3	0	1	True
Convolution 6	12x12	10x10	128	3x3	0	1	True
Maxpool 7	10x10	5x5	————	2x2	0	1	False
Convolution 8	5x5	3x3	256	3x3	0	1	True
Dense Layer 9	256 channels	512 channels	————	——-	–	—	———
Dense Layer 10	512 channels	512 channels	————	——-	–	—	———
Dense Layer 11	512 channels	10 channels	————	——-	–	—	———

Table 3.1: Compositions of the CNN selected as test bed.

The extraction of features is performed by the convolution layers on three different scales. The transition of scales is done through pooling layers. In the last scale, only one convolution layer is used due to the small resolution of the image at that time of the computation. The final three layers deal with the classification.

Figure 3.1 illustrates the operation of the system. The data flow between the features extraction and classification blocks remains unchanged in the full CPU and custom versions.

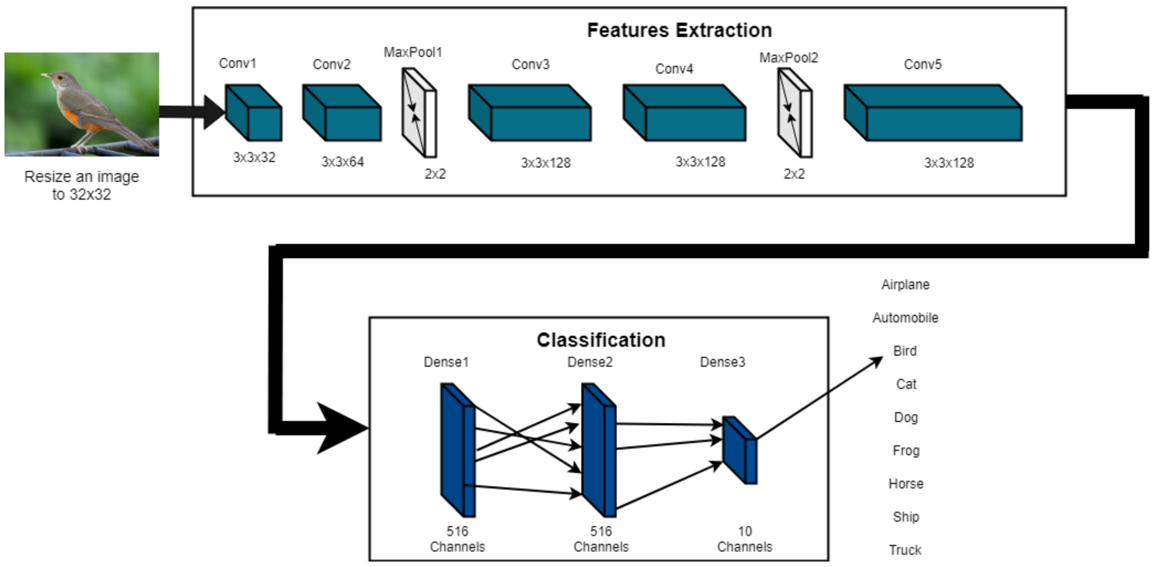

Figure 3.1: System high level architecture.

3.2 CNN Offline Training

In the past, it was proven the inefficacy on performing a normal training, and the subsequent quantisation of the weights and activations to reduce the size of the network. For that reason, a tailored training was performed with the specified bits for each case of the network quantisation. Thus, a more objective approach to achieve the final parameters was developed rather than randomly cut bits.

The created network was trained to detect between 10 classes using the CIFAR-10 dataset [11]. The training parameters were placed equal to those of the example in the framework [4], because this research is not focused on neural network training optimisations. In the table 3.2 is shown the details of the training developed for the different quantisations.

The trainings were executed during 500 epochs and each class used 5000 images. More specifically, 4500 images were used for training and the remaining 500 for precision mea-

Number of Epochs	500
Batch Size	50
Alpha	0.1
Epsilon	1e-4
LR Decay	$\left(\frac{1*10^{-7}}{1*10^{-3}}\right)^{\frac{1}{N\_Epochs}}$
Train Set Size	45000
Validation Set Size	5000
Loss Function	Squared Hinge Loss

Table 3.2: Target convolution neural network training parameters.

surement. Thus, all available images in the data set were used during the training. Figure 3.2 shows sample images of each class of the data set. The learning rate(LR) was decreased linearly at each epoch causing less updating on weights over time. The loss function used was the squared hinge loss, its details are available in [14].

Figure 3.2: Example images of the Cifar-10 classes.

3.3 Customising the Layer Processing Engines

The designated end system, represented in figure 3.3, is composed by the tailored layer processing engines integrated with several Xilinx IPs. The modules in blue represent the IPs of Xilinx, orange represents the processing unit of the layers and green the flow of data between the components. These IPs were created by the manufacturer to allow the custom made logic circuits to have access to the resources available in the target device. *BlackBoxJam_0* was created using the HLS hardware description language. In this block, the weights and activations of the CNN layers are initialised. Then, its respective calculation structures are created and connected by the previously defined order of layers in the network.

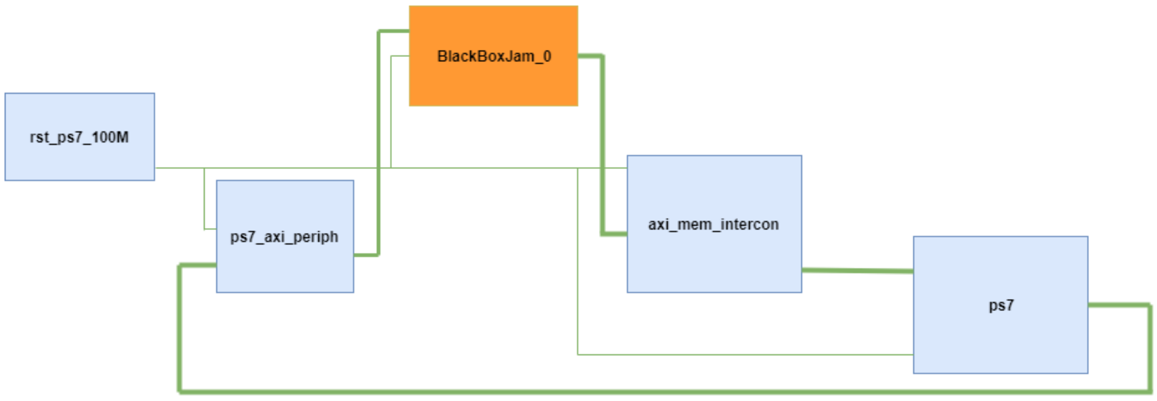

Figure 3.3: System high level architecture.

3.4 Port Mapping to zu4EV

The target processing devices have different resources, so for that reason, the I/O and memory mappings have divergences. In order to be able to synthesise the design, the platforms ports were switched with the information collected in its respective data sheets. Table 3.3 shows the pin changes made to pass the design from the Pynq-z2 to Sundance VCS-1.

Pin on Zynq-7020	Pin on zu4EV
G5	A15
F6	A14
A6	L13
C7	E15
A7	H13
B6	K14
G6	B15
C5	F13
B7	L14
B5	J14

Table 3.3: Pin mapping between the designs of the Zynq-7020 and the zu4EV.

4

Results

In this chapter, the results are presented. First, the errors in the object classification of the different network weight quantisations are exposed. Then the resources occupied in the target devices by the networks are explored. In the end, is showed the computation times, the throughput and power consumption of each network quantisation on the embedded platforms.

4.1 Training Error

The values in table 4.1 were obtained on processing different batches of samples from those used in the training operation. Therefore, it was intended not to influence the error function with biased validations. Network trainings with different quantisations were performed offline on a bench top computer with the graphical card NVIDIA geforce 750 TI.

	w1a1	**w1a2**	**w2a1**	**w2a2**
Test Error Rate	19.54 %	15.96 %	16.96 %	13.90 %

Table 4.1: Test error rate of the chosen CNN with Different weigth/activation quantisations. More information about the nomenclature used can be found in table 2.1.

4.2 Pynq-Z2 Resource Mappings

In this section, the resources occupied by each quantisation version in the Pynq-Z2 are evidenced. This low-cost platform features the Zynq-7020 heterogeneous device. The resources that composed this target device are programmable matrices, memory and communication buffers.

4.2.1 Network with 1-bit Weight and 1-bit Activation

Table 4.2 shows the convolution neural network synthesis report with quantisation of 1-bit weight and 1-bit activation in the Zynq-7020. The available design space was enough to accommodate the digital circuit. Block Ram Tiles and LUTS are the resources with the highest occupancy rate.

Site Type	**Used**	**Available**	**Utilisation %**
Slice LUTs	27267	53200	51.25
LUT as Logic	22706	53200	42.68
LUT as Memory	4561	17400	26.21
Slice Registers	40118	106400	37.70
Registers as Flip Flop	40118	106400	37.70
Registers as Latch	0	106400	0.0
F7 Muxes	868	26600	3.26
F8 Muxes	69	13300	0.52
Block RAM Tile	105.5	140	75.36
RAMB36/FIFO	76	140	54.29
RAMB18	59	280	21.05
DSPs	24	220	10.91

Table 4.2: Resources used by the quantised neural network with 1-bits weight and 1-bits activation in the Pynq-Z2 device.

4.2.2 Network with 1-bit Weight and 2-bit Activation

Table 4.3 shows the convolution neural network synthesis report with quantisation of 1-bit weight and 2-bit activation in the Zynq-7020. The available design space was enough to accommodate the digital circuit. Again, Block Ram Tiles and LUTS are the resources with the highest occupancy rate.

Site Type	Used	Available	Utilisation %
Slice LUTs	39565	53200	74.37
LUT as Logic	32514	53200	61.12
LUT as Memory	7051	17400	40.52
Slice Registers	56990	106400	53.56
Registers as Flip Flop	56990	106400	53.56
Registers as Latch	0	106400	0.0
F7 Muxes	1709	26600	6.42
F8 Muxes	138	13300	1.04
Block RAM Tile	110.5	140	78.93
RAMB36/FIFO	86	140	61.43
RAMB18	49	280	17.50
DSPs	26	220	11.82

Table 4.3: Resources used by the quantised neural network with 1-bits weight and 2-bits activation in the Pynq-Z2 device.

4.2.3 Network with 2-bit Weight and 1-bit Activation

Table 4.4 shows the convolution neural network synthesis report with quantisation of 2-bit weight and 1-bit activation in the Zynq-7020. The available design space was enough to accommodate the digital circuit. Block Ram Tiles and LUTS are the resources with the highest occupancy rate, as expected.

4.2.4 Network with 2-bit Weight and 2-bit Activation

Table 4.5 shows the convolution neural network synthesis report with quantisation of 2-bit weight and 2-bit activation in the Zynq-7020. The available design space was enough to accommodate the digital circuit. Block Ram Tiles, RAMB36/FIFO and LUTS are the resources with the highest occupancy rate. In particular, the RAMB36/FIFO was fully utilised by this network quantisation.

Site Type	**Used**	**Available**	**Utilisation %**
Slice LUTs	35681	53200	67.07
LUT as Logic	30743	53200	57.79
LUT as Memory	4938	17400	28.38
Slice Registers	47805	106400	44.93
Registers as Flip Flop	47805	106400	44.93
Registers as Latch	0	106400	0.0
F7 Muxes	1709	26600	6.42
F8 Muxes	138	13300	1.04
Block RAM Tile	107.5	140	76.79
RAMB36/FIFO	86	140	61.43
RAMB18	43	280	15.36
DSPs	26	220	11.82

Table 4.4: Resources used by the quantised neural network with 2-bits weight and 1-bits activation in the Pynq-Z2 device.

Site Type	**Used**	**Available**	**Utilisation %**
Slice LUTs	37890	53200	71.22
LUT as Logic	27442	53200	51.58
LUT as Memory	10448	17400	60.05
Slice Registers	48958	106400	46.01
Registers as Flip Flop	48958	106400	46.01
Registers as Latch	0	106400	0.0
F7 Muxes	4474	26600	16.82
F8 Muxes	1676	13300	12.60
Block RAM Tile	140	140	100.00
RAMB36/FIFO	140	140	100.00
RAMB18	0	280	0.0
DSPs	32	220	14.55

Table 4.5: Resources used by the quantised neural network with 2-bits weight and 2-bits activation in the Pynq-Z2 device.

4.3 Sundance VCS-1 Resource Mappings

This platform is composed by the heterogeneous processing device Zynq UltraScale+ZU4EV-1E. It is important to notice that its resources are different from those presented in 4.2.

4.3.1 Network with 1-bit Weight and 1-bit Activation

Table 4.6 shows the convolution neural network synthesis report with quantisation of 1-bit weight and 1-bit activation in the ZU4EV-1E. The available design space was enough to accommodate the digital circuit. In terms of design mappings, the reprogrammable logic units were occupied in less than half, and the most used resources were the Block RAM tiles and the RAMB36/FIFO.

Site Type	**Used**	**Available**	**Utilisation %**
CLB LUTs	25975	87840	29.57
LUT as Logic	23347	87840	26.58
LUT as Memory	2628	57600	4.56
CLB Registers	39257	175680	22.40
Registers as Flip Flop	39357	175680	22.40
Registers as Latch	0	175680	0.0
CARRY8	1331	14640	9.09
F7 Muxes	822	58560	1.40
F8 Muxes	240	29280	0.82
F9 Muxes	0	14640	0.0
Block RAM TILE	105.5	128	82.42
RAMB36/FIFO	76	128	59.38
RAMB18	59	256	23.05
DSPs	24	728	3.30

Table 4.6: Resources used by the quantised neural network with 1-bits weight and 1-bits activation in the zu4EV device.

4.3.2 Network with 1-bit Weight and 2-bit Activation

Table 4.7 shows the convolution neural network synthesis report with quantisation of 1-bit weight and 2-bit activation in the ZU4EV-1E. The available design space was enough to accommodate the digital circuit, and the resources were filled at a similar rate as the previous one.

Site Type	Used	Available	Utilisation %
CLB LUTs	38321	87840	43.63
LUT as Logic	34797	87840	39.61
LUT as Memory	3524	57600	6.12
CLB Registers	55138	175680	31.39
Registers as Flip Flop	55138	175680	31.39
Registers as Latch	0	175680	0.0
CARRY8	1381	14640	9.43
F7 Muxes	1611	58560	2.75
F8 Muxes	480	29280	1.64
F9 Muxes	0	14640	0.0
Block RAM TILE	110.5	128	86.33
RAMB36/FIFO	86	128	67.19
RAMB18	49	256	19.14
DSPs	26	728	3.57

Table 4.7: Resources used by the quantised neural network with 1-bits weight and 2-bits activation in the zu4EV device.

4.3.3 Network with 2-bit Weight and 1-bit Activation

Table 4.8 shows the convolution neural network synthesis report with quantisation of 1-bit weight and 1-bit activation in the zu4EV-1E. The available design space was enough to accommodate the digital circuit.

Site Type	Used	Available	Utilisation %
CLB LUTs	35263	87840	40.14
LUT as Logic	32183	87840	36.64
LUT as Memory	3080	57600	5.35
CLB Registers	44655	175680	25.42
Registers as Flip Flop	44655	175680	25.42
Registers as Latch	0	175680	0.0
CARRY8	1313	14640	8.97
F7 Muxes	1614	58560	2.76
F8 Muxes	480	29280	1.64
F9 Muxes	0	14640	0.0
Block RAM TILE	107.5	128	83.98
RAMB36/FIFO	86	128	67.19
RAMB18	43	256	16.80
DSPs	26	728	3.57

Table 4.8: Resources used by the quantised neural network with 1-bits weight and 2-bits activation in the zu4EV device.

4.3.4 Network with 2-bit Weight and 2-bit Activation

This version presented errors during the synthesis of the digital circuit. More specifically, the project requires more RAMB36/FIFO than the existing ones. In other words, there is not enough UltraRam memory in the target device to place the project. We came across by a very common design constraint in the development of embedded systems, that is the device does not has enough memory to accommodate the project.

4.4 Processing Benchmarks

In the scope of the performance, the metrics used for comparison were the number of frames per second(FPS) computed. Tables 4.9 and 4.10 evidence the computation times of the different network quantisations, in the CPU and recofigurable logic of the device Zynq-7020, respectively. Table 4.11 shows the processing times of the quantised networks in the CPU of the zu4EV.

	w1a1	**w1a2**	**w2a1**	**w2a2**
Inference per Image	1172985 us	4257559 us	4266916 us	7221958 us
Classification Rate	0.85 FPS	0.23 FPS	0.23 FPS	0.13 FPS

Table 4.9: Computation Times of the CPU in the Zynq-7020.

	w1a1	**w1a2**	**w2a1**	**w2a2**
Inference per Image	3934 us	3981 us	3980 us	5837 us
Classification Rate	254.19 FPS	251.19 FPS	251.26 FPS	171.32 FPS

Table 4.10: Computation times of the customised circuit in the Zynq-7020.

	w1a1	**w1a2**	**w2a1**	**w2a2**
Inference per Image	298278 us	2004327 us	1981726 us	—
Classification Rate	3.35 FPS	0.50 FPS	0.50 FPS	—

Table 4.11: Computation times of the CPU in the zu4EV.

An attempt was made to place the generated bitstreams of the network quantisations on the reconfigurable logic of the zu4EV , but there were problems with a driver. The mismatch was generated due to incompatibilities in the image kernels. A custom boot image for the device would be necessary to allow run the implementations.

4.5 System Latency

Table 4.12 shows the number of operations of the different layers and its latency in each quantisation on the Pynq-Z2.

	Latency W=1 and A=1	**Latency W=1 and A=2**	**Latency W=2 and A=1**	**Latency W=2 and A=2**	**Nº Operations**
Convolution 1	16200 us	16200 us	16200 us	32400 us	1555200
Convolution 2	14112 us	14112 us	14112 us	56448 us	28901376
Convolution 3	20736 us	20736 us	20736 us	82944 us	21233664
Convolution 4	28800 us	28800 us	28800 us	115200 us	29491200
Convolution 5	20736 us	20736 us	20736 us	82944 us	5308416
Dense Layer 1	294912 us	294912 us	294192 us	294912 us	2359296
Dense Layer 2	32769 us	32758 us	32768 us	32769 us	524288
Dense Layer 3	8192 us	8192 us	8192 us	8192 us	65536

Table 4.12: Throughput and number of Operations of the different weight/activation configurations of the CNN in the target devices.

4.6 Power Consumption

While the VCS-1 board has an external consumption meter, it only worked in sync with custom hardware targeting the specific systems. Since we tested and ported the design across platforms, we had to use an estimator to get power profile logs of the two target devices for comparison. Otherwise, we would not have intel about the power consumption of the Pynq-Z2 and the VCS-1. We used an official tool, the Xilinx Power Estimator (XPE), to get approximated values about the power consumption. Figure 4.1 shows the consumption obtained for each network quantisation in the diferent boards.

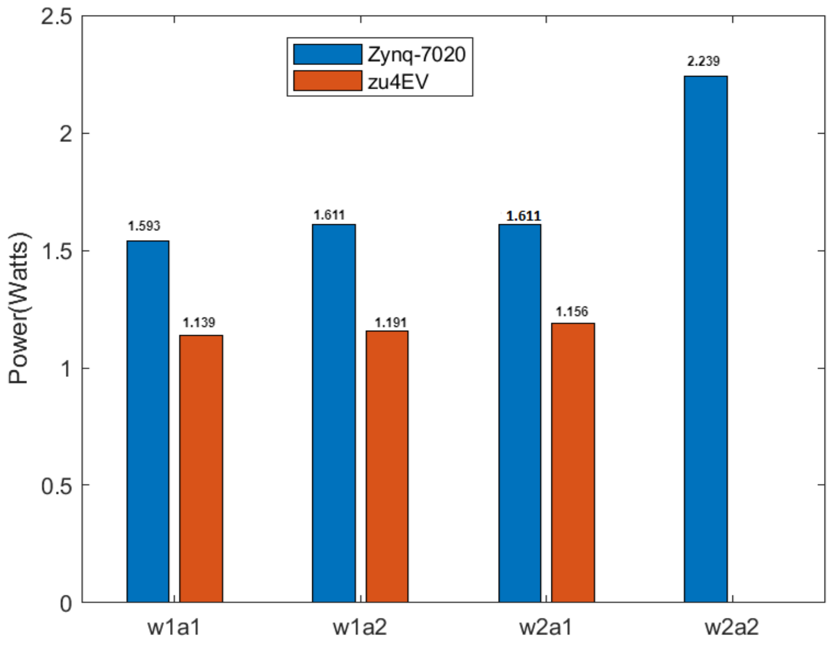

Figure 4.1: Estimated power consumption of the System-on-Chip(SoC) in the target devices.

5

Analysis and Discussion

In this chapter, we explore the results presented in several ways, since metric calculations to trade-off graphs.

Observing the errors of the different quantisations in table 4.1, we verified that there is a dependence between the number of bits and the errors in the classification. The **w1a2** configuration presents less error than the **w2a1**, but we can't infer that the precision in the activations influences more the classification than the layers since the validation test is limited.

Table 5.1 presents the speedup between the performances recorded in tables 4.9 and 4.10. The speedup is a quite significant being of greater magnitude in the configurations **w1a2**, **w2a1** and **w2a2**.

	w1a1	**w2a1**	**w1a2**	**w2a2**
SpeedUp	299	1092.13	1092.23	1317.85

Table 5.1: Speedup between hardware and software computation of the Zynq-7020.

Regarding the CPUs of the different devices, we can observe better computation performance on the Arm Cortex-A5 of the zu4EV. The processor with better results was released a few years after the beginning of the tablets and smartphones era, while the other has come out at the beginning. Thus, on that processor, a higher clock frequency and lower L2 cache access latency can be achieved, influencing the final performance on our test bed CNN.

Table 4.12 contains the number of operations and latency of each layer at the different quantisations. The number of operations of all quantised networks does not diverge, something expected since the layers are the same. Regarding latency, it is the same in all dense layers. In the convolution layers, latency is the same for **w1a1**, **w2a1** and **w1a2** configurations. However, for the heavyweight version, the latency is higher than in the others due to the existence of bottlenecks in accessing that amount of data. It is also possible to decrease latency by optimising the number of processing engines of each layer. Thus, a higher throughput can be achieved.

Data about the power consumption of the SoC was obtained using a Xilinx official estimator, and its values are showed in figure 4.1. Estimators can work to get fast insight during prototyping, but they are limited in terms of precision. In this case, the tool used was developed by the manufacturer, which gives more confidence in the inferred data. About the Pynq-Z2, the heavyweight version had the highest power consumption. Further conclusions about the other quantisations were not possible since the values of its logs are very similar. The results of the quantisations in VCS-1 are inconclusive due to the same fact. However, we could observe higher power consumption in the Pynq-Z2 compared to the last mentioned device. The tool only estimated the power consumed for the SoC of the device, so it is expected that the complete system has higher values.

In the analysis of the resources tables, we were able to draw some conclusions. First of all, that the most used resourses by the networks were the LUTs and the RAMB36/FIFO. Particularly, in the **w2a2** configuration the RAMB36/FIFO was pushed to the limit with a full occupation. It is also important to note that the **w1a2** version occupies more memory LUTs than the others. Secondly, there are no significant variations between the remaining resources in different quantisations.

In figure 5.1, we observe that the increase in the number of bits of the weights and activations is associated with better classification accuracy. Consequently, the digital circuit uses more LUTs. This behaviour in the project synthesis was expected a priori. However, version **w1a2** uses more device resources than **w2a2**. We believe this happened due to the ineffective partitioning of the instructions into the layers processing engines(PEs). In this case, we created some redundant processing units, causing the computation of a fewer number of instructions for each one of them.

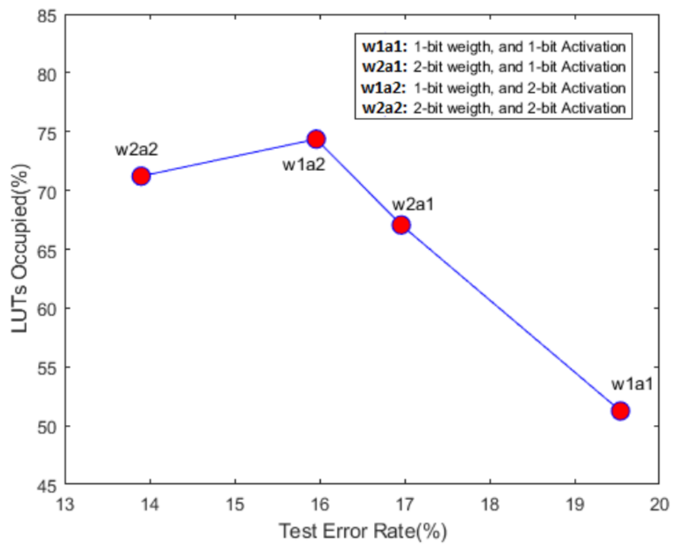

Figure 5.1: Test error rate of the different quantisations with its respective LUT ocupation on the Zynq-7020.

Table 5.2 shows the performance of the quantised networks as a function of FPGA LUTs occupancy running on the CPU. There was a performance degradation of around 60% between version **w1a1** and the others. However, the best version does not even achieve 1 FPS. Therefore, the obtained processing time in this board is not suitable for real-time applications.

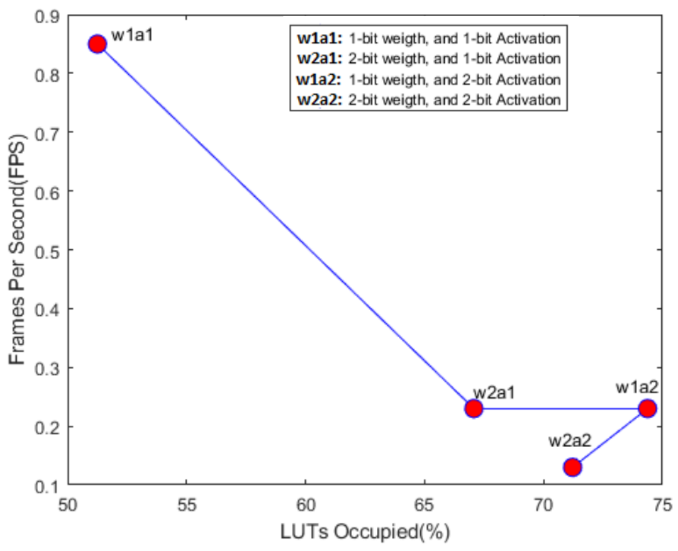

Figure 5.2: LUT occupation with its respective performance Off-Chip on the Zynq-7020.

Table 5.3 shows similar performances between the networks with the quantisations **w1a1**, **w2a1** and **w1a2**. Regarding **w2a2**, there were performance losses of nearly 100 FPS. The occupation of LUTs had already been seen previously, but the table was designed to show the trade-off.

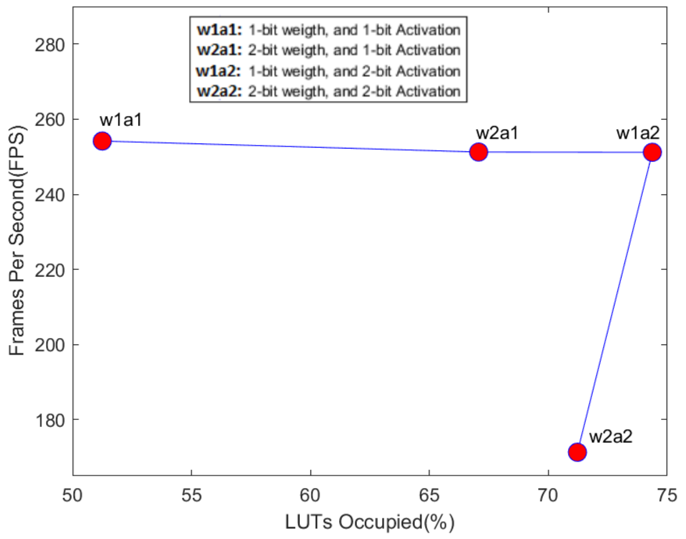

Figure 5.3: LUT occupation with its respective performance On-Chip on the Zynq-7020.

In 5.4, we find that the quantisation **w1a1** has less processing time, but consequently more classification error. The **w2a2** configuration has less error, although worse performance. The quantisated networks **w1a2** and **w2a1** have similar performances and similar classification errors. It is corroborated that the CPU version has very low computation performances.

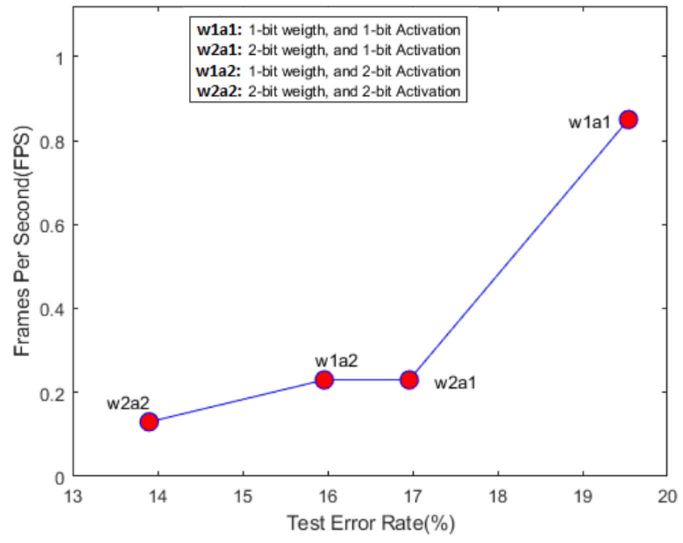

Figure 5.4: Test error rate of the different quantisations with its performance respective Off-Chip on the Zynq-7020.

Observing 5.5, it is possible to verify that the networks **w2a1** and **w1a2** have similar validation errors and computation times. The quantisation **w1a1** presents a computation performance similar to **w2a1** and **w1a2**, but the error is more substantial. Finally, **w2a2** presents the lowest validation error, yet it has the lowest processing time of all quantisations.

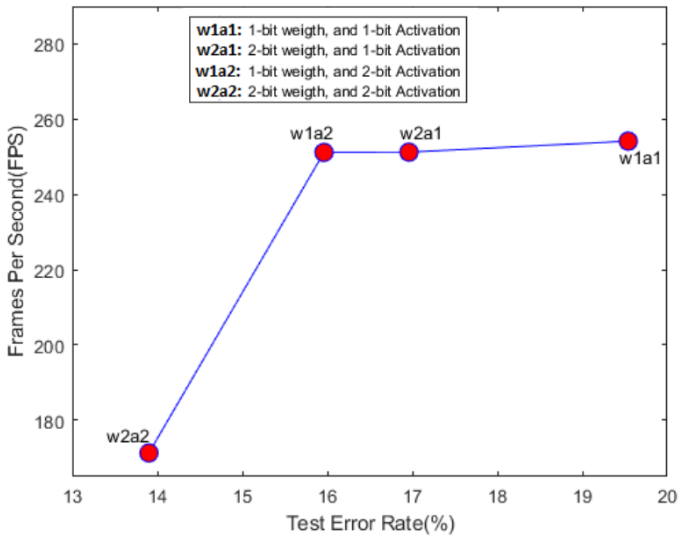

Figure 5.5: Test error rate of the different quantisations with its respective performance On-Chip on the Zynq-7020.

The target devices present a quite number of differences, despite being from very close families. First, the basic logical units are different, in the Pynq-Z2 are the Lookup tables(LUTs) and in the Zu4EV are the Configurable Logic Blocks (CLBs). The last mentioned basic unit is composed of LUTs, Flip-Flops and Cascadable adders. Its number is also different, with zu4EV having more design resources. Second, the number of RAMB36/FIFO did not allow the synthesis of the quantisation **w2a2** in the zu4EV. For the Pynq-Z2 device, it was possible to perform the design synthesis. However, the resource that the w2a2 network could not fit in the zu4EV was pushed to the limit in the Pynq-z2. Also, the computation times of its respective CPUs were slightly different. Because of this differences, the previous analysis made for the zynq-7020 was done too for the zu4EV.

Table 5.6 shows a lower percentage of resource utilisation compared to the previous analysed device for the same values of precision. The **w1a1**, the most lightweight version, had a degradation in the error of 5 per cent compared to the best version in this device. The quantisations in **w2a1** and **w1a2** present similar resource occupancy and classification errors.

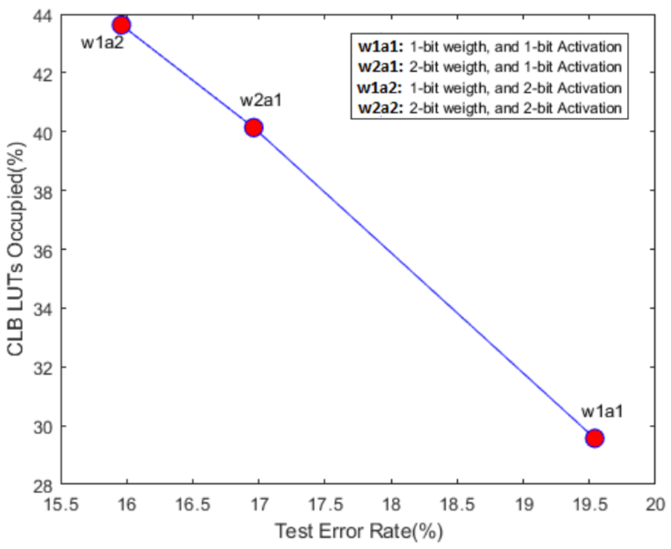

Figure 5.6: Test error rate of the different quantisations with its respective CLB LUT ocupation on the zu4EV..

The relation between the error of the quantisations and its computational performance in the zu4EV can be observed in table 5.7. Better processing times were obtained on this device, and for instance, in the version w1a1, it has exceeded 3 FPS. The versions with the symmetric quantisations continued to show equal computation times, despite being on a different CPU.

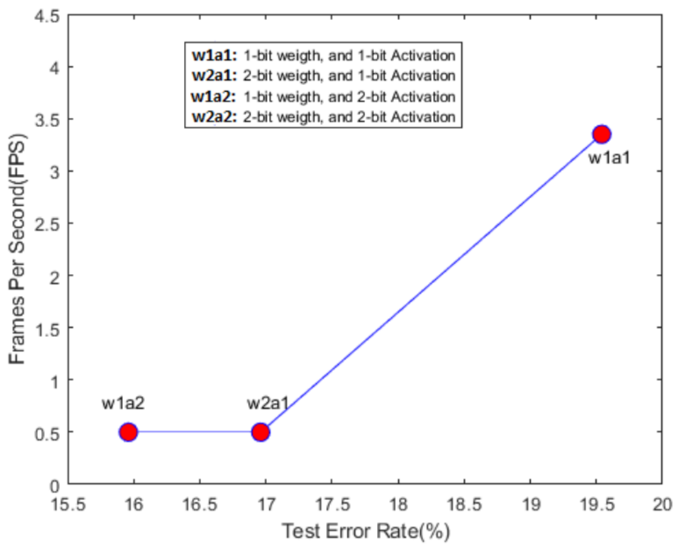

Figure 5.7: LUT occupation with its respective performance Off-Chip on the zu4EV.

As for table 5.8, we verified that there is dependence between the occupancy of the target device and the computation time. Therefore, the quantisation in w1a1 shows the best trade-off between size and performance. Although, it loses on classification accuracy.

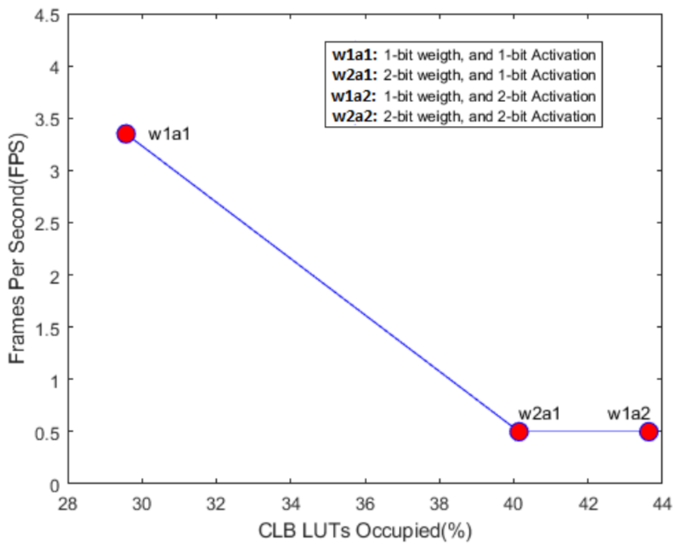

Figure 5.8: CLB LUTs occupation with its respective performance Off-Chip on the zu4EV.

6

Conclusions and Future Work

In the early stages of our work, for different bit quantisations, it was performed the training of the network. When analysing the values in the test error, we considered that they were quite low for the number of bits used. In terms of error comparison among the different quantisations, there is no abysmal distinction between the lightweight and heavier versions. The versions **w2a1** and **w1a2** present similar errors, invalidating any possible conclusion regarding the part of the layer that has more influence on the accuracy of the network. We believe that these precision values were inflated by the easy identification of the classes in the images used for validation test, and in a real situation, the errors would be greater than the obtained ones. The false assumption about the veracity of the error obtained in the evaluation is commonly made in machine learning.

The resources analysis of the different devices proved the existence of divergences in the families of products. The disparity in resources ranges from the basic logical unit to the types of CPUs. Zu4EV has around 30000 more programmable arrays than the Zynq-7020. However, it has fewer RAMB36/FIFOs, a critical resource in the design of the test bed network that was fulfilled on Zynq-7020 and on zu4EV was not satisfied in the heavier version of the quantisations. Project planning may be essential in order to choose the device with the resources that best suit each application. In particular, for designs where a lot of processing engines are needed, the zu4EV is more suitable. Thus, more reprogrammable matrices are available. In contrast, for applications where there is a lot of dependence on the input data, Pynq-Z2 is better suited because it has more UltraRam, manifesting in less latency in memory accesses. Regarding embedded solutions, Sundance VCS-1 is a complete development tool due to the ability to collect consumption profiles that allows to fine tune each resource. In the Pynq-Z2 it can only be estimated with underlying errors. Above all, device prices are considerably different, about a few hundred for Pynq-Z2 and a few thousand for Sundance VCS-1. Thus, prototyping planning can save a few thousand euros, depending on the final user application.

Despite the successful synthesis of some bitstreams for the zu4EV, driver incompatibilities have been encountered. Cause of that, the processing of the test bed CNN on the System-on-Chip of the board was disabled. The Xilinx IPs continue to have hidden features that make it difficult to readjust designs between different devices. Therefore, we came across a barrier between prototyping and the final solution. This particular problem causes delays and adds costs to projects.

The processing times of the custom circuit were colossally different from those of the CPU, in the Pynq-Z2. Nevertheless, there is no implication that all solutions have to be made in hardware due to the greater time-to-market and the limitation of persons able to develop this type of projects. The relationship between the performances of the different quantisations was very similar in the different application mappings. Analysing the several metrics used, the quantised version in **w2a1** was the one that found the greatest balance between performance, error and resource utilisation in the Pynq-Z2. In the particular case of the Sundance VCS-1, **w1a1** achieved a better equilibrium between the metrics in scope.

In the end, it has been shown, that with simple quantised convolution neural networks, levels of accuracy suitable for applications that rely on object classification can be obtained. As for heterogeneous devices, great flexibility in the mappings of applications allowed us to target embedded systems. Although energy consumption was not thoroughly explored, we have successfully addressed the trade-offs between accuracy, resources and processing performance.

The work carried out gave origin to conference submissions. In particular, one short paper on the proposed work and one live demo were accepted. Further details about the articles can be consulted in the attachments of this work.

Embedded solutions can be deeper explored starting with the results from this work. The first logical step to take is to change the necessary driver to run the generated bitstreams in the zu4EV SoC. Once this is complete, it is possible to collect the real consumption profiles of the test bed convolution neural networks. Thus, allowing to study new compromises, for example between the number of operations of the layers and the energy consumption, among others.

Lower latency, higher throughput, or less target device occupancy, can be obtained through the optimisation of the number of customised layer processing engines. The customisation of the engines is a mean to reach new values in the metrics used.

It also would be important to place the classifier in a real application, for example, in a camera on top of a mobile robot, to test it with different images of those used on the test set. Thus, the reliability of the classifier would be verified for its integration in real applications.

In our work, we only trained the different network quantisations to classify between 10 classes. Nevertheless, it would be interesting to train the network to detect other objects, so the applications of the classifier are not so limited.

Bibliography

[1] Jeff Burt. Intel to start shipping xeons with fpgas in early 2016. *eWEEK*, 2015.

[2] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. *CoRR*, abs/1807.05520, 2018.

[3] Jason Cong and Bingjun Xiao. Minimizing computation in convolutional neural networks. In *ICANN*, 2014.

[4] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[5] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola, and Vladimir Vapnik. Support vector regression machines. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 155–161. MIT Press, 1997.

[6] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir D. Bourdev. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014.

[7] Hamed Habibi Aghdam and Elnaz Jahani Heravi. *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. 01 2017.

[8] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.

[9] Tobias Kalb, Lester Kalms, Diana Göhringer, Carlota Pons, Fabien Marty, Ananya Muddukrishna, Magnus Jahre, Per Gunnar Kjeldsberg, Boitumelo Ruf, Tobias Schuchert, et al. Tulipp: Towards ubiquitous low-power image processing platforms. In *Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2016 International Conference on*, pages 306–311. IEEE, 2016.

[10] D. Kang, D. Kang, J. Kang, S. Yoo, and S. Ha. Joint optimization of speed, accuracy, and energy for embedded image recognition systems. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 715–720, March 2018.

[11] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989.

[14] Ching-Pei Lee and Chih-Jen Lin. A study on l2-loss (squared hinge-loss) multiclass svm. *Neural computation*, 25, 03 2013.

[15] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2181–2191. Curran Associates, Inc., 2017.

[16] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 185–201, Cham, 2018. Springer International Publishing.

[17] Ananya Muddukrishna, Asbjørn Djupdal, and Magnus Jahre. Power profiling of embedded vision applications in the tulipp project.

[18] Hiroki Nakahara, Haruyoshi Yonekawa, Tomoya Fujii, and Shimpei Sato. A lightweight YOLOv2: A binarized cnn with a parallel support vector regression for an FPGA. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, pages 31–40, New York, NY, USA, 2018. ACM.

[19] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016.

[20] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[21] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Heng Wai Leong, Magnus Jahre, and Kees A. Vissers. FINN: A framework for fast, scalable binarized neural network inference. *CoRR*, abs/1612.07119, 2016.

[22] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. *CoRR*, abs/1512.06473, 2015.

[23] W. Xie, C. Zhang, Y. Zhang, C. Hu, H. Jiang, and Z. Wang. An energy-efficient fpga-based embedded system for cnn application. In *2018 IEEE International Conference on Electron Devices and Solid State Circuits (EDSSC)*, pages 1–2, June 2018.

[24] Inc. Xilinx. Device manufacturer.

[25] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, and Jason Cong. Energy-efficient CNN implementation on a deeply pipelined FPGA cluster. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ISLPED '16, pages 326–331, New York, NY, USA, 2016. ACM.

[26] G. Zhou, A. Cichocki, Q. Zhao, and S. Xie. Efficient nonnegative tucker decompositions: Algorithms and uniqueness. *IEEE Transactions on Image Processing*, 24(12):4990–5003, Dec 2015.

[27] Xiaotian Zhu, Wengang Zhou, and Houqiang Li. Adaptive layerwise quantization for deep neural network compression. *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2018.

Appendix

1. Short Paper on the Proposed Work at 15th Portuguese Meeting on Reconfigurable Systems;
2. Demo at exp.at’19;

1 YOLOv3 object detector: exploring energy efficient implementations on Reconfigurable Logic

Bruno Oliveira, ISR-UC and Jorge Lobo, ISR-UC

1.1 Abstract

Object detection and classification is a problem with great relevance in computer vision, since it enables a wide range of target applications, such as robot navigation and security. At present, there are a set of solutions that solve this problem, the most successful relying on neural networks. Among the best known are GoogleNet, AlexNet and YOLO. However, the underlying processing requires high performing computational platforms such as GPUs, CPU clusters, or custom ASICs. Apart from ASICs, that can be lower power but need to be custom made with a high cost, they are typically high power and not well suited for embedded systems. However there has been some progress in low power approaches, driven in part by the smart phone and tablet market, and heterogeneous platforms are now available that explore a mix of architectures (CPUs and GPUs) and reconfigurable logic. In this work we propose an implementation of a state-of-the-art object detector, Yolov3, on a hybrid platform for use in a wide range of applications, thoroughly exploring the design space targeting power efficiency and detection performance. The underlying algorithm is analysed, and key components for concurrent and parallel computation identified. Mappings of this to the heterogeneous platform will be explored, ranging from a baseline CPU implementation to a full custom implementation maximising the use of the available resources. A set of metrics is considered for the evaluation of the different configurations.

1.2 Introduction

Embedded computer vision applications have been somewhat limited due to low power and performance constraints. The current state-of-the art in image processing algorithms, namely in object detection and classification, has impressive results, but rely on substantial computation resources.

Pushed by the mobile device consumer market, there is now a wider spectrum of low power solutions of CPU+GPUs, that combined with FPGAs enable low power heterogeneous platforms. On these platforms, design space exploration can be done to optimise the algorithms efficiency.

In this work, we will pursue an implementation of YOLOv3 [6] on a Multi-Processor System on Chip (MPSoC)+Zynq UltraScale, shown in figure 1.

YOLO (You Only Look Once) is a open source state-of-the-art object detector and it has been used in numerous applications, ranging from academic to commercial. Its core is composed by a Convolutional Neural Network (CNN), which simultaneously predicts boxes with possible locations of objects and performs the attribution of class probabilities for those boxes in a single image passage. However, to obtain a good precision with a small computation time, the network needs to have an enormous amount of layers, requiring a lot of memory to store the weights and logic for the calculations. A network compression needs to be made to fit it in the target platform. Fortunately, techniques like prunning[7], quantization[8], binarization [4] or a combination of a few of them are presenting good results in reducing its size, while maintaining acceptable losses of accuracy.

In the past, a lightweight version of YOLOv2 [5] was implemented on a similar target platform. They proved it was feasible, but limited, since due to compromises, the reduced resulting accuracy might not suit some critical applications.

Another attempt to apply the YOLO in embedded systems was done by [3] using a NVIDIA Jetson-Tx2. However, the version implemented is already outdated, because of the limited number of classes compared to the latest version of YOLO.

1.3 Proposed System

Figure 1 shows an overview of the proposed system.

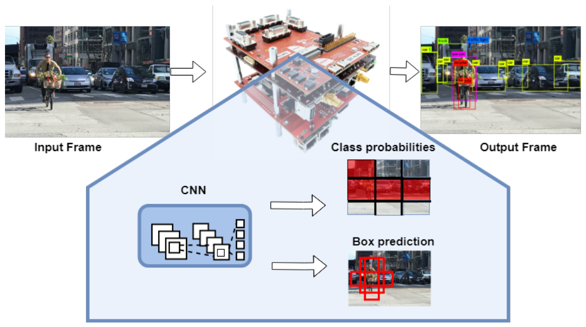

Figure 1: Overview of the proposed system.

An input frame is processed in the neural network, in order to produce a tensor with the coordinates of multiple boxes and the respective probability of containing an object. Labels are assigned to predictions with the highest degree of confidence. The CPU will be responsible for delivering the image data through shared memory, so that it can be processed by the custom circuit developed.

1.4 YOLOv3 Algorithm

The YOLOv3 [6] consists of a total of 106 layers, such as convolution layers, shortcuts, upsamples, route layers and yolo layer. However, [1] has evidence that convolution operations occupy over 90% of total system computation. This lead us to focus attention on accelerating convolution layer operations. The value of each output pixel is obtained by the following formula:

$$O_n(x,y) = \sum_{c=0}^{C-1} \sum_{h=-\frac{H-1}{2}}^{\frac{H-1}{2}} \sum_{w=-\frac{W-1}{2}}^{\frac{W-1}{2}} I(x+w, y+h, c) K_n(w,h) \tag{1}$$

Where O_n(x,y) is the output pixel obtained by the convolution of the nth filter of the layer, K_n, with the input image I. The number of channels in the input is represented by C and WxH are the filter dimensions.

The convolution of the layer filters with the input image can be parallelised given the absence of data dependence between the operations, as show in equation 1. This parallelisation and further optimisations will be further explored on the target hardware. As a baseline for comparison we already ran tests using the code provided by the YOLOv3 authors [6] on our hardware with CPU and CPU+GPU. Table 1 shows the computation time, but we also need to collect energy consumption data.

YOLO	v3	v3 Tiny
Intel Core I7-4770 CPU@ 3.40Ghz×8	7.3556s	0.7752s
Intel Core I7-4770 CPU@ 3.40Ghz×8 + OpenMP	2.2579s	0.2757s
Intel Core I7-4770 CPU@ 3.40Ghz×8 + GeForce GTX 750 TI	0.2270s	0.1204s

Table 1: Processing times of a 768×576 frame.

1.5 Hardware Overview

Image processing has a strong dependence with the resources available in the target hardware. Therefore, in order to explore the space of possible solutions for detecting objects, a platform is required that allows communication between the different components to avoid bottlenecks in the memory accesses.

The selected platform is the Xilinx Multi-Processor System on Chip (MPSoC)+Zynq UltraScale, because it allows applications to benefit from a collection of computation units with different performances and power consumption characteristics. The target device is composed by a quad-core Arm Cortex-A53 with a clock frequency up to 1,5Ghz, a dual-core Arm Cortex-R5 with a clock frequency up to 600 Mhz, a GPU Mali-400 with two cores and a Xilinx ZU4EV FPGA. The power consumption ranges between 2-24 W, which enables to fine tune optimal implementations for each embedded application.

In brief, the Zynq is a deeply versatile device capable of achieve high-performance using specialised, customised and optimised combination of traditional methods.

1.6 Hardware Implementations

The number of layers that YOLOv3 possess, produces a huge amount of data. Due to memory limitations, a series of techniques must be done to accomplish data compression. The possible methods intent to explore are:

- Binary neural networks: synthesise binary layers for FPGAs [4];
- Quantization: reduce the precision of weights[8];
- Prunning: eliminate redundancies [7];

Another method that optimises structural data, is the reuse of the same logic structures to perform computation of all layers. A multiplexing system controls the weights for the current layer being processed [2]. Since most of the layers have equal size, this is a feasible technique, yet it is necessary to take into account the overhead of the control system.

Mappings of YOLOv3 to the heterogeneous platform will explore the above optimisation methods, ranging from a baseline CPU implementation to a full custom implementation maximising the use of the available resources.

Acknowledgment

This work was partially supported by project FCT UID/EEA/00048/2019. We would like to thank Tulipp Project (H2020 No 688403) for providing the hardware.

References

[1] Jason Cong and Bingjun Xiao. Minimizing computation in convolutional neural networks. In *ICANN*, 2014.

[2] S. Himavathi, D. Anitha, and A. Muthuramalingam. Feedforward neural network implementation in fpga using layer multiplexing for effective resource utilization. *IEEE Transactions on Neural Networks*, 18(3):880–888, May 2007.

[3] D. Kang, D. Kang, J. Kang, S. Yoo, and S. Ha. Joint optimization of speed, accuracy, and energy for embedded image recognition systems. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 715–720, 2018.

[4] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2181–2191. Curran Associates, Inc., 2017.

[5] Hiroki Nakahara, Haruyoshi Yonekawa, Tomoya Fujii, and Shimpei Sato. A lightweight yolov2: A binarized cnn with a parallel support vector regression for an fpga. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, pages 31–40, New York, NY, USA, 2018. ACM.

[6] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[7] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Heng Wai Leong, Magnus Jahre, and Kees A. Vissers. FINN: A framework for fast, scalable binarized neural network inference. *CoRR*, abs/1612.07119, 2016.

[8] Xiaotian Zhu, Wengang Zhou, and Houqiang Li. Adaptive layerwise quantization for deep neural network compression. *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2018.

2 Interactive Demonstration of an Energy Efficient Object Detector Implementation on Reconfigurable Logic

Bruno Oliveira, ISR-UC and Jorge Lobo, ISR-UC

2.1 Abstract

In this interactive demonstration we present the usage of a state-of-the-art object detector, YOLOv3, in a CPU+GPU+FPGA heterogeneous platform for embedded systems. Profiles of each function were analysed in order to achieve the most energy efficient solution in the design space. A camera is used to collect the frames, a board runs the algorithm and a monitor shows the labels of the detected objects attached to the image. The main purpose of this demo is to demonstrate to the conference attendees a functional version of an object detector in a low-power platform. In addition, we also intend to encourage the development of applications in this type of devices.

2.2 Introduction

Although computer vision algorithms provide reliable results in the detection and classification of objects, high performance computational devices are required to achieve real-time processing. These platforms usually have associated a high cost and power consumption. This can be a limiting factor for battery power embedded systems where the total energy and the computational requirements became prohibitive. Therefore, power efficient implementations of object detectors has been a hot topic of research because allow its employ in a wide range of applications.

In our work we are implementing a energy efficient version of the YOLOv3 [4] object detector in a heterogeneous CPU+GPU+ FPGA platform. We are using the Sundance VCS-1, a Multi-Processor System on Chip (MPSoC) that includes a Zynq UltraScale FPGA. The combination of traditional components with shared memory allows a wide design space. Mappings of the application into the resources of the platform are being explored using a set of metrics.

The SDSoC framework, created by Xilinx, enables development and hardware acceleration of embedded applications using standard programming languages, such as C or C++. The SDS++ system compiler parses the directives of a program and makes the specific generation of a bit file for the FPGA or a binary file for the CPU. Using the Tulipp tool-chain[2] we analyse the consumption profiles collected by an external meter connected between the power supply and the processing units. This allows us to have a detailed energy profile of the tested hardware and software mapping of the implementation in the design space, allowing for informed design options and fine tuning an energy efficient implementation.

A short paper on the proposed work [1] contains more details about the target platform and the techniques that will be necessary to carry out to overcome its complexity during the implementation phase.

2.3 Overview of the Demonstrated System

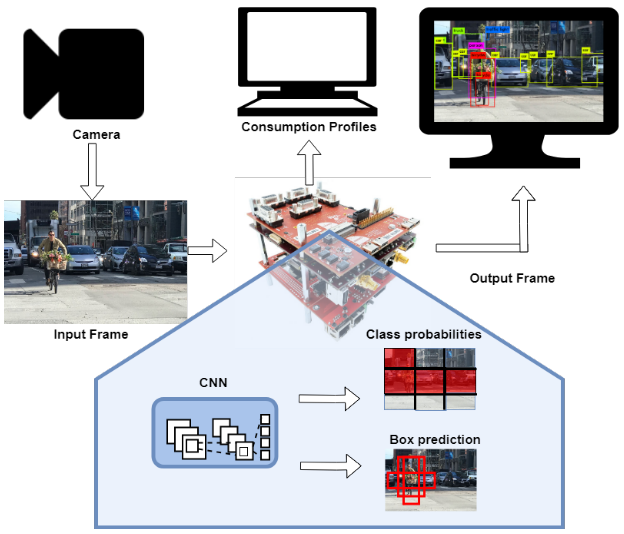

Figure 1: Overview of the demonstration system components and the data flow, with the hardware CPU+GPU+FPGA heterogenous platform at the centre.

Fig. 1 shows the interaction between the main modules of the system. Frames captured by the camera are sent to the device via a wired connection. The board runs the algorithm of the object detector with the purpose of doing the detection and classification. A host computer has access to the application's consumption profiles and shows the information in a very perceptible graph. Labels are assigned to predictions with the highest degree of confidence and the image is shown on a monitor. The frame rate of the images on the monitor will be determined in function of a set of parameters.

Beyond the demo presented, we envision that remote access to the system can not only obtain the curated image classification data, but also allow remote reconfiguration and testing of different neural network overlays. This can be used as a remote teaching lab, so that the hardware is actually deployed in a real field test, or for operational updates of working systems.

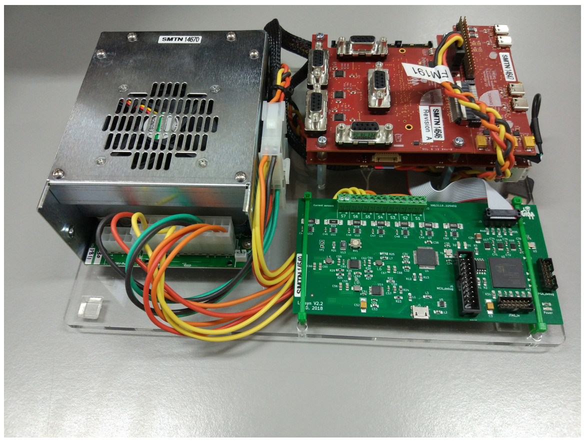

Figure 2: Hardware setup used in the demonstration, a Multi-Processor System on Chip (MPSoC) that includes a Zynq UltraScale 4EV FPGA and attached a external consumption meter [3].

2.4 Hardware and Software Interfaces

2.4.1 Hardware Overview

The selected platform, the Sundance VCS-1, is shown in fig. 2. This heterogeneous platform is a highly flexible device that makes use of reconfigurable logic combined with traditional architectures to achieve low power solutions. An external consumption meter is connected between the processing units and the power supply, in order to collect application power profiles during its run-time. This type of data is useful for the development of applications with low power requirements. A host computer connected to the meter and to the processing platform collects the power profiles logs.

2.4.2 Profiling Interface

Visualisation of the profiles enables programmers to fine tune parallelisation and optimise implementations where low power consumption is a critical factor. To this purpose, we will use the Tulipp tool-chain[2] during the development of the application. The demonstration will show the live power profiling of each function that composes the object detector. A power profile of a computer vision application made by the tool-chain is shown in fig. 3 to demonstrate its potential in the development of low power applications.

2.5 Summary

In this demo an energy efficient implementation version of YOLOv3[4] is proposed. We expect to have a working version of YOLOv3 suitable for embedded applications were trade-offs in performance are being parameterized for the

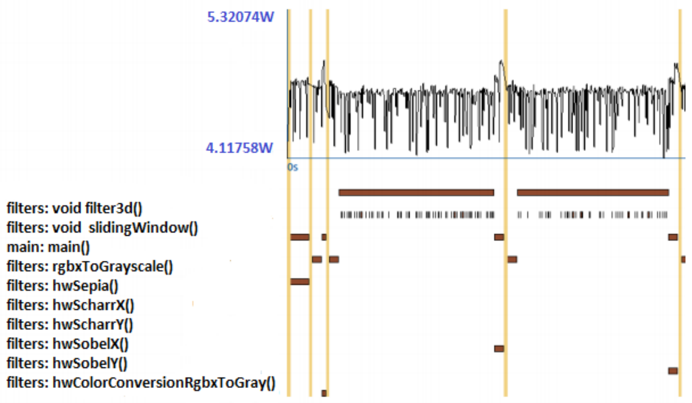

Figure 3: Screen shot of a power consumption profile of application.

targeted system. Mappings of the algorithm are being fine tuned, to achieve a low power solution, using the profiling tool. A set of metrics such as energy consumption and mean average precision(mAP) also serve to benchmark against the state-of-the-art implementations on CPU+GPU. The end application is not limited to smart cities where there is a high demand for the integration of IoT devices. In fact, remote areas with poor access to power sources and connection bandwidth, such as agriculture, forest monitoring, etc, will be the key end applications where our system can be integrated, providing a low power edge computing solution.

Conference visitors can place a range of objects in front of the camera and observe its labeling on the monitor. They can also observe the power profiles of each function of the application on a computer. The live results achieved by this implementation can be evaluated by comparing with the data collected during the baseline benchmarks on standard architectures.

Acknowledgment

We would like to thank the Tulipp Project (H2020 No 688403) for providing the hardware necessary to this research.

References

[1] G. Oliveira, Bruno and Lobo, Jorge. YOLOv3 Object Detector: Exploring Energy Efficient Implementations on Reconfigurable Logic. In *Proceedings of the 15th Portuguese Meeting on Reconfigurable Systems.*

[2] Tobias Kalb, Lester Kalms, Diana Göhringer, Carlota Pons, Fabien Marty, Ananya Muddukrishna, Magnus Jahre, Per Gunnar Kjeldsberg, Boitumelo

Ruf, Tobias Schuchert, et al. Tulipp: Towards ubiquitous low-power image processing platforms. In *Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2016 International Conference on*, pages 306–311. IEEE, 2016.

[3] Ananya Muddukrishna, Asbjørn Djupdal, and Magnus Jahre. Power profiling of embedded vision applications in the tulipp project. 2017.

[4] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.