POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Data Science and Engineering

Master Degree Thesis

Deploying Deep Learning on FPGA: an assessment of ConvNets performance on Xilinx Zynq MPSoC using Vitis-AI development platform

Supervisors
Prof. Andrea Calimera
Dr. Roberto Giorgio Rizzo

Candidate
Gabriele Cuni

Academic Year 2021-2022

Acknowledgements

I would like to thank Professor Andrea Calimera for giving me the opportunity to undertake this exciting project and Doctor Roberto Giorgio Rizzo thanks to whom it was possible to overcome the most difficult challenges imposed by the thesis.

I thank my parents and my family who allowed me to finish this challenging journey, always supporting me in my choices and encouraging me in difficulties.

I thank my friends who I spent the best moments with and in particular Lorenzo and Emanuele with whom I started this adventure and shared the joys and difficulties.

Finally, I thank Carlotta, essential part of my life and beloved partner who knows my feelings and fears the most; the person who I share my desires and ideals with. Thank you for always being by my side.

Thanks to everyone, *"I am what I am because of who we all are"*

Abstract

Deep neural networks are one of the most promising technologies in the IoT field, nevertheless they require a high number of operations to be executed. IoT application development is often subject to strict limitations in terms of hardware resources, which makes it complex to use deep machine learning techniques on edge devices. Additionally, edge computing often requires low execution times, in order to be suitable for real-time applications. The above contrasting requirements place a challenging technology problem, which can be addressed by deploying deep neural networks on an efficient and optimised hardware. Field programmable gate array (FPGA) can be a viable alternative to GPUs to accelerate deep neural network inference, even on edge devices. In this thesis, we propose an assessment of ConvNets performance, achieved through Vitis-AI on a Zynq UltraScale+ MPSoC. The assessment is done by testing a set of Mobilenets, which has been obtained by varying the width multiplier and the input resolution, on different FPGA configurations, that are obtained by varying the allocated resources to the deep learning processing unit. On one hand the results show a high throughput, which is compatible with real-time application, even on the smallest available FPGA architecture. On the other hand, all models suffer a large accuracy reduction, that is due to the need to use the post-training quantization technique. Although, taking into account, as an upper-bound, the achievable accuracy using a quantization aware-training technique, the results show that the deployment of deep neural networks on FPGA is a viable option.

Contents

Chapter 1

Introduction

1.1 Context

Nowadays a massive amount of data is made from sensors and smart devices and the increase of computing power is bringing a change in the data analysis field, more and more services and algorithms are run on the edge rather than exclusively in the cloud. Specifically, deep learning models are bringing the greatest innovations, concerning the data analysis areas [28]. We live in a connected world thanks to the spreading of storage and computing devices, such as data centers, personal computers, smartphones and wearable devices. According to the Statista document [26], There are 13.1 billion connected devices at the end of 2022, meanwhile the installed ones are much more with a number evaluation of 42.62 billion. Furthermore, the forecasted number of connected devices in 2030 is 29.4 billion, which means a 350% increase in a decade according to [26]. The thriving increase in the data sources for big data pushes for a paradigm shift: from a centralized data analysis on large-scale data centers to a decentralized computing on edge-smart devices. Nevertheless, the decentralized paradigm creates severe challenges on the network capacity [2]. In addition to the network workload increasing, there are some applications that have tight latency constraints and security availability requirements, which are incompatible with a long range communication with a cloud server; such applications require the edge computing paradigm for the implementation [7].

Given the above reasoning, edge computing is a promising technology to be used as an alternative to the cloud computing [21], specifically, for the time constrained applications. Although, the hybrid approach using both cloud and edge computing brings major advantages such as: reduced network usage, response speed improvement and cloud backups [18].

The increased computation power and data availability has allowed a considerable diffusion and use of deep learning techniques, these algorithms have had a major impact in the fields of computer vision and natural language processing [3], [17]. Although, same special application domains, such as autonomous driving and the more general concept know as Internet of Vehicles are limited due to the following factors [28]:

- Communication cost: constant communication between edge devices and cloud servers without splitting the computing phase over both devices lead to a massive

amount of stress on the network infrastructure.

- Latency: There are not guarantees that the access time to a cloud server is always below a given threshold and might exceed the time requirement for critical applications as cooperative platooning or cooperative autonomous driving [16].
- Reliability: In presence of network failure the access to a cloud service it is not guarantee, which is a major issue for many industrial application.
- Privacy: Sending data acquired from sensors and required for deep learning applications could pose security and privacy issues.

The above mentioned challenges of the IoT and deep learning can be relieved by edge computing since its proximity to the users. The convergence of edge computing and artificial intelligence is pushed since they benefit each other by providing an edge intelligence. The aim is to achieve edge intelligence, which will push the transition from the cloud to the edge computation [28]. This approach brings multiple benefits:

- A little-big design can be used with a small deep learning model deployed near the final user and a big neural model deployed on the cloud server ready to be used when additional computing power is required, thus reducing the network usage and the application latency [14].
- The reliability of the application is increased since the edge intelligence can be exploited, even in the presence of network failures.
- Privacy issues can be relieved, as data can be saved on the edge or can be aggregated prior to transmission.

The effort to bring neural networks to the edge is justified by the daring increase in research in this area in recent years. A set of different models, such as Convolutional Neural Network (CNN or CovNets) and Recurrent Neural Networks (RNN) have been developed and tested for image, speech and video recognition tasks. The state-of-the-art models have reached a classification accuracy, object detection capability and feature extraction proficiency such as to allow their implementation within finished and usable software products.In general, neural networks and specifically deep learning models have a high approximation capability which makes them a perfect candidate for a wide range of tasks and a promising solution for many artificial intelligent applications [27]. The high approximation capability is fundamental in order to be subject to the required techniques for the implementation on edge devices.

Although, deep neural networks require a high storage and computational capacity. Take as an example the largest state-of-the-at model, the Megatron-Turing NLG 530B with 530 billion parameters [22]. Therefore, in addition to the advantages, the combined use of AI and edge computing poses new implementation challenges such as the hardware choice.

The hardware used to enable the edge intelligence range from general purpose computer components to special purpose devices and boards that can be configured or designed specifically for the aim:

- Graphics Processing Unit (GPU) is the most spread technology, since its compatibility and performance.
- Field Programmable Gate Array (FPGA) which is an efficient programmable hardware usually used as a prototyping environment.
- Application Specific Integrated Circuit (ASIC) is special purpose hardware, each ASIC has a custom design specifically for the aim application.

CPUs are not in the above list because they perform from 10 up to 100 GFLOP per second, but they have a low power efficiency, that is usually less than 1 GOP/J, therefore they are not used for deep learning both on cloud or edge computing. Alternatively, the performance offered by the GPUs, with peak operations up to 11 TFLOP/s, makes them an ideal candidate for deep learning applications. In addition, GPUs are becoming the main choice for deep neural networks since the major frameworks such as Tensorflow, Pytorch and Caffe, support them with easy to use interfaces [27]. Nevertheless, GPUs use a significant amount of energy during the computation, which is a key performance metric for edge computing. ASIC design guarantees high throughput and low energy consumption, but they require a very long development time and high cost compared with other hardware solutions. Consequently, FPGAs are an excellent trade off between low power consumption, high throughput and configurability at a reasonable cost [5].

1.2 Challenges

The first challenge encountered deploying neural networks on FPGAs is the need to compress the network. It is especially necessary for ConvNets as they require a significant amount of memory. Furthermore, deep learning models are usually trained with weights and activations stored with floating point representation on 32-bit. Often, in order to achieve high efficiency, it is necessary to transform the data representation into 8-bit fixed points, at the same time reducing the memory occupation of the model.

The above is a hardware challenge since not all the FPGAs have all the necessary memory or do not allow floating point computation. However, the greatest challenge is due to the lack of widespread tools for implementing deep learning on FPGAs. As mentioned, neural networks need some steps to be used on acceleration hardware boards, but there are not standard solutions that provide the necessary flow. Development flow tools are provided by hardware manufacturers and need to be explored and standardized to promote the use of FPGAs in the field of smart edge computing.

Finally, there is not an extensive literature available that directly concerns the analysis and creation of a stable flow for the implementation of deep neural networks on FPGAs.

1.3 Motivations

This thesis work proposes FPGAs as an efficient alternative to GPUs. Thanks to their high parallelism, FPGAs are an excellent hardware option for the implementation of deep neural networks, because they are more energy efficient with respect to GPUs [5].

Moreover, ConvNets hardware accelerators are going towards FPGAs rather than GPUs accelerators for two reasons. The first reason is that FPGAs are showing high capabilities for edge intelligence application [1]. The second one is their customization potential as FPGAs are a configurable hardware accelerator cable to satisfy very different design constraints. Therefore, this hardware technology is an exciting solution for all embedded applications that need energy saving and high computation performance at the same time [4]. Surely the possibility of being reprogrammed makes the FPGA a technology that will be exploited more and more in the research field, but also in the industrial applications. This is because the only more efficient and performing alternative is the design of an ASIC which for obvious reasons is a non-versatile technology, complex to design and with high access costs.

Finally, this research work was carried out because at the moment there are no standardized development flows for the deployment of deep neural networks on FPGAs. The absence of standardized tools is certainly a limitation to the spread of edge computing as reported in the survey [28]. The presence of tools for the deployment of neural networks on FPGAs such as TFlite is for microcontrollers would be a great incentive for the diffusion of this technology in the edge computing field.

1.4 Contributions

This thesis work proposes a development flow that accepts as input a state-of-the-art convolutional neural network pre-trained on the Imagenet dataset and reproduces as output an executable model which can be deployed on a Xilinx FPGA, the Zynq UltraScale+ MPSoC. The proposed flow is made over the Xilinx Vitis AI environment and the Tensorflow framework and it deals with all the aspects necessary for the treatment of the models. Here are listed all the phases of the developed flow:

- The retrieval of the models required through the Tensorflow framework. The software is optimized to download and stores all the Mobilenet models that we have used during the thesis work, but other Convnets can be optimized by the flow as well.
- The preparation of the validation datasets starting from the Imagenet validation dataset by applying the required pre-processing and selecting the images for the quantization phase.
- The quantization of the models according to the post-training quantization technique and the storing of the intermediate models representation which still are able to be evaluated through the Tensorflow framework.
- The evaluation of the quantized models and non-quantized models in terms of accuracy. All the evaluations are executed on the previously mentioned validation dataset.
- The compilation of the quantized models in the instruction set of the FPGA. Each model is compiled for each FPGA hardware configuration that was tested.

- The deployment of the compiled models on the board and the evaluation of the models in terms of throughput.

During the thesis eight different FPGA hardware configurations are made and evaluated over the Zynq UltraScale+ MPSoC. The proposed architectures are generated by selecting different resource utilization on the FPGA.

Thanks to the proposed development flow sixteen different Mobilenets are tested for each FPGA hardware configuration. The testing phase had the objective to evaluate the performance obtained in terms of inference speed and the accuracy by the models once run on the board.

Chapter 2

Background

2.1 Mobilenet

The Mobilenet model is an efficient convolutional neural network, which is developed by Google Inc. and it is designed for the image recognition task. The main purpose of the model is to be a flexible, small and low latency model that can be easily shaped to meet the design constraints for embedded vision and edge mobile applications [6].

The Mobilenet main objective is optimizing the latency. In order to reach its goal the neural network is built of depthwise separable filters, which are an alternative version of the standard convolution layers. The depthwise separable convolution splits the filter and the combination phases in two different layers, as opposed to the traditional convolution. Thanks to this technique, which is extremely efficient, the Mobilenet significantly reduces both the model size and the number of operations for each layer [6]. In table 2.1 the Mobilenet topology is shown.

The above reasoning shows how the model achieves low latency, instead the flexibility is guaranteed by two parameters the α so called width multiplier and the resolution multiplier.

2.1.1 Width Multiplier

The task of the width multiplier parameter also known as α is to shrink the Mobilenet uniformly at each layer. For a given layer holds the formula (2.1) provided by [6], where D_f is the width and height of the input feature map, D_k is the dimension of a square kernel, M is the number of the input channels, N is the number of output channels and $\alpha \in (0,1]$. By varying the width multiplier parameter can be applied a trade off between accuracy and latency and in this thesis the set [1, 0.75, 0.5, 0.25] is used.

$$D_k \cdot D_k \cdot \alpha M \cdot D_f \cdot D_f + \alpha M \cdot \alpha N \cdot D_f \cdot D_f \tag{2.1}$$

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 2.1: Mobilenet Architecture from [6]

2.1.2 Resolution Multiplier

The other parameter is the resolution multiplier also known as input image resolution, because in practice it is set by the input resolution. By changing this parameter all the Mobilenet layers are reduced by the same multiplier. For a given layer holds the formula (2.2) provided by [6], where ρ is the resolution multiplier. In this thesis the resolution parameter is used with these values: 224, 192, 160 and 128.

$$D_k \cdot D_k \cdot \alpha M \cdot \rho D_f \cdot \rho D_f + \alpha M \cdot \alpha N \cdot \rho D_f \cdot \rho D_f \tag{2.2}$$

2.2 Field Programmable Gate Array - FPGA

FPGAs are a flexible hardware device for implementing custom functionality at a low design and production cost. In order to be an off-the-shelf programmable device, they are made by a set of programmable logic cells, also known as configurable logic blocks. The cells are linked by an interconnection network, which can be programmed as the logic

blocks. Additionally, around the device, there are a set of input and output connections in order to link the programmable logic with the outside system. The programmable logic is also equipped with other components, such as digital signal processing (DSP) blocks, which are the embedded blocks in charge of compute intensive arithmetic operation, such as convolution operations for the deep learning algorithms. The memory on-chip is composed of block RAMs (BRAM), lookup tables (LUT) and flip-flops (FF). Finally, there are other essential hardware components, such as the clock management unit and high speed I/O links [20]. Figure 2.2 shows the structure described above.

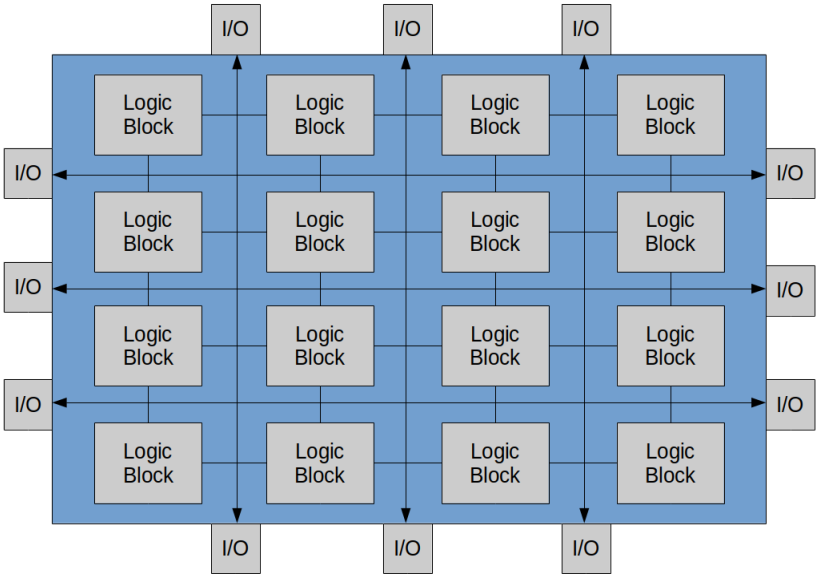


Figure 2.2: FPGA basic schema

FPGAs are an optimum hardware solution for both battery operated devices and cloud servers and can be considered a deep learning accelerator as they have high hardware parallelism and associative operations. Moreover they are candidates for edge computing and edge intelligence applications, thanks to the high efficiency and flexibility. In addition, the possibility to make deep pipeline and multi thread firmware is the perfect spot for the feed forward step execution in neural networks [20].

2.3 Zynq UltraScale+ MPSoC Overview

The Zynq UltraScale+ MPSoC ZCU104 is an evaluation board whose purpose is to enable the design and testing of embedded machine learning vision tasks and applications, figure 2.3. The board is meant to build, prototype and test accelerated software applications such as drones and medical imaging, Advanced Driver Assistance Systems (ADAS) and surveillance.

The diagram shown in figure 2.4 represents the on board microchip Zynq UltraScale+

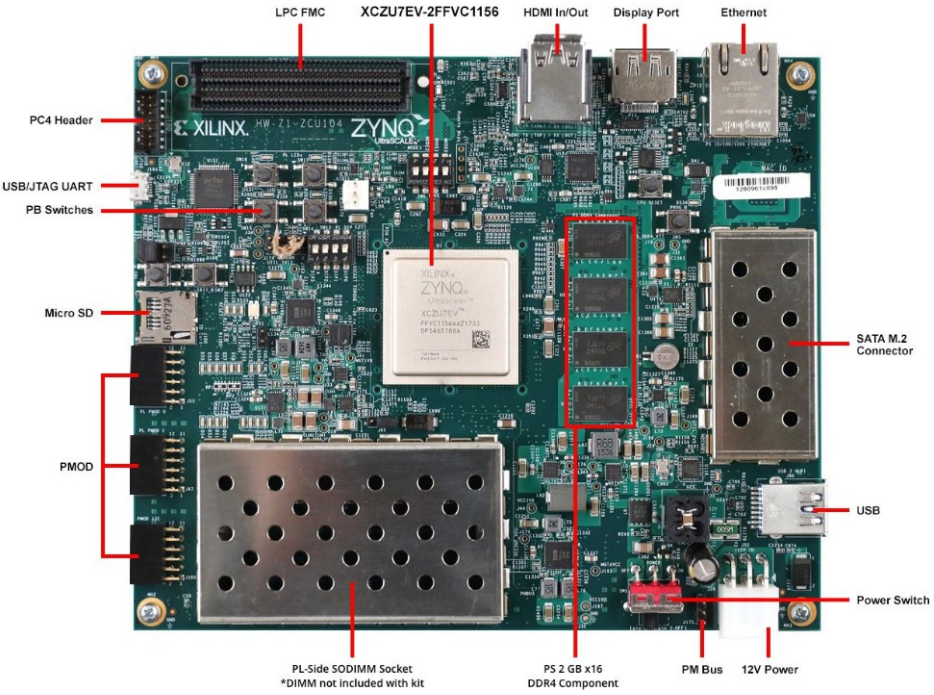


Figure 2.3: Zynq UltraScale+ XCZU7EV-2FFVC1156 MPSoC photo taken from [13]

XCZU7EV-2FFVC1156 MPSoC. It is composed of a Processing System (PS) and a Programmable Logic (PL). The processing system is equipped with 2 GB DDR4 RAM and three processors:

- Application Processing Unit which is a quad-core Arm Cortex-A53 ,with a frequency up to 1.5 GHz
- Real-time Processing Unit (RPU) which is a dual-core Arm Cortex-R5, with a maximum frequency of 600 MHz
- Graphics Processing Unit (GPU) which is a Arm Mali-400 MP2, with a frequency up to 667 MHz

In addition, it is equipped with high speed connectivity such as DisplayPort v1.2a, USB 3.0, SATA 3.1, PCIe 1.0/2.0 and PS-GTR. Finally, the processing system is equipped with general connectivity such as GigE, USB 2.0, CAN, UART, SPI, quand SPI NOR, NAND and SD/eMMC [12].

The programmable logic is a 16nm FinFET+ FPGA that is the home of the Deep learning Processing Unit (DPU) [11][8]. It is equipped with a Block RAM that is the standard on-chip random access memory for FPGA, useful to process big data amounts without access to the processing system DDR4 RAM. Furthermore, the Ultrascale+ devices are equipped with a Ultra RAM that is a larger and flexible memory block. This new type of memory block provides up to 500 Mb of excess data storage directly available

on-chip [9]. Finally, the programmable logic has a video codec unit which incorporates the standards H.264 and H.265 and it is linked with the processing system with a high speed connectivity PCe Gen4 [12].

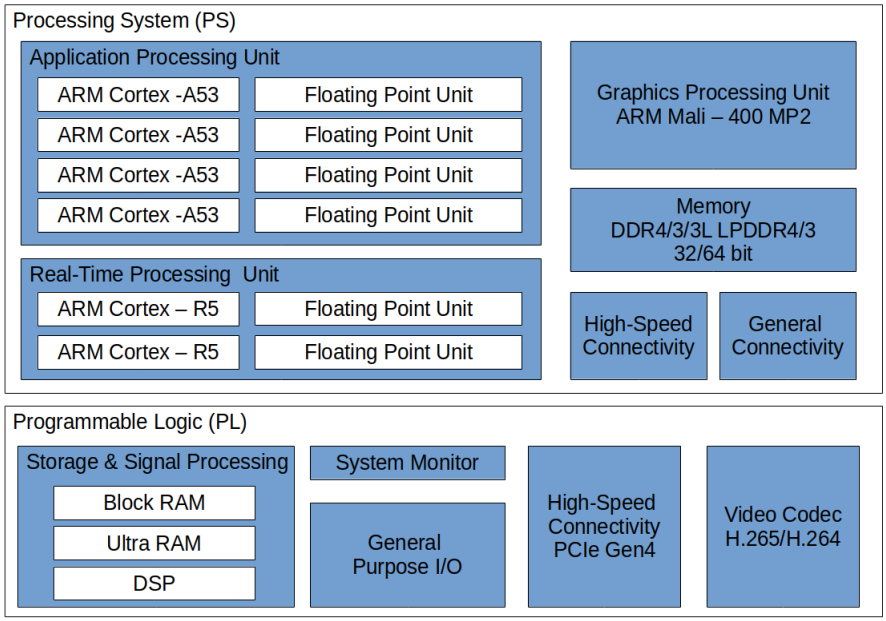


Figure 2.4: Hardware diagram of the Zynq UltraScale+ MPSoC

2.4 PYNQ

PYNQ is an embedded open-source python project made by Xilinx in order to be deployed on several Xilinx platforms and it is installed on the Zynq board by means of a bootable linux image.

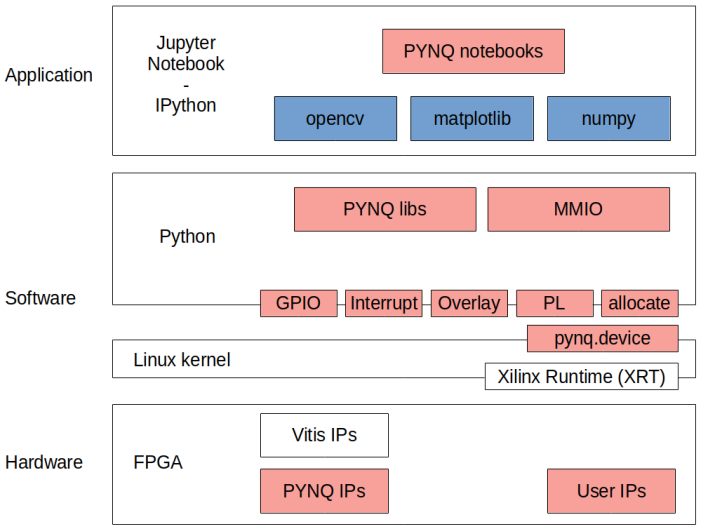


Figure 2.5: PYNQ application schema

This project enabled software developers without hardware design experiences to access the potential of the XIlinx Programmable Logic and Deep learning Processing Unit. The key features of PYNQ which allows an easy use of the Programmable Logic is the PYNQ overlays combined with the possibility to code in Python language. These overlays are ready to use hardware libraries made by hardware engineers, which are provided to embedded developers, as software libraries are provided to application developers. Therefore overlays are FPGA designs that provide a software python interface which links together the software running on the processing system and the firmware running on the programmable logic that is hardware accelerated. Using the PYNQ overlays it is possible to make a hardware accelerated application without hardware expertise using premade python wrappers of FPGA designs. The starting point of a PYNQ software that uses overlays is the loading of the base overlay which is the reference design of the board made by the Xilinx Vitis AI platform. The base overlay is composed by three files:

- a bitstream file in order to shape the FPGA connections.
- a Vivado design file to set up the programmable logic IP
- a bit file that provides the python wrappers

The configuration files of the base overlay are made by the Vitis AI environment when a new deep learning processing unit is created.

2.5 Xilinx Vitis AI

The Vitis AI development environment is the set of tools and libraries given by Xilinx in order to deploy deep neural networks on the Xilinx boards. It is distributed by a Docker system which includes machine learning frameworks, Vitis AI models, development modules and DPU overlays.

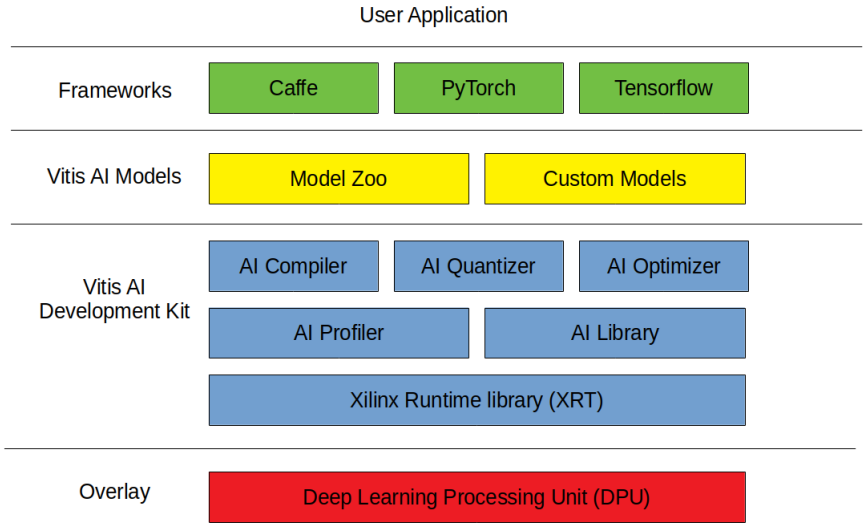


Figure 2.6: Xilinx Vitis AI block schema

The frameworks made available are: Tensorflow 1 and Tensorflow 2, PyTorch and Caffe. Any framework has its own development flow inside the development environment and its own Vitis AI Classes and Methods. The neural network models can be taken by the Vitis AI model zoo which contains a selection of pre-optimized models already quantized and ready to be deployed on the Xilinx devices, on the other hand custom models can be imported inside the development environment and optimized by the user. The Vitis AI development kit provides multiple software utilities useful to transform a custom model into a deployable model on a Xilinx board or device. These utilities are [10]:

- AI Optimizer: An optional utility that reduces the model complexity by pruning its neurons
- AI Quantizer: It is a fundamental step in order to deploy the model on the DPU. It transforms a floating point model into a fixed point 8-bit integer model and it supports post-training quantization and quantization aware-training. Following the Tensorflow 2 flow the quantizer is a Python class able to load a float model either in h5 format or saved model format and it requires a representative calibration

dataset which is used to calibrate the model weights. The dataset can be a subset of the train or validation one, we used 4096 images taken by the imagenet validation dataset.

- AI Compiler: It is the utility which transforms the quantized model in a format that is compatible with the DPU microarchitecture. Both the tensorflow 1 and 2 made available a compiler that is callable from a shell command. The command requires as input the quantized model by Vitis AI and an arch file containing the fingerprint of the target DPU.
- AI Profiler: It is a built-in utility that analyze efficiency of the model
- AI Library: It is an API able to link the code written by the software developer with the DPU through the Vitis AI Runtime.

2.6 Deep learning Processing Unit - DPU

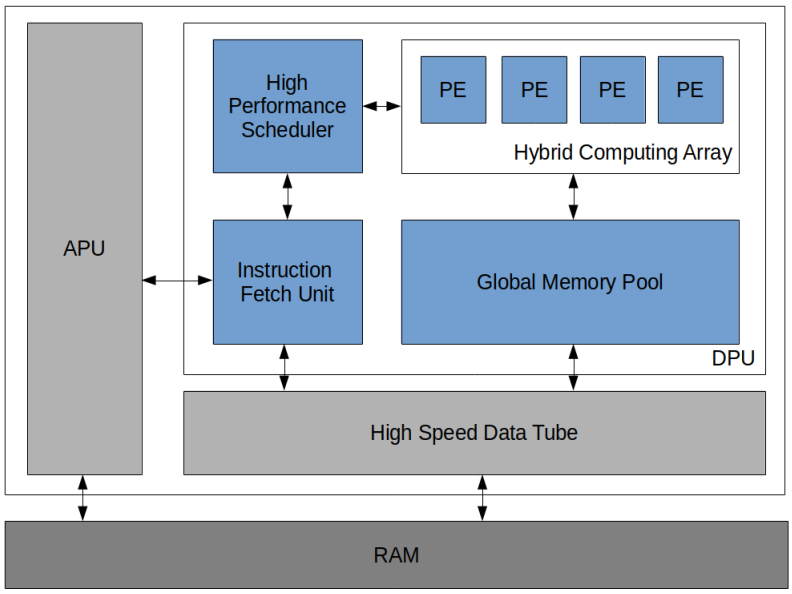


Figure 2.7: DPU hardware diagram

The Deep learning Processing Unit is the fundamental feature of the board, it is a set of parameterizable IP cores that are implemented on the programmable logic of the Xilinx board. The DPU has a specialized instruction set designed in order to accelerate the software application for computer vision as deep neural networks [8]. As an example the above mentioned instruction set supports a set of layers such as: convolution, depthwise convolution, max pooling and fully connected. The DPU designed for the board Zynq

UltraScale+ MPSoC ZCU104 is called DPUCZDX8G which means it is designed for CNN layer, that the quantization method is DECENT, that it is in 8 bit and it is for general purpose instead of high throughput, low latency or cost optimized [10].

Chapter 3

Assessment Organization

In this chapter the proposed deployment flow will be analyzed in detail. Here a generic introduction to its functioning is made and in the following sections each phase is analyzed point by point.

By observing the figure 3.1 it can be immediately understood that the flow is divided into three main phases: the model building, the hardware building and the deployment phases.

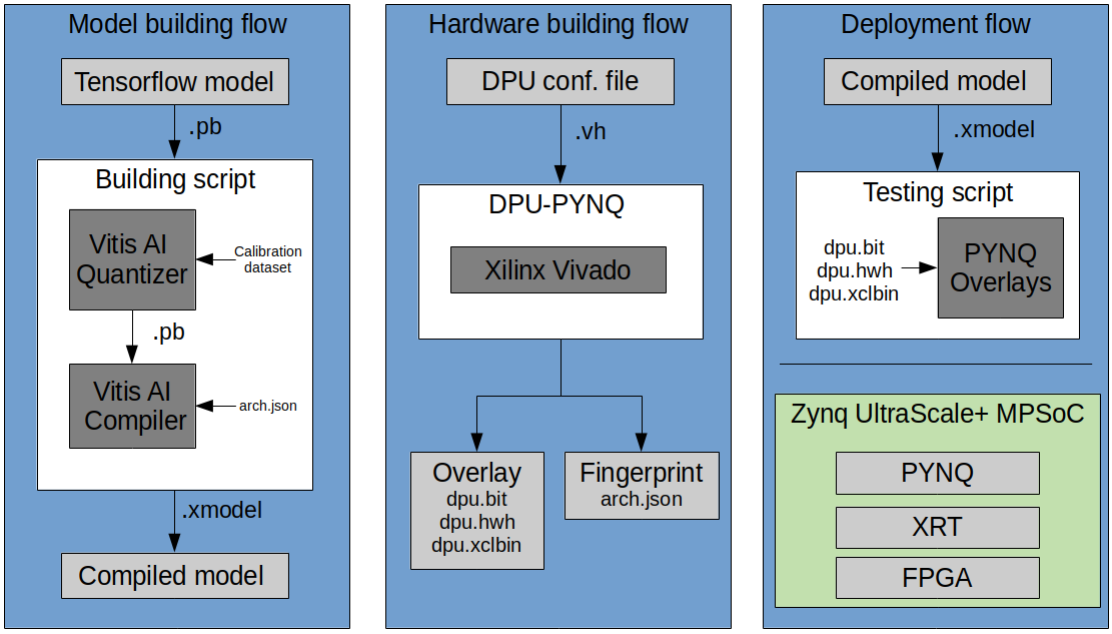


Figure 3.1: The proposed deployment flow

The hardware building phase is done on the host computer. The objective is the creation of different DPU overlays in order to program the hardware configuration of the FPGA. These overlays are made with the DPU-PYNQ environment, which creates the

DPU specifically for the PYNQ environment that is installed on the Zynq board. The two main results of this flow are the overlay files and the fingerprint file. This phase is the first one that must be executed, because the resulting files are mandatory to execute the other two flows.

The second phase is the model building one, which is also executed on the host computer. The main operations are the quantization and the compilation that are paramount to have a deployable deep neural network graph. These steps are done by the proposed Python building script, thanks to the Vitis AI library. It is important to notice that the compiler requires the fingerprint file created in the previous flow.

The last flow is the deployment on the Zynq Ultrascale+ MPSoC, obviously this is the last phase to be done as it requires the compiled model from the model building flow and the DPU overlay files from the hardware building flow. During this phase is executed the testing script in Python that evaluates the inference throughput of the compiled model on the FPGA.

3.1 DPU overlay generation

The first step in order to deploy a neural network on the programmable logic of the embedded device is the creation of the PYNQ DPU overlay. The overlay is made on the host computer of the developer thanks to the Vitis AI environment which uses the Vitis Software Platform in order to synthesize the hardware description. All the DPU configuration knobs can be modified by a configuration file on the development docker system by setting the corresponding flag. All available configuration parameters are listed below:

- Architecture of the DPUCZDX8G
  - This knob is related to the level of parallelism of the convolution unit.
  - The available preconfigured designs are: B4096, B3136, B2304, B1600, B1152, B1024, B800, B512.
  - The above mentioned numbers in the architecture's names are the peak operation clock that is possible on the given architecture. As a result of the parallelism the B512 has a peak operation/clock of 512.
- Number of cores
  - This knob sets the number of DPU instantiated and the maximum of four IP cores can be configured.
  - It is set by default to 1.
- Ram usage
  - This knob selects the amount of on-chip memory (BRAM) used in the DPU architecture which stores the intermediate data as weights, biases and activations.

  - The available options are high or low. They correspond to a fixed amount of BRAM for a given DPU architecture.

- DSP usage
  - This knob can be set high or low.
  - The low configuration will use DSP only for multiplication in the convolution.
  - The high configuration will use DSP for both multiplication and accumulation.

- Low power mode
  - This option can be on or off and it disables the PE clock when the DPU is idle.

- Extra operations
  - Channel augmentation
  - Depth wise convolution
  - Average pooling
  - Element wise multiplication
  - ReLU type
  - Softmax

The compilation makes the three overlay files descripting the FPGA hardware, the hardware description fingerprint file that is the input of the Vitis AI compiler and a report with all the features of the DPU. The report shows useful parameters in order to compare the different DPU architectures such as lookup table, lookup table used as memory, registers, BRAM cells, URAM cells and digital signal processor number.

	LUT	LUTasMem	REG	BRAM	URAM	DSP
B4096	102319	11355	195936	168	92	1380
B3136	87435	7676	157763	146	84	1096
B2304	78380	6663	136841	124	76	844
B1600	70699	5744	116418	102	68	624
B1152	61818	5108	92833	36	76	424
B1024	63380	4832	94259	90	30	436
B800	56357	4428	80137	30	68	314
B512	51371	3740	67278	26	30	220

Table 3.1: DPU hardware features

It can be seen in table 3.1, that in general, reducing the number of operations required per clock results in a reduction in the use of all seven different types of resources. However, the B1152 architecture has an increase in URAM compared to the B1600 and

unexpectedly has fewer LUTs, REGs, BRAMs and DSPs than the B1024. This peculiarity will have an effect on the final results which can be appreciated in particular in the figure 4.3.

The other DPU options are taken constant during the experiments and they are set as follow: The core number is set to one, the RAM usage is set to low, the DPS usage is set to high, the low power option is disabled, the channel augmentation is enabled, the depth wise convolution is enabled, the average pooling is enabled, the element wise multiplication is disabled and the ReLU leaky option is enabled.

3.2 Quantization

Neural networks are trained with a 32-bit floating point representation for both weights and activations. Inference is usually performed with this floating point representation, nevertheless, both tasks can be performed with a reduced numerical representation with a small accuracy loss. Quantization can be applied to either weights or weights and activation (3.2) and the target representation can be a smaller float format such as 16-bit float, a fixed representation such as 8-bit integer, or a custom representation. There are three types of benefits using the quantization technique:

- Time and energy cost reduction to access the memory.
- Since operations can be performed on small, fast and efficient hardware, better performance is achieved even in the computation phase.
- The appropriately quantized model can be run on hardware platforms without the floating point unit (FPU), allowing deployment in otherwise impossible environments.

Given the above reasoning the 8-bit integer representation is the most used option, since the computations can be performed by the ALU without a FPU. it is also the minimum unit register format inside the CPU and the RAM is byte addressed. A smaller representation would require either custom hardware support or custom packing/unpacking drivers. Finally, as shown in figure 3.2, the quantization technique can be applied in two different ways that are presented below.

3.2.1 Post-training quantization

The quantization performed during this thesis is called post train quantization which means that the model is trained in the normal floating point format and after the training phase the weights and activations are recast to a lower precision format that in this case is the fixed point integer over 8 bit. The advantages of this method are its simplicity, the possibility that the model does not need to be retrained and the labels of the train or validation dataset are not necessary. This advantage was necessary because the training of a state-of-the-art network on the imagenet dataset would have been impossible given the computational limitations encountered and the timing of the thesis work. The disadvantage is a significant accuracy drop on complex tasks, especially for small neural

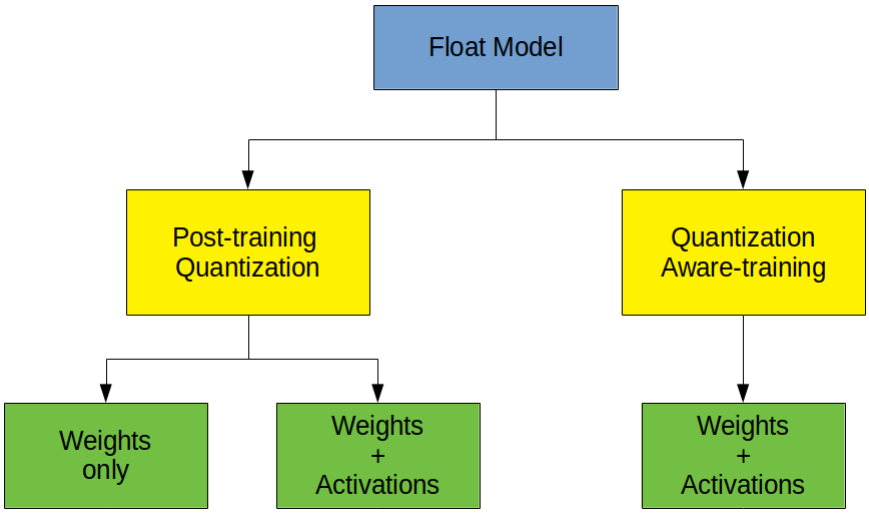


Figure 3.2: Quantization methods

networks, as they have a reduced approximation capacity, compared to larger models, which are more redundant.

There are many conversion techniques to represent a real number using an integer. The Xilinx Vitis AI quantizer is based on the tensorflow one, therefore it is an affine quantizer [25] that executes an asymmetric integer quantization. This method is designed to deal with the ReLU activation, which has not a centered distribution.

$$realvalue = (int8value - z) \cdot \Delta \tag{3.1}$$

$$int8value = \frac{realvalue}{\Delta} + z \tag{3.2}$$

The formulas (3.1) and (3.2) are the implementation of the quantization on Tensorflow, where the 8-bit values are in two's complement. The zero point and the scale factor are computed either for each tensor or for each channel. Additionally, the zero point is zero for weights and the 8 bit range is [-127,127], although the zero point is in the range [-128,127] for the activations, therefore the activation 8 bit range is [-128,127]. [25]

Activations	$\Delta = \frac{a_{max} - a_{min}}{2^{Nbit} - 1}$	$z = \frac{a_{max} - a_{min}}{2}$
Weights	$\Delta = \frac{2 \cdot \max(\lvert a_{max} \rvert, \lvert a_{min} \rvert)}{2^{Nbit} - 1}$	$z = 0$

Table 3.2: How to determine Δ and z

Let be a_{max} and a_{min} the minimum and maximum floating point values in a tensor a,

the table 3.2 shows how to compute Δ and z for that tensor. Given the above formulas Δ and z can be computed for the weights by reading the weights from the neural networks, instead for the activations these two parameters depend on the input dataset distribution. Therefore, during the quantization step a calibration dataset is made from the imagnet validation dataset by selecting 4096 images without their labels and are stored in a tensorflow format, ready to be input of the Vitis AI quantizer. The calibration dataset is a set of representative input images that is used to compute a_{max} and a_{min} of the activation tensors.

Finally, since weights and activations are mapped to 8-bit starting from floating point, the main layer operation $w \cdot x$ can overflow, therefore the intermediate results must be stored in a bigger integer representation and then converted back to 8 bit [19].

3.2.2 Quantization aware-training

As mentioned, the post-training quantization technique has the disadvantage of a significant accuracy reduction. A solution is found by resorting to quantization-aware training, i.e. quantizing the model during training, so that the gradient descent based algorithm takes into account the limited representation of the data on 8 bits and modifies the weights of the neural network accordingly. However, the quantization during the training phase generates problems to the gradient descent algorithm, specially with small weights updates that are converted to zero during back propagation, making it impossible to converge. A solution is adding fake quantization nodes to the models, figure 3.3, therefore quantization is used only in the forward pass, where the added nodes simulate the quantization effect, but the actual computation is done in float, then for the back propagation step it is used a differentiable function in order to avoid the quantization nodes [19].

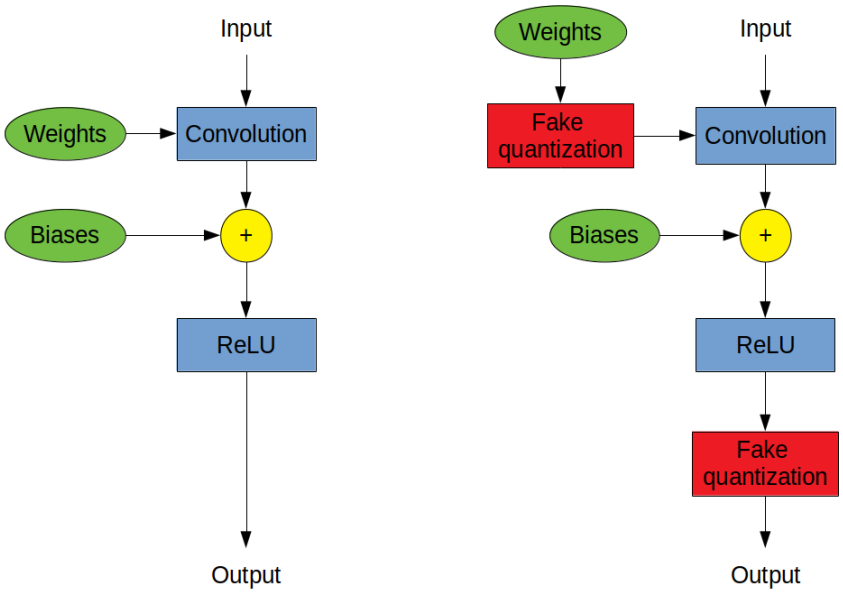


Figure 3.3: Fake quantization node positioning

In the figure 3.3 can be seen where the nodes for the fake quantization have been inserted both for the weights and for the activations, in particular thanks to the node for the quantization of the activations it is possible to directly store the values a_{max} and a_{min} so Δ and z can be computed without needing a quantization dataset.

3.3 Quantized models evaluation

To evaluate the accuracy of the quantized models, the entire validation dataset of imagenet was used, which includes 50,000 images, 1,000 classes and 50 images per class. The dataset was pre-processed and saved in a TFrecord data structure, so that it could be read easily by the validation script executed within the Vitis AI environment. The dataset thus read from the TFrecord file has undergone a further pre-processing in order to reflect the pre-processing originally performed to train the Mobilenets [15]. The metric selected for the evaluation was the categorical accuracy provided by Tensorflow and evaluated in the two variants top-1 and top-5 [24].

3.4 Models compilation

The purpose of the Vitis AI compiler is to convert the instruction set of the quantized network graph to a DPU instruction set by applying optimization techniques. The Vitis AI environment compriend a family of compiler ables to target any Xilinx DPUs for different boards.

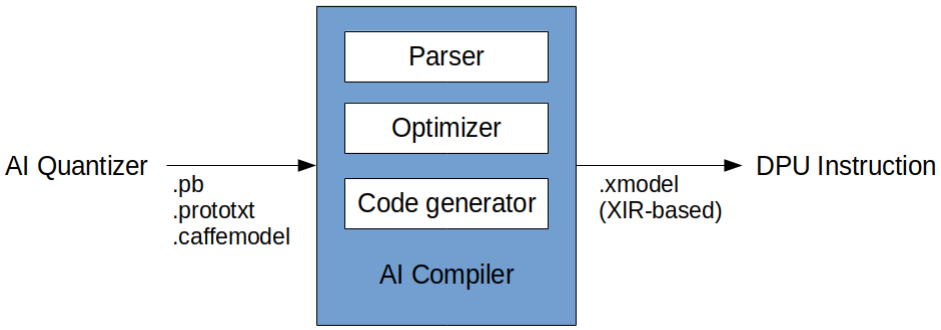


Figure 3.4: Xilinx AI Compiler detailed schema

Figure 3.4 shows the compiler steps in order to convert the instruction set. The first step is the parsing of the quantized network graph aiming to create an intermediate representation (IR). During the second step on the IR are applied multiple optimization techniques, such as node fusion, inherent parallelism exploiting and data reuse. The last step generates the Xilinx model, which is compatible with the selected DPU architecture and is based on the DPU instruction set, that in this thesis is the DPUCZDX8G [10].

Among the available compilers we used the Tensorflow 2 one, which requires as input the graph of a neural network originally created and trained with Tensorflow 2 and then

quantized with the Xilinx quantizer. The quantized model is still in the pb format and it is converted in the xilinx model format thanks to the above mentioned procedure. Additionally, the file containing the DPU fingerprint with json extension is also required. There is one fingerprint file for each DPU and they were generated when the DPU was created.

3.5 Deployment on the Zynq UltraScale+ MPSoC ZCU104 board

The deployment of the neural networks on the board starts by uploading the base overlay, which is the reference design of the board made by the Xilinx Vitis AI platform. The base overlay is made up of three files: the bitstream file, the Vivado design file and the bit file that provides the python wrappers. They are made during the DPU generation and there are a specific trio of files for any DPUs. Other than configuration files, also the neural networks obtained from the compilation process must also be uploaded on the Zynq board. The neural models are in the Xilinx format after the compilation, which is a single graph file with extension xmodel.

The communication with the Xilinx board takes place via the ethernet cable, through which it is connected to the server. Once a connection with the development server has been established, the board can be reached with the internet browser. The Zynq operating system and the PYNQ application expose a Jupyter Notebook environment thanks to which it is possible to control the device. Therefore, the uploading operations of the files to the device, viewing and executing Python source code and executing bash commands via a linux shell are possible thanks to the Jupyter Notebook environment.

On board of the Zynq UltraScale+ MPSoC device a Python script was executed in order to test all the mobilenets with all the DPU architectures. The software we created accepts as input the hyper parameters of the mobilenet model that must be tested and the name of the DPU on which you want to test. Then the script performs the latency test by submitting 200 images of the imagenet validation dataset to the neural network and calculates the preprocessing time, the inference time and the total time of both the operations. The tests were performed with 200 images because these guarantee an accuracy of 99% on the measurement of the inference time, of 95% on the measurement of the preprocessing time and of 96% on the total time. The confidence interval was evaluated with a confidence level of 99%.

3.5.1 CPU and FPGA workload

It is important to note that not all the code executed on the Zynq device is executed on the FPGA areas. All the normal Python code is executed on the quad-core Arm Cortex-A53, instead only the hardware accelerated code is executed on the programmable logic. The software we have deployed on the Zynq board executes the preprocessing of the images and the computation of the timing measurement on the CPU, instead the Mobilenet feed forward pass is executed on the FPGA specially configured by the selected DPU.

Furthermore, it should be noted that all the computations performed on the CPU can

be performed in 32-bit floating point, while only the operations performed on the FPGA need to be performed in 8-bit fixed point format. For this reason, the Arm Cortex-A53 processor is also in charge of converting the incoming and outgoing data to the neural network.

By analyzing the graph of the neural networks in the Xilinx format, it is possible to see the exact position of the nodes where the conversion of the data from floating point to integer is done. As you can see in the figure 3.5 these nodes are called upload and download and they refer to the upload operation of the data on the FPGA and to the download operation of the result of the inference on the CPU.

Finally, as can seen from the figure 3.5 the softmax node is not executed on the FPGA and for this reason this node has been implemented by us in Python code within the testing script and it is also executed on the CPU.

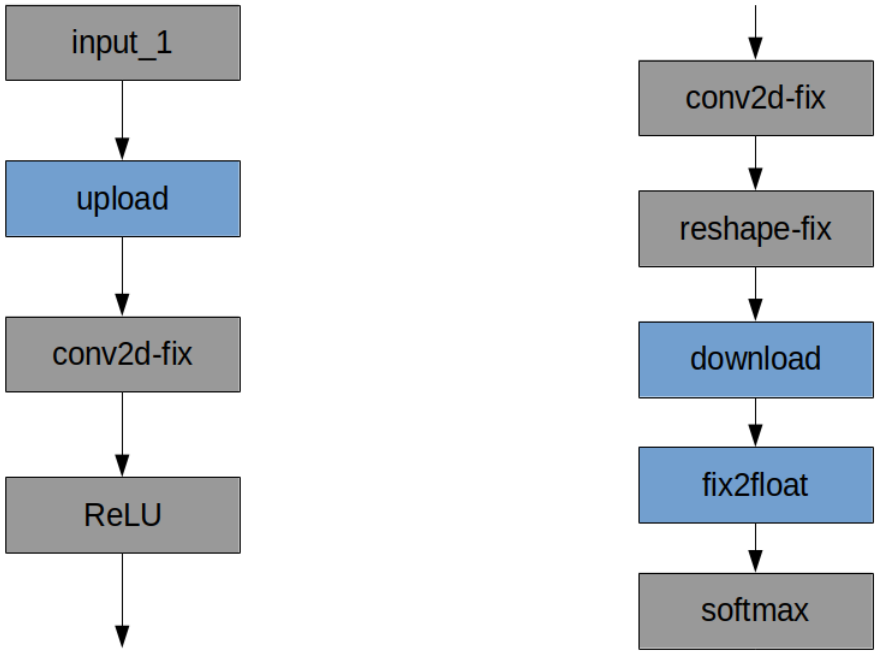


Figure 3.5: Input and output section of the Mobilenet 1.0 224 model graph

Chapter 4

Experimental Results

4.1 Methodology

The images used for the quantization and validation of the quantized models were taken from the ILSVRC 2012 imagenet validation dataset. Starting from this dataset, the quantization dataset was generated by shuffling the order of the original images (seed 42) and then extracting the first 4096 images. Instead, the validation of the quantized models was done on the entire validation dataset of imagenet ILSVRC 2012 which contains 50,000 images. All the operations listed above were performed by the tf2_makeDatasets.py script which saved the result inside the TFrecord data structures. This script was run inside a virtual environment containing Python version 3.7.5 The quantization, validation and compilation of the models are instead performed by the Python script tf2_vai_flow2.py executed within the docker environment of Vitis AI version 1.3.2. The hardware design of the DPUs was performed within the xilinx/vitis-ai docker:1.3.411 in conjunction with the Xilinx Vitis Unified Software Platform 2020.2 development environment and the Xilinx Runtime Library (XRT) 2020.2. The electronic board used is the Xilinx Zynq UltraScale+ EK-U1-ZCU104-G on which PYNQ v2.6 has been installed. With this configuration, the version of Python available on the board is 3.6.5, thanks to which the test_mobilenet.py script was executed to perform the latency tests of the mobilenet v1. Latency tests were done with 200 images from the ILSVRC 2012 imagenet validation dataset.

4.2 Results

Here will be analyzed a set of graphs, each for any DPU architectures. Any graph has a Pareto frontier in order to focus the attention on the efficient choices. The first two will be the best and the worst DPU architectures in terms of hardware resources availability, that will be analyzed in detail, then the remaining architectures will be presented. In all the following graphs the conventional measure of accuracy is shown which is called Top-1 accuracy, with this measurement the response expected from the model must be exactly the one with the highest probability value. Sometimes references are made to the Top-5 accuracy, which instead refers to a measurement that considers the network response correct if the expected response is present in the five classes with greater probability.

Furthermore, the number of frames per second indicated in the graphs is calculated from the total of the preprocessing and inference times.

On one hand the results show a high throughput, which is compatible with real-time applications, even on the smallest available FPGA architecture. On the other hand, all models suffer a large accuracy reduction, if the proposed post-training quantization technique is used. Although, taking into account, as an upper-bound, the achievable accuracy using a quantization aware-training technique, the results show that the deployment of deep neural networks on FPGA is a viable option.

Finally, it is important to notice that the throughput results expressed in FPS are taken from the measurements done on the Zynq UltraScale+ MPSoC within the testing script proposed by us. Instead the showed accuracy results are an achievable upper-bound obtained by Tensorflow developers thanks to the quantization aware-training technique [23] on which the Vitis AI quantizer is based.

4.2.1 B4096 Architecture

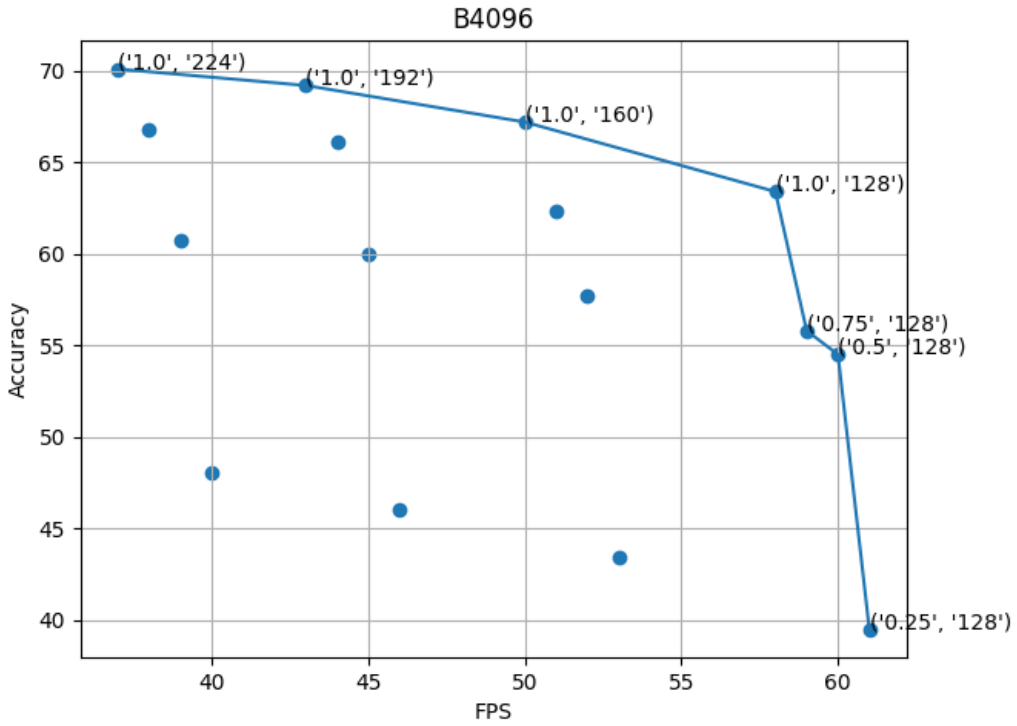


Figure 4.1: B4096 is the best performing DPU architecture

Figure 4.1 shows the results obtained with the best performing DPU architecture in terms of number of operations per clock. As you can see from the graph, the four neural networks with the alpha parameter equal to 1 are the choice to be made if you

want to maximize accuracy. As you can see in the figure 4.4, this feature is shared with all architectures except the B512. It is interesting to note that with the performances offered by this DPU, even the most accurate network (70.1 % top-1 and 88.9 % top-5), and therefore also the most demanding of resources, is able to exceed the threshold of 35 FPS considering both preprocessing and inference. This result guarantees the possibility of using this architecture with the most accurate model even in a real-time environment. In addition, it should be noted that the fastest network on each architecture, which is Mobilenet 0.25 128, on the B4096 is capable of reaching a throughput of 61 FPS. This is the maximum result obtained in terms of speed, taking into account the total time including preprocessing and inference.

4.2.2 B512 Architecture

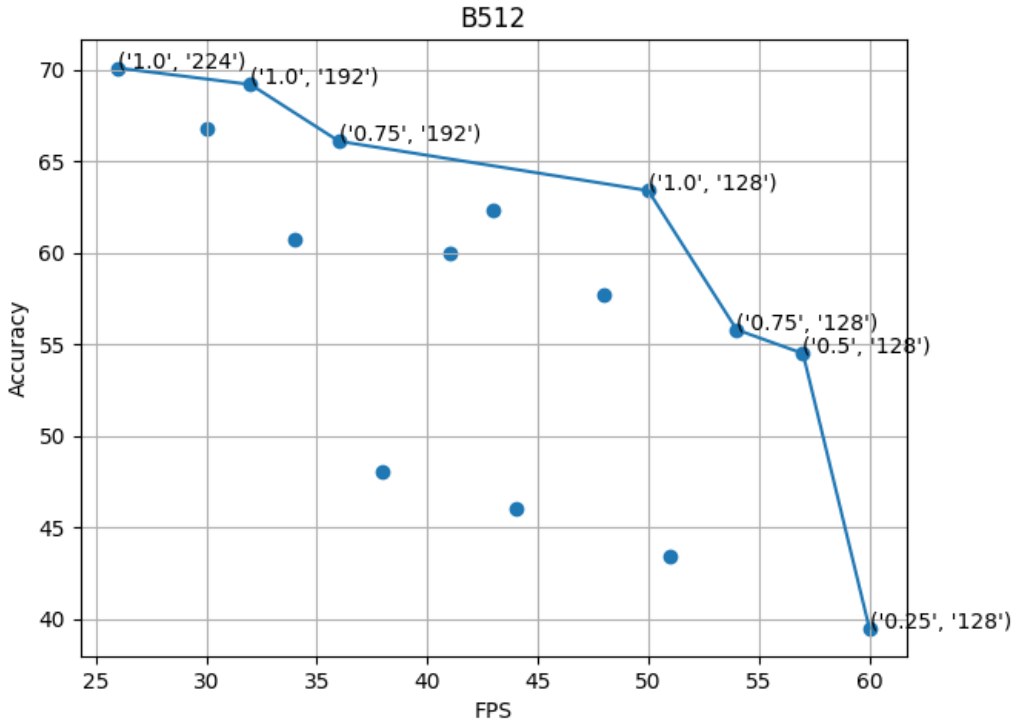


Figure 4.2: B512 is the DPU architecture with less resources

After the best performing DPU it is interesting to analyze the smallest one, the so-called B512 architecture. As can be seen in figure 4.2, the fastest model and the more accurate one remains the same as on the B4096 architecture. With the B512 architecture, however, the most accurate network does not allow to obtain a minimum of 30 FPS, therefore to exceed this threshold you have to sacrifice a bit of accuracy by reducing the size of the input image. It is interesting to notice the best throughput, that is equal

to 60 FPS, because it is not far from the result obtained on the B4096, therefore if the hardware resources optimization is taken into account, the best choice is the B512 architecture. Finally, this architecture is the only one that does not have all the networks on the Pareto frontier in common with the other architectures. Specifically, Mobilenet 1.0 160 is replaced by Mobilenet 0.75 192.

4.2.3 Other Architectures

Analyzing the results obtained on the remaining architectures, figures 4.4, it is observed that, on the one hand, considering the lightest and fastest network, the Mobilenet 0.25 128, the performance in terms of latency does not vary much between one architecture and one other. On the other hand, if we consider the slower, but more accurate model, the Mobilenet 1.0 224, it can be seen in table 4.1 that the latency varies significantly with the architecture. Therefore, if execution speed and resource savings is prefered, the B512 architecture is the best choice from every point of view, if instead accuracy is the primary target, the choice of architecture must be chosen based on the latency requirement.

	B512	B800	B1024	B1152	B1600	B2304	3136	B4096
Min FPS	26.28	29.08	30.95	27.05	34.08	35.32	36.17	37.64
Max FPS	60.74	60.88	61.34	61.13	61.34	61.38	61.45	61.45

Table 4.1

4.2.4 Lookup Table Analysis

Figure 4.3 shows the number of FPS in relation to the number of lookup tables available to each DPU. For this reason, each curve is made up of eight points, each of which represents the number of lookup tables available for the given architecture. For practical reasons, the performances of four different models are shown, obtained by varying the width multiplier parameter. The figure 4.3 shows an interesting fact; switching between DPU architectures leads to a different increase, in terms of FPS, depending on the alpha parameter of the model. Lower is the alpha parameter, lower is the FPS increase by changing the architecture; Higher is the alpha parameter, higher is the FPS increase by changing the architecture.

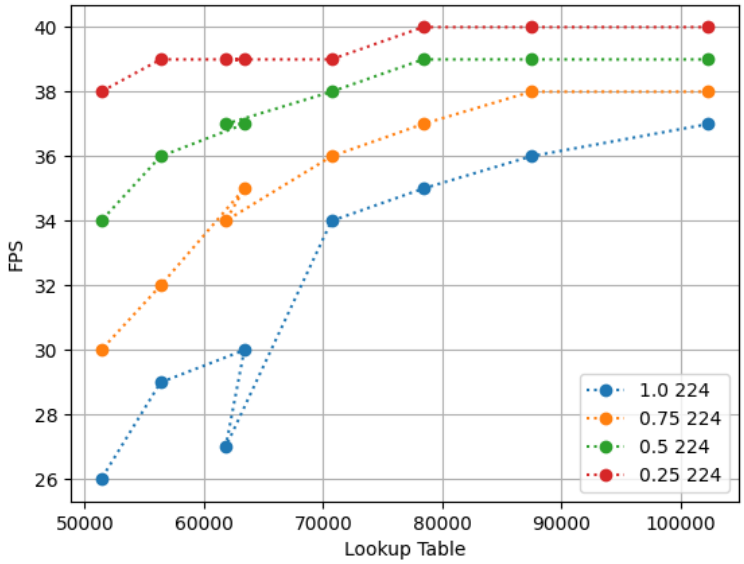
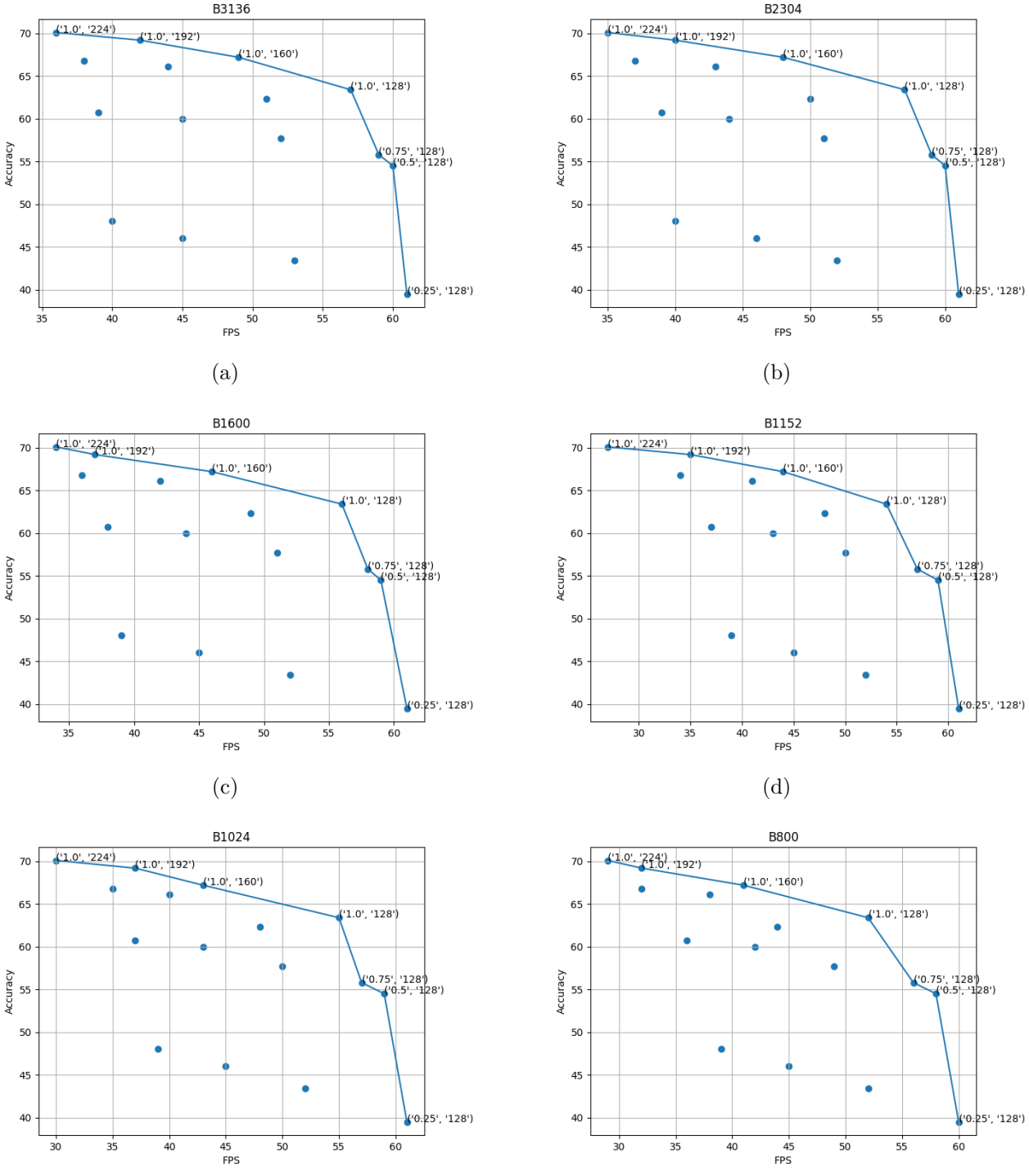


Figure 4.3: The graph shows the variation of FPS obtained by varying the number of LUTs available on the DPU

4.3 Rules of thumb

Here are some general considerations regarding the outcomes deriving from the analysis of all the graphs and numerical outputs, that can summarize the results obtained and provide a quick starting point for the choice of the most suitable networks and FPGA configurations in function of the desired system:

- If the goal is to have the highest accuracy, it is necessary to keep the width multiplier equal to one and select how many FPS to obtain thanks to the size of the input image. By varying the image size parameter, a very high increase in inference speed is obtained at the expense of a small loss of accuracy.
- If the aim is to exceed the threshold of 50 FPS it is necessary to lower the width multiplier, thanks to which it is possible to exceed the threshold of 60 FPS. However, this increase in preprocessing and inference speed leads to a notable decrease in accuracy.
- Speaking of architectures, choosing a DPU with a high number of resources is advantageous only if you want to use a high width multiplier parameter, instead with a neuron number decreasing there is no saving by using a DPU with a high number of resources, in these conditions it is better to opt for the architecture that uses the least number of possible resources.



(a) (b) (c) (d) (e) (f)

Figure 4.4

Chapter 5

Conclusions

In the research carried out during this thesis, we implemented and tested a deployment flow for hardware accelerated deep neural networks on FPGA. The proposed development flow that deals with model preparation, programmable logic configuration and testing, allowed us to execute different Mobilenets topologies on the FPGA on board of the Zynq UltraScale+ MPSoC.

The accelerated implementation of the Mobilenets on the FPGA has got excellent performance on all the DPU configurations. The fastest model has a throughput of 61 FPS on the best performing DPU and 60 FPS on the smallest DPU configuration, with a top-1 accuracy of 39.5%. Instead, the more accurate model, which is also the slowest, has a throughput of 38 FPS on the fastest DPU and 26 FPS on the smallest one, with a top-1 accuracy of 70.1%. The results show a high throughput, which is compatible with real-time edge applications, even on the smallest available FPGA architecture. Therefore, it can be said that the deployment of deep neural networks on FPGA is an excellent design choice, although it is necessary to be very careful in the quantization phase to avoid excessive accuracy reduction of the models.

Bibliography

[1] Ahmed Jawad A AlBdairi et al. «Face Recognition Based on Deep Learning and FPGA for Ethnicity Identification». In: *Applied Sciences* 12.5 (2022), p. 2605.

[2] Marco V Barbera et al. «To offload or not to offload? the bandwidth and energy costs of mobile cloud computing». In: *2013 Proceedings Ieee Infocom*. IEEE. 2013, pp. 1285–1293.

[3] Junyi Chai et al. «Deep learning in computer vision: A critical review of emerging techniques and application scenarios». In: *Machine Learning with Applications* 6 (2021), p. 100134.

[4] Christiam F Frasser et al. «Using Stochastic Computing for Virtual Screening Acceleration». In: *Electronics* 10.23 (2021), p. 2981.

[5] Rania O Hassan and Hassan Mostafa. «Implementation of deep neural networks on FPGA-CPU platform using Xilinx SDSOC». In: *Analog Integrated Circuits and Signal Processing* 106.2 (2021), pp. 399–408.

[6] Andrew G Howard et al. «Mobilenets: Efficient convolutional neural networks for mobile vision applications». In: *arXiv preprint arXiv:1704.04861* (2017).

[7] Wenlu Hu et al. «Quantifying the impact of edge computing on mobile applications». In: *Proceedings of the 7th ACM SIGOPS Asia-Pacific workshop on systems*. 2016, pp. 1–8.

[8] Xilinx Inc. *DPUCZDX8G for Zynq UltraScale+ MPSoCs*. Version PG338 (v3.4). Jan. 2022.

[9] Xilinx Inc. *UltraRAM: Breakthrough Embedded Memory Integration on UltraScale+ Devices*. Version WP477 (v1.0). June 2016.

[10] Xilinx Inc. *Vitis AI User Guide*. Version UG1414 (v2.0). Jan. 2022.

[11] Xilinx Inc. *ZCU104 Evaluation Board*. Version UG1267 (v1.1). Oct. 2018.

[12] Xilinx Inc. *Zynq UltraScale+ MPSoC Software Developer Guide*. Version UG1137 (v2021.2). Oct. 2021.

[13] Xilinx Inc. *Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit*. URL: https://www.xilinx.com/products/boards-and-kits/zcu104.html#hardware.

[14] Yiping Kang et al. «Neurosurgeon: Collaborative intelligence between the cloud and mobile edge». In: *ACM SIGARCH Computer Architecture News* 45.1 (2017), pp. 615–629.

[15] Keras-Team. *Imagenet Utils*. URL: https://github.com/keras-team/keras/blob/b80dd12da9c0bc3f569eca3455e77762cf2ee8ef/keras/applications/imagenet_utils.py#L250.

[16] Hakima Khelifi et al. «Bringing deep learning at the edge of information-centric internet of things». In: *IEEE Communications Letters* 23.1 (2018), pp. 52–55.

[17] Ivano Lauriola, Alberto Lavelli, and Fabio Aiolli. «An introduction to deep learning in natural language processing: Models, techniques, and tools». In: *Neurocomputing* 470 (2022), pp. 443–456.

[18] Burhan A Mudassar, Jong Hwan Ko, and Saibal Mukhopadhyay. «Edge-cloud collaborative processing for intelligent internet of things: A case study on smart surveillance». In: *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE. 2018, pp. 1–6.

[19] Manas Sahni. *Making Neural Nets Work With Low Precision*. URL: https://sahnimanas.github.io/post/quantization-in-tflite/.

[20] Ahmad Shawahna, Sadiq M Sait, and Aiman El-Maleh. «FPGA-based accelerators of deep learning networks for learning and classification: A review». In: *ieee Access* 7 (2018), pp. 7823–7859.

[21] Weisong Shi et al. «Edge computing: Vision and challenges». In: *IEEE internet of things journal* 3.5 (2016), pp. 637–646.

[22] Shaden Smith et al. «Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model». In: *arXiv preprint arXiv:2201.11990* (2022).

[23] Tensorflow. *MobileNetV1*. URL: https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md.

[24] Tensorflow. *Sparse Categorical Accuracy*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseCategoricalAccuracy.

[25] Tensorflow. *TensorFlow Lite 8-bit quantization specification*. URL: https://www.tensorflow.org/lite/performance/quantization_spec.

[26] Lionel Sujay Vailshery. *Number of IoT connected devices worldwide 2019-2021, with forecasts to 2030*. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[27] Teng Wang et al. «A survey of FPGA based deep learning accelerators: Challenges and opportunities». In: *arXiv preprint arXiv:1901.04988* (2018).

[28] Xiaofei Wang et al. «Convergence of edge computing and deep learning: A comprehensive survey». In: *IEEE Communications Surveys & Tutorials* 22.2 (2020), pp. 869–904.