

ACCELERATING DEEP NETWORK-BASED IMAGE ANALYSIS USING RE-CONFIGURABLE ARCHITECTURE

*A thesis submitted in partial fulfilment of the requirement for the award of the
degree of Master of Technologies in VLSI Design & Micro Electronics Technology*

By

RANITA PAL

Class Roll Number : 002010703004

Examination Number : M6VLS23008

Registration Number : 154106 of 2020-2021

Under the Guidance of

Dr. Sayan Chatterjee

(Professor, Jadavpur University, Kolkata)

&

Prof. Amlan Chakrabarti

(Professor and Director, A.K. Choudhury School of IT, University of Calcutta)

Jadavpur University

June 2023

**FACULTY OF ENGINEERING & TECHNOLOGY
JADAVPUR UNIVERSITY**

CERTIFICATE OF RECOMMENDATION

This is to certify that the thesis entitled — “ACCELERATING DEEP NET WORK-BASED IMAGE ANALYSIS USING RE-CONFIGURABLE ARCHITECTURE” has been carried out by **Ranita Pal** bearing Class Roll No: 002010703004, Examination Roll No.: and Registration No: 154092 of 2020-21, under my guidance and supervision and be accepted in fulfilment of the requirement for the degree of “ **Master of Technology in VLSI Design & MicroElectronics Technology**” in the Department of Electronics and Telecommunication Engineering”. No prior submission of the research results in this thesis for any degree at any other institution or university.

Sayan Chatterjee 12/6/2023

**PROF. SAYAN
CHATTERJEE**

(Internal Thesis Supervisor,
Dept. of Electronics &
Telecommunication
Engineering, Jadavpur
University)

Dr. Sayan Chatterjee
Professor
Electronics & Telecomm. Engg. Dept.,
Jadavpur University, Kolkata - 700032.

Amlan Choudhury 12/06/2023

PROF. AMLAN CHAKRABARTI
(External Thesis Supervisor,
Professor and Director, A.K.
Choudhury School of IT, Uni-
versity of Calcutta)

**Director
A. K. Choudhury
School of IT
University of Calcutta**

Manotosh Biswas 12/06/23

DR. MANOTOSH BISWAS

(Head of Department,
Dept. of Electronics &
Telecommunication
Engineering, Jadavpur
University)

MANOTOSH BISWAS

Professor and Head
Electronics and Telecommunication Engineering
Jadavpur University, Kolkata - 32

Ardhendu Ghosal 12/06/23

**PROF. ARDHENU
GHOSAL**

(Dean,
Faculty of Engineering & Tech-
nology, Jadavpur University)



DEAN
Faculty of Engineering & Technology
JADAVPUR UNIVERSITY
KOLKATA-700 032

**FACULTY OF ENGINEERING & TECHNOLOGY
JADAVPUR UNIVERSITY**

CERTIFICATE OF APPROVAL

The forgoing thesis titled “**ACCELERATING DEEP NETWORK-BASED IMAGE ANALYSIS USING RE-CONFIGURABLE ARCHITECTURE**” is here approved as a creditworthy study of an engineering subject conducted and presented satisfactorily to warrant its acceptance as a pre-requisite to the degree for which it was submitted. It is understood that the undersigned does not automatically support or accept any argument made, opinion expressed, or inference is drawn in it by this approval, but only approve the thesis for the reason for which it was submitted.

**Committee on Final Examination for
Evaluation of the Thesis**

Signature of Internal Supervisor

Signature of External Supervisor

Signature of External Examiner1

Signature of External Examiner2

Author's declaration of originality

I, **RANITA PAL**, hereby declare that the work presented in the thesis is entirely my original work. I have appropriately acknowledged and referenced any sources utilized in the creation of this work. No part of this work has been previously submitted for academic assessment or publication, except where specifically referenced. I affirm that I have not engaged in any form of plagiarism, and I accept full responsibility for the content and integrity of this work.

By signing this declaration, I affirm that the above statements are true and that I have complied with the ethical standards and requirements set forth by the academic institution and the relevant guidelines for originality and integrity in research.

Name: **Ranita Pal**

University Registration No.: **154106 of 2020-2021**

Exam Roll No: **M6VLS23008**

Class Roll No.: **002010703004**

Thesis Title: **ACCELERATING DEEP NETWORK-BASED IMAGE ANALYSIS USING RE-CONFIGURABLE ARCHITECTURE**

Signature of Candidate:

Date:

Acknowledgements

I would like to begin by expressing my heartfelt gratitude to my supervisors, **Prof. Amlan Chakraborti** and **Dr. Sayan Chatterjee** for their guidance, expertise, and unwavering support in shaping this thesis. I have been very fortunate to have a guide like them. Their positivity, confidence, and ideas help me to complete my thesis, and they guide me as a guardian.

I extend my sincere thanks to **Dr. Manotosh Biswas**, the Head of the Department for consistently extending a helping hand whenever needed. I also thank my fellow project mates, friends, and technical and non-technical staff of Jadavpur University who have helped me directly or indirectly during the tenure of my thesis work.

I want to express my gratitude to my parents and family also, for their invaluable love and encouragement.

Ranita Pal

Jadavpur University

Kolkata-32, West Bengal

Abstract

This thesis investigates the utilization of reconfigurable architectures, specifically field-programmable gate arrays (FPGAs), for accelerating deep neural network-based image classification tasks. Four different DNN classification model architectures are analyzed using a custom medical image dataset. The trained model weights are compiled into an hardware-specific model weights and executed on an FPGA board, taking advantage of the parallel processing capabilities and hardware acceleration offered by the DPU architecture. The evaluation of performance and efficiency gains encompasses factors such as inference time, accuracy, and resource utilization. The experimental results demonstrate that the integration of software and hardware components leads to significant improvements in speed and energy efficiency. However, certain limitations and areas for improvement are identified, highlighting future research opportunities for optimizing reconfigurable architectures for image analysis tasks. In summary, this thesis presents a comprehensive exploration of the advantages and challenges associated with reconfigurable architectures for accelerating deep neural network-based image analysis. The findings provide valuable insights into the potential of FPGAs and DPUs in enhancing the performance of image analysis tasks based on deep neural networks and establish a foundation for future advancements in this field.

List of Acronyms

FPGA	Field Programmable Gate Array
DPU	Xilinx® Deep Learning Processing Unit
CPU	Central Processing Unit
SoC	System on Chip
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
IP	semiconductor Intellectual Property
AI	Artificial Intelligence
DL	Deep Learning
AXI	Advanced eXtensible Interface
TCL	Transaction Control Language
XSA	Xilinx Support Archive
API	Application Programmable Interface
DDR	Double Data Rate
HDL	Hardware Description Language
DTG	Device Tree Generator
mAp	Mean Average Precision
V.P.	Viral Pneumonia

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Objective	3
1.3	Contribution of Hypothesis	3
1.4	Thesis organization	4
2	Thesis Background	6
3	Initial Experiments and Findings	9
3.1	Vitis	9
3.2	Vitis AI	10
3.3	Hardware Core	11
3.3.1	DPU architecture	11
3.3.2	DPU Naming	13
3.3.3	DPU Configuration	13
3.3.4	DPU Integration	16
4	Tools and Prototyping Environment Utilized	19
4.1	Xilinx	19
4.1.1	Vitis AI	19
4.1.2	Vitis AI DPU	20
4.1.3	Vitis AI Quantiser	21
4.1.4	Vitis AI Compiler	23
4.1.5	ZCU104	25
4.2	User Hardware	26

4.3	User Software	26
4.3.1	Docker	26
4.3.2	Jupyter Notebook	27
5	Model Comparison	28
5.1	RESNET-50	30
5.1.1	Description	30
5.1.2	Reason for choosing the model	30
5.1.3	Training/Fine-tuning the model	31
5.1.4	Quantization	32
5.1.5	Compiling quantised model to xmodel	34
5.2	DENSENET-169	35
5.2.1	Description	35
5.2.2	Reason for choosing the model	35
5.2.3	Training/Fine-tuning the model	36
5.2.4	Quantization	37
5.2.5	Compiling quantised model to xmodel	38
5.3	VGG-16	39
5.3.1	Description	39
5.3.2	Reason for choosing the model	39
5.3.3	Training/Fine-tuning the model	40
5.3.4	Quantization	41
5.3.5	Compiling quantised model to xmodel	42
5.4	MobileNet	43
5.4.1	Description	43
5.4.2	Reason for choosing the model	43
5.4.3	Training/Fine-tuning the model	44
5.4.4	Quantization	45
6	Description of Workflow	47
6.1	Tools Execution Process	47
6.1.1	Vivado 2020.2 HLx edition	47

6.1.2	Vitis AI	48
6.1.3	Petalinux	48
6.1.4	Edge AI environment on the ZCU104 board .	48
6.2	Integration of fine-tuned models with ZCU104	49
6.2.1	Hardware	49
6.2.2	Software	54
6.2.3	Prepare Files for Platform Packaging	58
6.3	Application on DPU	64
6.3.1	Docker Setup on the Host	65
6.3.2	Model Deployment	65
6.3.3	Deployment on Edge boards	66
7	Result Analysis	70
7.1	Dataset	71
7.2	Proof of Concept and Accuracy Analysis	74
7.3	Model Size and Resource Utilization	77
7.4	Hardware Acceleration:	
	Performance & Evaluation	78
8	Conclusions	82
9	Future Work	83
10	Summary	84
	Bibliography	85

List of Figures

3.1	<i>DPU architecture</i>	12
3.2	<i>DPU Naming</i>	13
3.3	<i>DPUCZDX8G architecture</i>	14
3.4	<i>DPU IP with all its interfaces</i>	18
4.1	<i>Vitis AI Quantizer Framework</i>	21
4.2	<i>Vitis AI Quantizer Workflow</i>	22
4.3	<i>Vitis AI Compiler Framework</i>	23
4.4	<i>Vitis AI Compiler Workflow</i>	24
4.5	<i>Xilinx ZCU104 Evaluation Board and Peripheral Connections</i>	25
5.1	<i>Confusion Matrix for ResNet-50 model</i>	31
5.2	<i>Confusion Matrix for DenseNet-169 model</i>	36
5.3	<i>Confusion Matrix for VGG-16 model</i>	40
5.4	<i>Confusion Matrix for MobileNet model</i>	44
6.1	<i>Simplified view of development flow in proposed solution</i>	50
6.2	<i>Screenshot of DPU configuration used in this project</i>	52
6.3	<i>Screenshot of a block diagram of hardware configuration used in this project</i>	53
6.4	<i>Additional User Packages in PetaLinux</i>	55
6.5	<i>Petalinux Root filesystem configuration window</i>	56
6.6	<i>Device Tree for ZCU104</i>	57
6.7	<i>Petalinux configuration window</i>	57
6.8	<i>Petalinux Root filesystem configuration formats window</i>	58

6.9	<i>Contents of linux.bif</i>	59
6.10	<i>Screenshot of Putty Configuration Window</i>	63
6.11	<i>Screenshot of terminal shortly after Petalinux has suc- cessfully booted greeting with a login screen.</i>	63
6.12	<i>creenshot of DPU Signature Viewed ZCU104 with DEx- plorer</i>	64
6.13	<i>Screenshot of Quantized model file</i>	66
6.14	<i>Screenshot of Compiled xmodel file</i>	67
7.1	<i>Screenshot of COVID dataset image</i>	71
7.2	<i>Screenshot of COVID class prediction</i>	72
7.3	<i>Screenshot of Normal class dataset image</i>	72
7.4	<i>Screenshot of Normal class prediction</i>	73
7.5	<i>Screenshot of Viral Pneumonia class dataset image</i>	73
7.6	<i>Screenshot of Viral Pneumonia class prediction</i>	74

List of Tables

3.1	<i>DPU parameters in Vivado</i>	16
5.1	<i>Classification Report of ResNet-50 model</i>	32
5.2	<i>Confusion Matrix for Quantized ResNet-50 model . .</i>	33
5.3	<i>Classification Report of Quantized ResNet-50 model .</i>	33
5.4	<i>Confusion Matrix for Compiled ResNet-50 model . .</i>	34
5.5	<i>Classification Report of Compiled Resnet-50 model . .</i>	34
5.6	<i>Classification Report of DenseNet-169 Model</i>	37
5.7	<i>Confusion Matrix for Quantized DenseNet model . .</i>	37
5.8	<i>Classification Report of DenseNet-169 Quantized Model</i>	38
5.9	<i>Confusion Matrix for Compiled DenseNet-169 model</i>	38
5.10	<i>Classification Report of Compiled DenseNet-169 model</i>	39
5.11	<i>Classification Report of Vgg-16 model</i>	41
5.12	<i>Confusion Matrix for Quantized Vgg-16 model</i>	41
5.13	<i>Classification Report of Quantized Vgg-16 Model . . .</i>	42
5.14	<i>Confusion Matrix for Compiled Vgg-16 model</i>	42
5.15	<i>Classification Report of Compiled Vgg-16 model . . .</i>	43
5.16	<i>Classification Report of MobileNet</i>	45
7.1	<i>Comparison of average accuracy values per class for ResNet-50 model</i>	75
7.2	<i>Comparison of average accuracy values per class for DenseNet model</i>	75
7.3	<i>Comparison of average accuracy values per class for Vgg-16 model</i>	76

7.4	<i>Comparison of average accuracy values per class for MobileNet</i>	76
7.5	<i>Comparison of Model Sizes and Resource Utilization .</i>	77
7.6	<i>Inference Time Comparison between CPU and DPU .</i>	79

Chapter 1

Introduction

In recent years, deep neural networks (DNNs) have revolutionized the field of image analysis, enabling remarkable advancements in tasks such as image classification [1], object detection [2], and semantic segmentation. However, as DNNs continue to grow in complexity and demand higher computational resources, the need for efficient and high-performance hardware accelerators becomes crucial [3]. Re-configurable architectures, such as field-programmable gate arrays (FPGAs), offer a promising solution for accelerating DNN computations due to their flexibility, parallel processing capabilities, and energy efficiency [4].

This thesis explores the acceleration of a few different deep neural network-based image classification architectures through the integration of software and re-configurable hardware architecture. Specifically, the focus lies on leveraging the capabilities of the ZCU104 FPGA board and the development and optimization of a deep learning processing unit (DPU) architecture to enhance the performance and efficiency of image analysis tasks.

1.1 Motivation

Running artificial neural networks on different types of hardware-based accelerators is a topic extensively discussed for many years. Many educational institutions across the world use various development boards as teaching aids and for quick prototyping. The motivation behind this work lies in the demand for real-time image classification and the potential of FPGA-based implementations to achieve high-performance inference in resource-constrained environments [5]. Furthermore, hardware acceleration enables scalability and flexibility. With the ability to scale up the number of accelerator units or integrate multiple accelerators, deep learning models can be trained and deployed on large-scale systems or distributed environments [6]. This scalability is particularly crucial for handling massive datasets and complex network architectures.

FPGA DNN accelerators [7] have become relatively new, offering low-latency, energy-efficient, and customizable solutions for accelerating deep neural network computations in various applications. The current research on implementing DPUs on FPGAs is relatively limited compared to other areas focusing on the given algorithms. However, there is vast untapped potential for further research, particularly in implementing image classification applications on student-friendly platforms. By utilizing different deep learning models on the ZCU104 evaluation kit, educational purposes can be served effectively. This proposed solution holds the promise of enhancing students' understanding of how artificial neural networks can be accelerated on FPGAs, facilitating the acceleration of various applications on hardware platforms.

1.2 Objective

This thesis aims to design architectures for important deep neural networks used in image analysis, primarily focusing on the software domain. The designed architectures will be optimized and fine-tuned for specific image analysis tasks. Subsequently, the implementation phase will involve leveraging accelerated DPU IP on hardware platforms. This entails configuring the DPU architecture using tools like Vitis IDE and integrating it with the FPGA board. The ultimate goal is to achieve high-performance and energy-efficient image classification by executing the optimized deep neural networks on the hardware platform, harnessing the capabilities of the DPU for accelerated inference.

1.3 Contribution of Hypothesis

The main idea for the proposed solution was that Vitis AI could be used to quickly deploy neural network applications that were accelerated by DPU without modifying hardware architectures. Four alternative deep neural network models will be tested on the ZCU104 hardware platform in order to demonstrate its usability and simplicity. If Vitis AI performs as expected, it will make accelerating deep neural network inference easier than ever before and enable quick prototyping of new embedded solutions requiring real-time medical image processing. This approach holds the potential to simplify and expedite the development of advanced embedded systems in the medical domain.

1.4 Thesis organization

This thesis is formulated in the following chapters:

- The **Thesis Background** chapter provides a concise overview of previously proposed solutions that are relevant to the current problem. It includes a list of similar solutions along with a brief analysis from the author regarding the feasibility of utilizing these solutions to address the problem at hand. This section offers a valuable glimpse into the existing work that has been done in the field, providing context and highlighting the advancements made prior to the current study.
- The **Initial Experimental and Findings** chapter presents the workflow and initial experimental phase undertaken in the thesis before arriving at the final solution. It outlines the various tools that were initially intended for use in finding the solution but were ultimately disregarded, accompanied by the reasons for their exclusion. This chapter sheds light on the early stages of the research process, providing insight into the methodologies explored and the reasons behind the choices made in the pursuit of the ultimate solution.
- The **Used Tools** chapter comprehensively documents and explains the key tools employed in the development of the solution. It provides a detailed inventory of the essential software, hardware, and technologies utilized throughout the project, accompanied by thorough explanations of their functionalities and contributions. This chapter serves as a comprehensive reference, offering readers a comprehensive understanding of the tools harnessed to implement the solution effectively.
- The **Model Comparison** chapter conducts a comprehensive evaluation of four distinct models, considering various factors

such as time, performance, and model size, for image analysis in both software and hardware contexts. Through a meticulous analysis, this chapter provides a comparative assessment of the models, shedding light on their respective strengths and weaknesses. By examining their performance metrics and considering their computational efficiency, this chapter enables a thorough understanding of the trade-offs involved in selecting a suitable model for image analysis tasks.

- The **Description of Workflow** chapter serves as a comprehensive resource for individuals seeking to replicate the provided solution and implement various deep neural networks on the FPGA platform, specifically the ZCU104 Evaluation board. It offers detailed instructions and necessary tools to enable users to successfully replicate and implement the solution.
- The **Result Analysis** section digs into the methodology used to analyze the proposed solution and presents the experimental findings obtained.
- The **Conclusion** section summarizes the key findings and offers a concise overview of the main analysis results.
- The **Future work** section highlights potential areas for improvement and suggests possible enhancements to the proposed solution.

Chapter 2

Thesis Background

This chapter provides an overview of earlier attempts to run deep neural networks on similar platforms. The objective was to identify existing solutions that address similar problem descriptions. Among these attempts, a particularly close match was found, where a Convolutional Neural Network (CNN) was implemented on the ZCU104 board using DPU IP [8]. This implementation aimed to leverage custom hardware platforms to accelerate the inference process and reduce overall energy consumption. In this chapter, we dig into the details of the closest solution and examine its relevance to the current problem.

Numerous studies have been conducted utilizing the DPU on FPGA platforms, particularly on Zynq devices. These works explore the potential of leveraging the DPU’s capabilities to accelerate various computational tasks. The utilization of DPU on FPGA has emerged as a popular research area, showcasing its effectiveness and versatility in accelerating neural network computations and enabling efficient hardware acceleration. For example, in the conference paper [9], the work focuses on achieving time-predictable execution of Deep Neural Networks (DNNs) accelerated on FPGA System-on-Chips (SoCs). The study utilizes the modern DPU accelerator by Xilinx and conducts extensive profiling on the Zynq Ultrascale+ platform.

Based on the profiling data, an execution model is proposed to derive response-time analysis. To enhance predictability, a custom FPGA module named DICTAT is introduced. Experimental results, including analytical bounds and measurements, demonstrate the effectiveness and performance of the approach for Advanced Driver Assistance Systems (ADAS) applications.

In the research paper **"FPGA-Based Adaptive Hardware Acceleration for Multiple Deep Learning Tasks"** [10] by the authors Yufan Lu; Xiaojun Zhai; Sangeet Saha; Shoaib Ehsan; and Klaus D. McDonald-Maier from School of Computer Science and Electronic Engineering, University of Essex, Colchester, UK addresses the challenges of deploying deep learning algorithms on resource-constrained mobile and embedded systems. The authors propose an adaptive hardware resource management system implemented on FPGAs, which dynamically manages on-chip hardware resources to accommodate various tasks. By utilizing dynamic function exchange (DFX) technology, the system allocates hardware resources for deploying deep learning units (DPUs), optimizing performance and power consumption. The prototype is implemented on Xilinx Zynq UltraScale+ series chips, and experimental results demonstrate significant improvements in computing efficiency for resource-constrained systems across different scenarios. The proposed scheme achieves power savings of 38% and 82% in low and high working load cases, respectively, resulting in an energy reduction of approximately 75.8% compared to the baseline approach.

In the article **"An Effective Design to Improve the Efficiency of DPUs on FPGA"** [11] by Yutian Lei, Qingyong Deng, Saiqin Long and Shaohui Liu; Sangyoon Oh, there was a similar attempt to accelerate multitasking applications involving multiple CNN mod-

els, resulting in low utilization and scheduling efficiency. This paper proposes an effective solution called Multi-Core with Different Size (MCDS) and DPU Plus, aiming to improve DPU utilization in terms of time and space. By increasing the number of DPU cores and optimizing their utilization, MCDS enhances overall throughput while efficiently managing on-chip resources. Additionally, the DPU Plus design enhances scheduling efficiency by concurrently implementing the DPU with other crucial auxiliary modules on the same FPGA. Experimental results demonstrate the effectiveness of the proposed approach, with a 16.2x acceleration compared to CPU execution and a 3.0x increase in system throughput.

The majority of articles in the field focus on Convolutional Neural Network (CNN) implementations on FPGA platforms utilizing the DPU as a hardware accelerator. In this thesis, however, a novel approach is taken by experimenting with various Artificial Neural Network (ANN) models for hardware implementation using the DPU IP. The results of these experiments reveal a remarkable inference speedup of 90x, showcasing the effectiveness and potential of this approach in accelerating neural network computations on FPGA-based systems.

Chapter 3

Initial Experiments and Findings

The current chapter focuses on the early experimental stages of this thesis, highlighting the evolution of the initial task description and the discovery of the proposed solution. It provides insights into the influence of Vitis and Vitis AI in guiding the development of the solution. The chapter explores the reasons behind the changes in the task description and examines the role played by Vitis and Vitis AI in shaping the direction of the research.

3.1 Vitis

Vitis, known as the Vitis Unified Software Platform, is a software platform developed by Xilinx that facilitates the development of embedded software and accelerated applications on various Xilinx platforms, including FPGAs, SoCs, and ACAPs. It offers a unified programming model for accelerating Edge, Cloud, and Hybrid computing applications. With its embedded programming versatility, Vitis has the potential to significantly enhance the development workflow and speed of embedded and accelerated applications [12].

One of the key advantages of Vitis is its integration of high-level frameworks, allowing developers to utilize programming languages

such as C, C++, or Python to create hardware-accelerated applications. Additionally, it provides low-level APIs that offer greater control over the implementation process when necessary. This combination of high-level frameworks and low-level APIs empowers developers to leverage the full potential of the underlying hardware and optimize the performance of their applications.

3.2 Vitis AI

Vitis AI serves as a dedicated development platform for AI inference on Xilinx hardware platforms. It encompasses a range of optimized IP (DPU), tools, libraries, models, and example designs. With a strong emphasis on high efficiency and user-friendliness, Vitis AI aims to streamline the process of developing and deploying AI models on Xilinx platforms [13].

Vitis AI adopts a Docker-based approach to deliver its suite of tools to developers, offering a more convenient installation experience compared to its predecessor, DNNDK. By leveraging Docker containers, Vitis AI simplifies the process of setting up the required software environment, ensuring consistency and ease of deployment across different systems.

The Vitis AI guide [14] offers comprehensive instructions and examples for developers to leverage the platform’s capabilities in performing different applications. It covers installation, model conversion, and optimization techniques, and provides practical examples, enabling effective deployment of AI models on Xilinx hardware platforms.

3.3 Hardware Core

The hardware core at the heart of the development environment is known as the Deep Learning Processing Unit (DPU). This programmable engine is specifically designed to optimize the performance of ANNs and CNNs. CNNs are particularly well-suited for tasks such as object detection and recognition, as they enable efficient interconnection between perceptrons. The DPU can be implemented within the programmable logic (PL) of various Xilinx boards, offering user-configurable options. This flexibility allows the DPU to meet the performance requirements of diverse applications while minimizing FPGA space and cost. More detailed information about the hardware architecture of the DPU, including the DPUCZDX8G optimized for Zynq Ultrascale+ MPSoCs, can be obtained from this document [15].

3.3.1 DPU architecture

The hardware architecture of the DPU, specifically the DPUCZDX8G optimized for Zynq Ultrascale+ MPSoCs, is depicted in Figure 3.1. It comprises four key components:

- **Instruction scheduler:** The instruction scheduler is responsible for managing the execution of instructions, which belong to a specialized set optimized for common operations in CNNs. These instructions are read from on-chip memory and scheduled for execution by the processing engines in the computing engine.
- **On-Chip buffer controller:** It handles operations on the on-chip memory, which stores input data, intermediate results, and

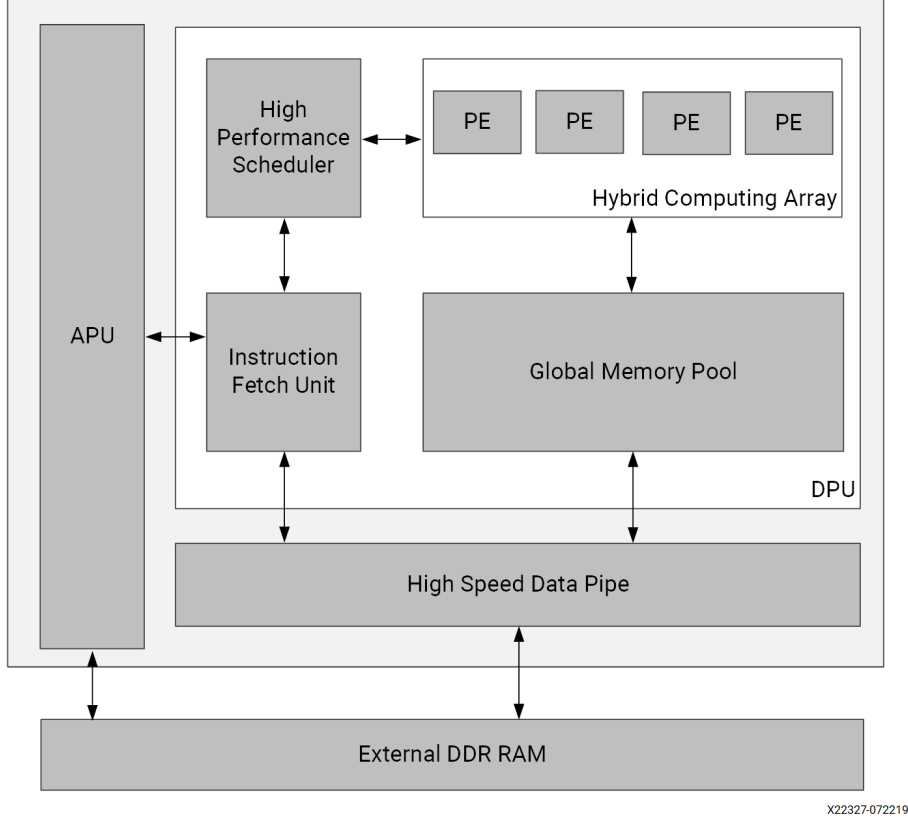


Figure 3.1: *DPU architecture*

outputs. This improves overall DPU performance by avoiding delays associated with off-chip memory communication.

- **Computing engine:** The computing engine serves as the execution core of the DPU, consisting of an array of processing engines (PEs) designed with a deep pipelined architecture to maximize throughput.
- **AXI Interfaces:** AXI interfaces are used for data communication with the off-chip memory and the processor, which controls the DPU's execution.

To further enhance performance, a DSP Double Data Rate (DDR) technique is employed, which increases resource utilization and requires an additional clock. These architectural components and techniques contribute to the DPU's efficiency and performance in accelerating neural network computations.

3.3.2 DPU Naming

Different fields of DPU name are used to indicate different features or purposes, and the naming scheme is shown in the figure 3.2.

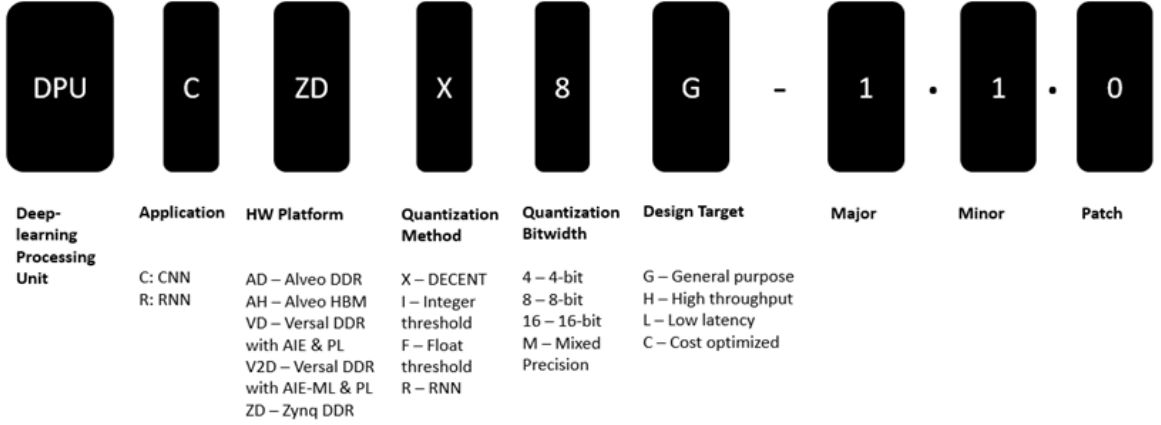


Figure 3.2: *DPU Naming*

The DPUCZDX8G IP is specifically optimized for integration with the Zynq UltraScale+ MPSoC. It can be seamlessly incorporated as a block in the programmable logic (PL) of the chosen Zynq UltraScale+ MPSoCs, establishing direct connections with the processing system (PS). Users have the flexibility to configure the DPU and adjust various parameters to optimize the utilization of PL resources or enable specific features. For detailed instructions on integrating the DPU into custom AI projects or products, please refer to the corresponding documentation - <https://github.com/Xilinx/Vitis-AI/tree/master/dsa/DPU-TRD>

3.3.3 DPU Configuration

The DPU is a configurable core that provides various options for managing resource utilization, such as DSP slices, LUT, block RAM, and UltraRAM. Key configuration parameters include:

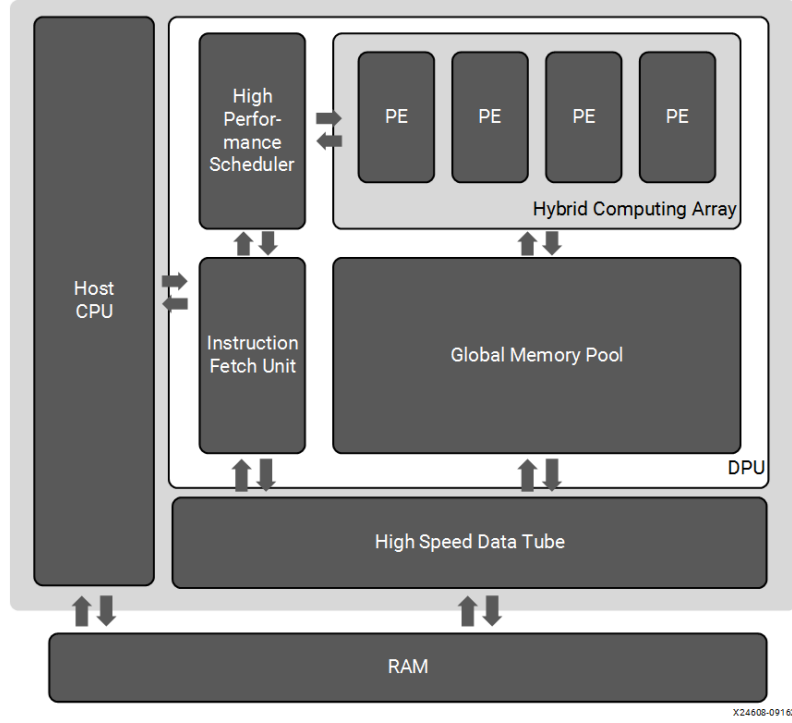


Figure 3.3: *DPUCZDX8G architecture*

- **Number of cores:** A single DPU instance can have up to 4 cores, balancing implementation efficiency with resource usage.
- **Architecture:** The DPU IP provides architectural choices with different levels of parallelism, such as B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096. The numeric value in the names represents the number of operations per clock cycle, impacting parallelism. Higher parallelism enhances performance but demands more resources for DPU implementation.
- **RAM usage:** The on-chip memory utilized to enhance performance is a RAM memory responsible for storing weights, biases, and intermediate results. When instantiating the DPU module, you have the flexibility to choose the amount of RAM to reserve. This can be done by selecting either the *High RAM usage* or the *Low RAM usage* option, allowing the users to optimize the memory footprint based on their specific requirements and constraints.

- **Channel augmentation:** This technique enhances performance by utilizing excess parallelism when the number of input channels is lower than the channel input parallelism of the architecture.
- **Depth-wise Convolution:** In standard convolution, each input channel is processed individually with a specific kernel, and the results are summed. With depth-wise convolution, it splits into two parts: depth-wise and point-wise. Depth-wise processes channels concurrently, while point-wise combines results with a 1x1 kernel. This boosts performance by utilizing parallelism to evaluate multiple activation maps to be evaluated per clock cycle.
- **Average Pool:** Average Pooling is a computation that shrinks a matrix by averaging its elements. The matrix is divided into smaller squares, like 2x2 to 8x8. Each square's elements are summed and divided by the square's element count to get the average. This reduces input size, enhancing performance in neural network inference.
- **ReLU type:** ReLU (Rectified Linear Unit) is an activation function commonly used in neural networks. It transforms negative values to zero while keeping positive values unchanged. There are different variations of ReLU that can be chosen: ReLU, ReLU6, and LeakyReLU.
- **Softmax:** A logistic regression function called the softmax allows for the conversion of any obtained value to probabilities between 0 and 1. Hardware dedicated to this operation is utilized, eliminating the need for the processor to perform it separately.

By customizing these features, users can optimize the DPU implementation for their specific application, finding the right balance

between resource utilization and performance. After configuration, the enabled options are saved in an *arch.json* file, which is later used by the Vitis AI tools.

The following are the parameters of the DPU configuration utilized in this project:

Parameter	Value
DPU cores	1
Arch of DPU	B1152
Usage	low
Channel augmentation	enabled
Average Pool	enabled
Conv	ReLu, Relu6, Leaky Relu
Softmax cores	0

Table 3.1: *DPU parameters in Vivado*

Once the configuration process of the DPU is finished, it's crucial to save the enabled options. This ensures the compiler can make accurate selections for instructions during synthesis. The chosen settings are automatically stored in a file named *arch.json*, which contains the configuration data. The *arch.json* file is later utilized within the Vitis AI environment for further processing and utilization.

3.3.4 DPU Integration

Once the structure and functionalities of the DPU core have been explained, it is essential to understand the integration process within a project. Vitis AI offers two possible solutions [16].

→ The first approach involves creating a project with VIVADO, the Xilinx software used to design, synthesize, and load hardware systems onto the FPGA.

→ The second option is to utilize the Vitis IDE, a comprehensive tool that enables the creation of applications combining software and hardware components.

In both cases, it is crucial to consider that the DPU requires communication with the processing system (PS), which is the software environment running on the processor. This communication involves control signals, instructions, and data exchange. To facilitate this task, communication interfaces need to be added, and a communication protocol must be selected. The DPU employs the AXI (Advanced eXtensible Interface) protocol, a burst-based protocol that divides communication into five independent channels: read address, read data, write address, write data, and write response. The architecture includes the following interfaces:

- An **AXI4-lite slave interface** is utilized to receive control data and base addresses where instructions are stored in memory. The received data is stored in internal registers for runtime availability.
- An **AXI4 master interface** is used to access the instruction memory based on the received base addresses. It reads the instructions that need to be executed.
- An optional **AXI4 master interface** is automatically inserted if the softmax option has been enabled during the configuration process. This interface is responsible for handling the softmax data, ensuring efficient processing of the softmax operation.

- Two **AXI4 master interfaces** are utilized to read the data that needs to be processed during execution. The presence of two separate interfaces enables the reading of larger data, maximizing the utilization of DPU parallelism.

In addition to the communication interfaces, integrating the DPU requires considering other signals such as interrupt signals, clocks, and resets. The number of interrupt signals corresponds to the number of cores in the DPU, with each core having its own interrupt signal. Two clocks are needed, one for managing the communication protocol and the other for the internal logic. The frequency of the internal logic clock should be twice that of the communication clock. Similarly, two resets are used, one for communication handling and the other for internal logic. The figure 3.4 shows a diagram of DPU IP with its interfaces.

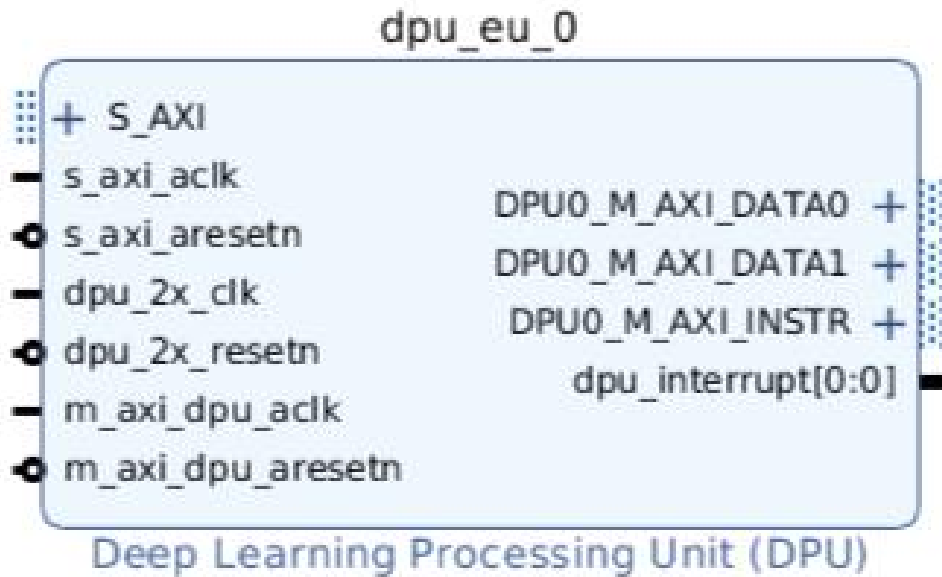


Figure 3.4: *DPU IP with all its interfaces*

Chapter 4

Tools and Prototyping Environment Utilized

This chapter explains the tools that were used to put the suggested solution into practice. There is a brief description of the history of each tool and how it was applied to the proposed solution under each one.

4.1 Xilinx

Xilinx is a renowned technology company that specializes in delivering a diverse range of computer hardware and software solutions [17]. Established in 1984 in Silicon Valley, Xilinx has gained global recognition for its programmable logic devices. Throughout this solution, Xilinx plays a crucial role by providing the majority of the tools utilized in the development process. With a global presence, Xilinx continues to supply its innovative products to customers worldwide.

4.1.1 Vitis AI

An Integrated Development Environment called Xilinx® Vitis AI can be used to speed up AI inference on Xilinx platforms. Throughout the development process, Vitis AI offers resources including ex-

ample designs and tutorials in addition to optimised IP, tools, libraries, and models. Its great efficiency and simplicity of usage enable it to fully realise the AI acceleration potential of Xilinx SoCs and Alveo Data Centre accelerator cards. For more information, refer to section 3.2

4.1.2 Vitis AI DPU

Vitis AI DPU (Deep Learning Processing Unit) is a hardware accelerator specifically designed by Xilinx for deep learning tasks. It is an essential component of the Vitis AI development platform, enabling efficient and high-performance execution of CNNs on Xilinx FPGA platforms. The DPU provides optimized hardware implementations for key CNN operations such as convolution, pooling, and activation functions, resulting in accelerated inference and improved energy efficiency. With its flexible architecture and integration with the Vitis AI toolchain, the Vitis AI DPU empowers developers to deploy and optimize their deep learning models for a wide range of applications, from edge devices to data centers.

The provided article [13], serves as a comprehensive guide for integrating the DPU into a custom platform. It offers step-by-step instructions and explanations on how to successfully integrate the DPU IP into your own hardware design. The guide covers various aspects, including the required software and hardware components, configuration options, and implementation considerations. By following the instructions outlined in the article, developers can effectively integrate the DPU into their custom platforms and leverage its capabilities for efficient and accelerated deep learning inference.

4.1.3 Vitis AI Quantiser

Integer quantization is a technique employed in the deployment of neural networks on Xilinx DPUs to enhance efficiency. By quantizing neural network weights and activations to integer values, various benefits are achieved, including reduced energy consumption, decreased memory usage, and optimized data path bandwidth during inference. This approach optimizes the deployment of neural networks on Xilinx DPUs, resulting in improved overall performance and resource utilization.

Vitis AI Quantizer is a tool provided by Xilinx as part of the Vitis AI development platform [13]. It is designed to facilitate the quantization process of deep neural networks, which is a technique used to reduce the precision of network parameters and activations. By quantizing a network, the precision of numerical values can be reduced from floating-point to fixed-point or integer representations. This results in smaller model sizes and improved inference performance on target hardware. The figure 4.1 shows the Vitis AI Quantizer Framework.

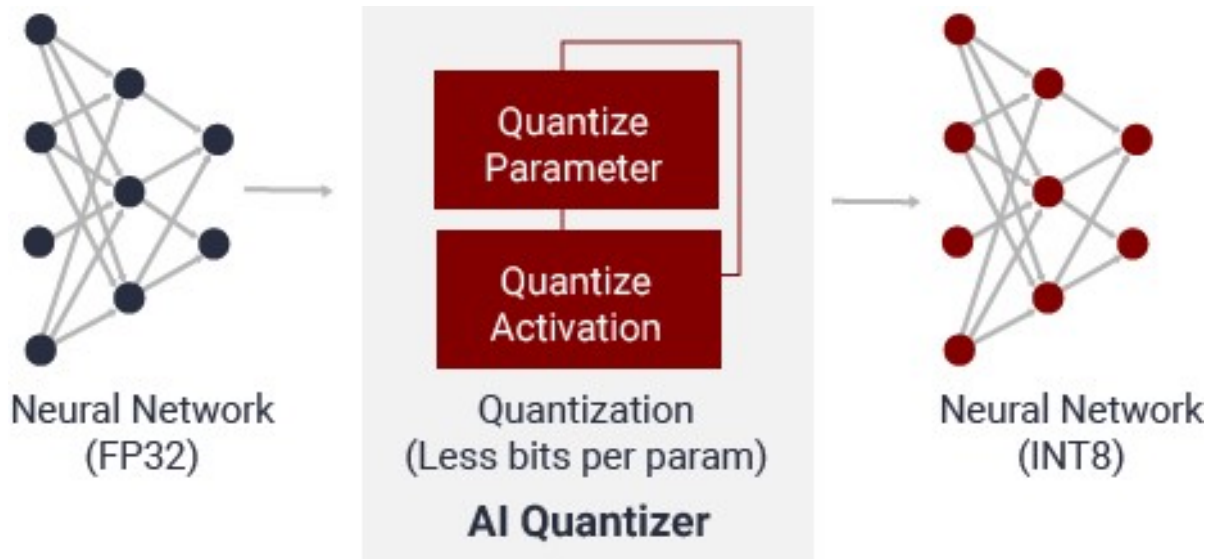


Figure 4.1: *Vitis AI Quantizer Framework*

In scenarios where specific network topologies lead to significant accuracy degradation, developers can utilize a technique called Quantization Aware Training (QAT). By leveraging the original training data, QAT enables multiple backpropagation passes to fine-tune quantized weights, minimizing accuracy loss. This approach optimizes the quantization process by considering accuracy impact during training, resulting in improved performance and preserving desired accuracy levels for neural networks. With Vitis AI Quantizer, developers can effectively optimize deep neural network models for Xilinx hardware platforms. The tool reduces precision of network parameters and activations, achieving a balance between model size, inference performance, and resource utilization. This enables efficient and high-performance deep learning inference on Xilinx devices [18]. The overall model quantization flow is detailed in the following figure 4.2.

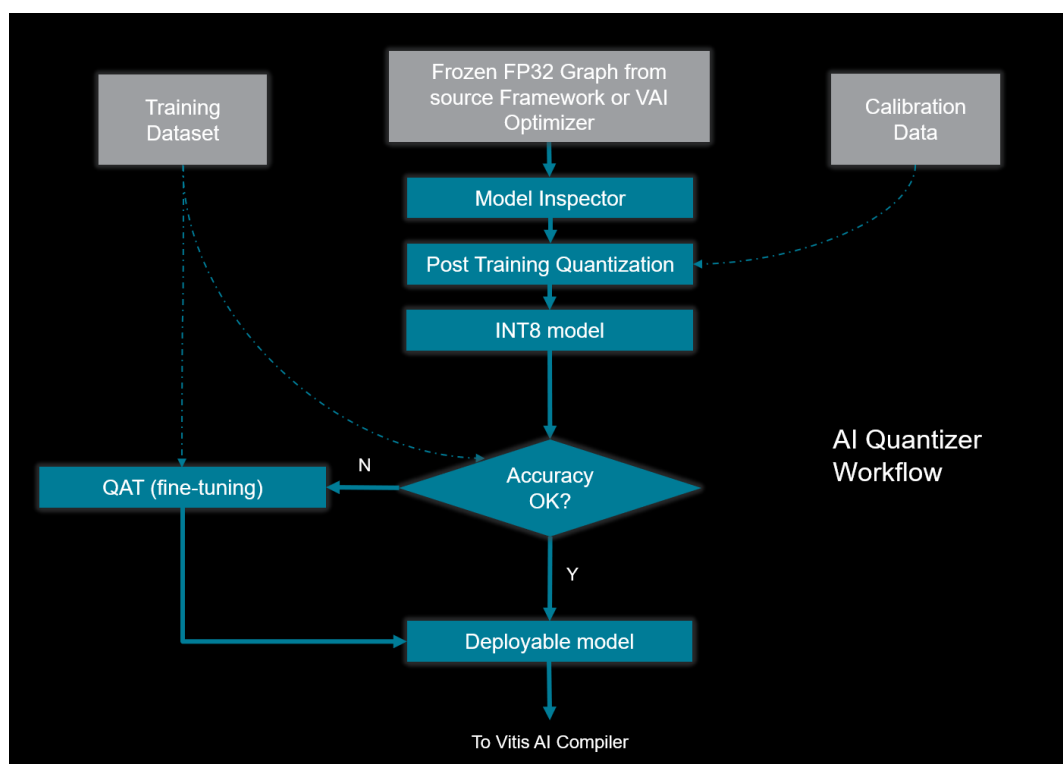


Figure 4.2: *Vitis AI Quantizer Workflow*

4.1.4 Vitis AI Compiler

After quantization, the Vitis AI Compiler plays a crucial role in constructing an intermediate representation (IR) of the model, known as Xilinx Intermediate Representation (XIR). This IR consists of separate control and data flow representations, allowing for various optimizations. For example, batch normalization operations can be fused with convolutions to enhance efficiency. The compiler optimizes the graph and partitions it into subgraphs, ensuring that DPU-executable subgraphs are identified and processed accordingly. Architecture-aware optimizations are then applied, and an instruction stream is generated for the DPU subgraph. The optimized graph is serialized into a compiled .xmodel file. [18]. The following figure 4.3 shows the Vitis AI Compiler Framework.

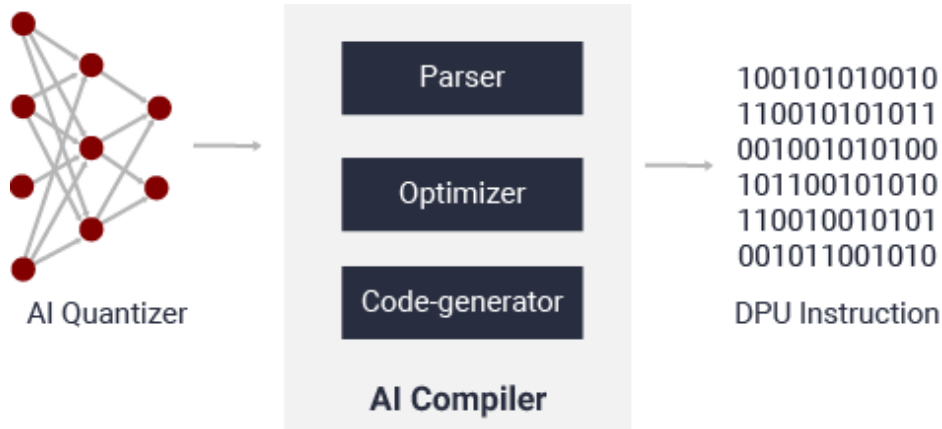


Figure 4.3: *Vitis AI Compiler Framework*

During the compilation process, a DPU arch.json file is provided to the compiler, which communicates the target DPU architecture. It ensures that the compiled model is compatible with the specific DPU's capabilities. Failure to use the correct arch.json file can result in runtime errors. Therefore, models must be recompiled if they are intended for deployment on a different DPU architecture.

After compiling the model into the .xmodel file, Netron can be used to review the final graph structure and gain insights into the optimized model representation. The following diagram 4.4 illustrates a high-level overview of the Vitis AI Compiler workflow:

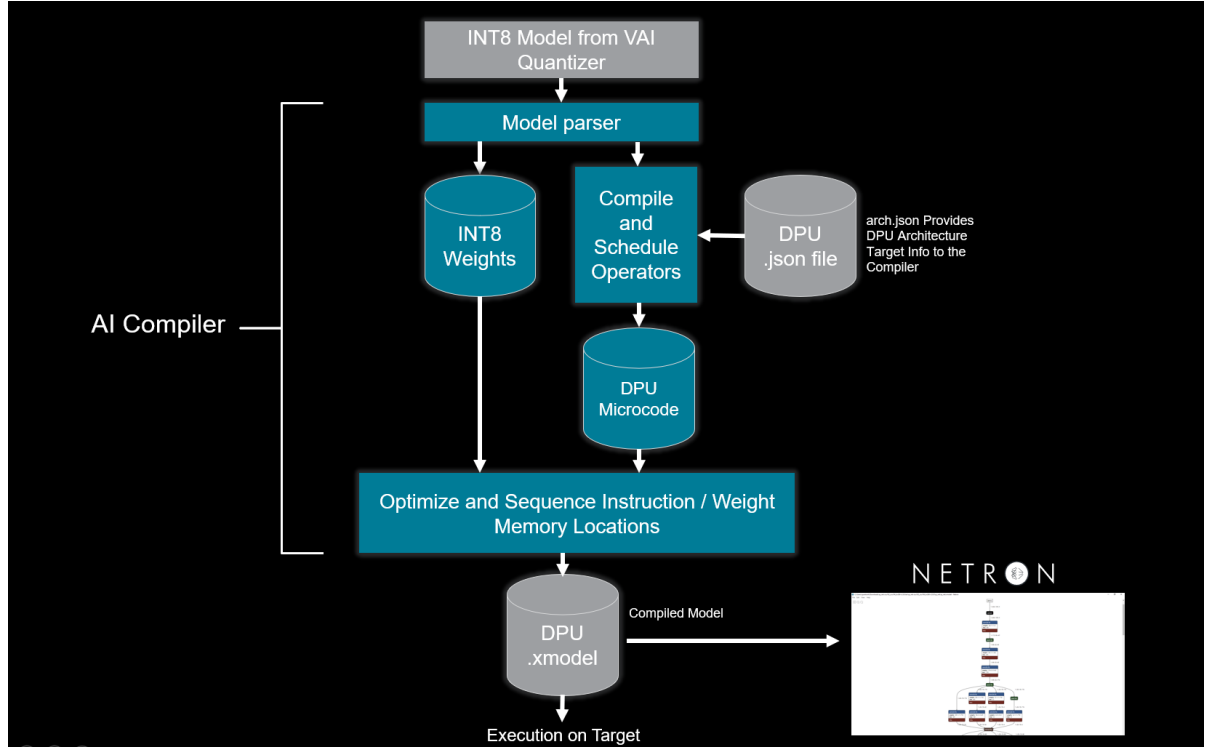


Figure 4.4: *Vitis AI Compiler Workflow*

NOTE! The Vitis AI Compiler is an integral part of the Vitis AI toolchain and is included within the VAI Docker environment. However, the source code for the compiler itself is not made available to users. Users can access and utilize the compiler functionality through the provided Vitis AI toolchain, which includes the necessary binaries and libraries for the compilation and optimization of models.

4.1.5 ZCU104

The ZCU104 board, developed by Xilinx, is an evaluation platform that utilizes the Zynq UltraScale+ ZU7 device. It offers a comprehensive set of features and interfaces for prototyping and testing various applications, including machine learning. The ZCU104 board provides high-performance processing capabilities, FPGA fabric, integrated peripherals, and connectivity options, making it suitable for a wide range of development projects. This Xilinx user guide [19] describes in detail the features of the ZCU104 evaluation board. This guide can be used for developing and evaluating designs targeting the Zynq UltraScale MPSOC devices, specifically ZCU104 board. The Vitis AI pre-built board images support the ZCU104 interfaces shown in the figure 4.5.

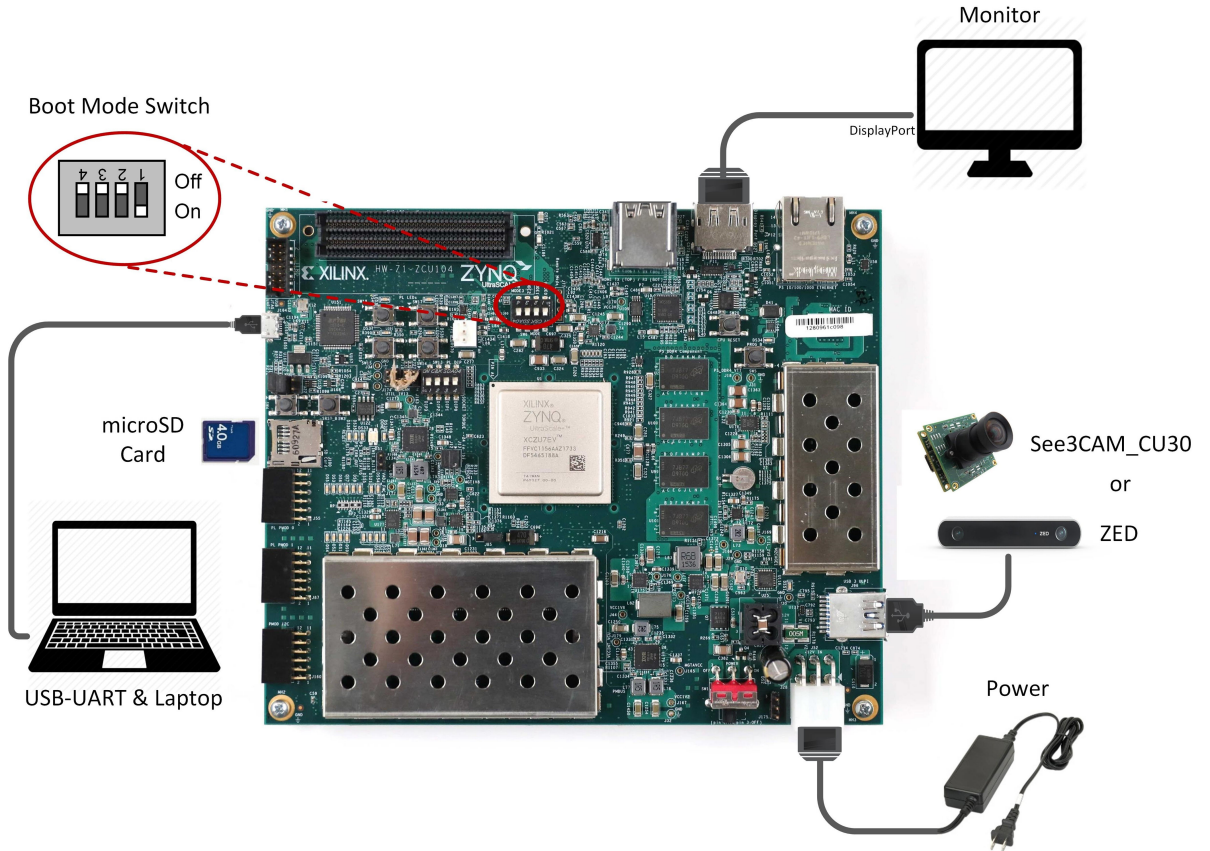


Figure 4.5: *Xilinx ZCU104 Evaluation Board and Peripheral Connections*

The Xilinx ZCU104 evaluation board incorporates the ZU7 Zynq UltraScale+ device, providing an ideal platform for accelerating machine learning applications. With its mid-range capabilities, the ZCU104 enables developers to kickstart their machine-learning projects efficiently. More detailed information about the ZCU104 board can be found on the Xilinx website. <https://www.xilinx.com/products/boards-and-kits/zcu104.html>.

4.2 User Hardware

The host PC utilized in this process was a desktop computer with the following key specifications:

- Processor: Intel Core I7 Processor
- RAM: 24GB DDR4-2666
- Storage device: M.2 SSD NVMe PCIe Gen3
- Operating system: Ubuntu 18.4.3 LTS

4.3 User Software

4.3.1 Docker

Docker is an open-source platform that enables developers to automate the deployment, scaling, and management of applications within isolated containers. Containers are lightweight, standalone environments that encapsulate an application and all its dependencies, including libraries, binaries, and configuration files.

The key components of Docker include:

- **Docker Engine:** The runtime engine that allows you to build, run, and manage containers. It provides the necessary tools and services for container orchestration.
- **Docker Image:** A read-only template that contains the application code, runtime, libraries, and dependencies required to run an application. Images serve as the building blocks for containers.
- **Docker Container:** Containers are isolated environments running applications with their dependencies, ensuring consistency and portability across different systems.
- **Dockerfile:** A text file that contains instructions for building a Docker image. It defines the base image, dependencies, configuration, and other components required to create a reproducible image.

Docker for Vitis AI

Docker for Vitis AI is a containerization platform that enables users to create and deploy portable and isolated environments for running Vitis AI applications. By utilizing Docker, users can package their Vitis AI application and its dependencies into a container that can be easily distributed and run on any system that supports Docker.

4.3.2 Jupyter Notebook

Jupyter Notebook is a browser-based interactive programming environment that enables the execution of programs, including languages like Python, in an interactive manner. It provides a user-friendly interface where code, text, and visualizations can be combined in a single document, making it convenient for data exploration, analysis, and collaboration.

Chapter 5

Model Comparison

The effectiveness and inference time of four different model architectures was explored in this study for the purpose of classifying chest X-ray images. The chosen architectures were initially pre-trained on large datasets and subsequently fine-tuned using our distinct chest X-ray image dataset.

In this project, **RESNET-50**, **DenseNet-169**, **VGG-16**, and **MobileNet** were the selected model architectures. These architectures were specifically chosen due to their robustness and track record in successfully addressing classification tasks.

The application of pre-trained weights, learned from training on extensive datasets like ImageNet, was initiated for each model. Subsequently, a fine-tuning process was undertaken using our chest X-ray image dataset. During this process, the basic model layers were frozen, except for the final dense layers, to enable the models to acquire domain-specific features and patterns.

After fine-tuning, the performance of each model is evaluated on both CPU and DPU platforms to assess their effectiveness, accuracy and inference time across different setups. This evaluation provides valuable insights into the potential advantages of leveraging specialized

hardware, specifically DPUs, for deep learning tasks. By comparing the performance metrics of the models on CPU and DPU platforms, we can determine the impact of hardware acceleration on their effectiveness. These metrics, such as accuracy, precision, recall, and F1 score, offer insights into the comparative performance of the models under different hardware setups.

Each of the model architectures will be analyzed in five segments, where the details of each respective segment are broken down:-

1. **Description of the model:** It contains a brief description of the model architecture along with its specializations.

2. **Reason for choosing the model:** It contains a logical reason for which the model architecture was chosen for the task.

3. **Training/Fine-tuning the model:** It contains steps and details regarding the training of the model.

4. **Quantization:** The Vitis AI Quantizer tool helps with the quantization process of DNN models, reducing their precision from floating-point to fixed-point representation for improved performance, computational efficiency and resource utilization on Xilinx devices.

5. **Compiling quantised model to xmodel:** Vitis AI platform includes a compiler infrastructure that optimizes and compiles the quantized AI models for deployment on Xilinx FPGAs or SoCs. The compilation process involves transforming the model and generating efficient hardware-specific instructions for accelerated inference.

5.1 RESNET-50

5.1.1 Description

ResNet-50, as part of the ResNet (Residual Network) family, is a CNN architecture. This study employs the '50-layered' version of ResNet, indicating a moderately deep network structure.

It introduces residual connections, allowing information to flow directly across layers. The model consists of several residual blocks, each containing stacked convolutional layers and shortcut connections. These connections enable the learning of residual mappings, alleviating the vanishing gradient problem.

The architecture of ResNet-50 typically includes initial convolutional layers, followed by four stages of residual blocks with different numbers of filters. The spatial resolution of the feature maps decreases as the network deepens, while the number of filters usually doubles at each stage. The final stage involves global average pooling to reduce the spatial dimensions, followed by a fully connected layer for classification. Hence, it is able to extract deep meaningful spatial features from the data.

5.1.2 Reason for choosing the model

RESNET-50 is a suitable model for chest X-ray classification due to its deep architecture, pre-trained weights, and transfer learning capabilities. Its depth allows it to capture intricate patterns in X-ray images, while pretraining on large datasets provides general visual representations. The model's residual connections address training challenges, and its success in image classification tasks makes it a promising choice for chest X-ray classification.

5.1.3 Training/Fine-tuning the model

After undergoing the fine-tuning process, the pre-trained ResNet-50 model yields the following results:-

- **Model size:** 286.8 MB
- **Evaluation of the fine-tuned model:** The confusion matrix 5.1 represents the performance evaluation of a classification model with three classes. The diagonal elements of the matrix indicate the true positive values for each class. Specifically, for Class 1, there are 18 instances correctly classified as positive. For Class 2, there are 18 instances correctly classified as positive. And for Class 3, there are 12 instances correctly classified as positive. The confusion matrix provides a detailed breakdown of the model's accuracy and misclassifications for each class.

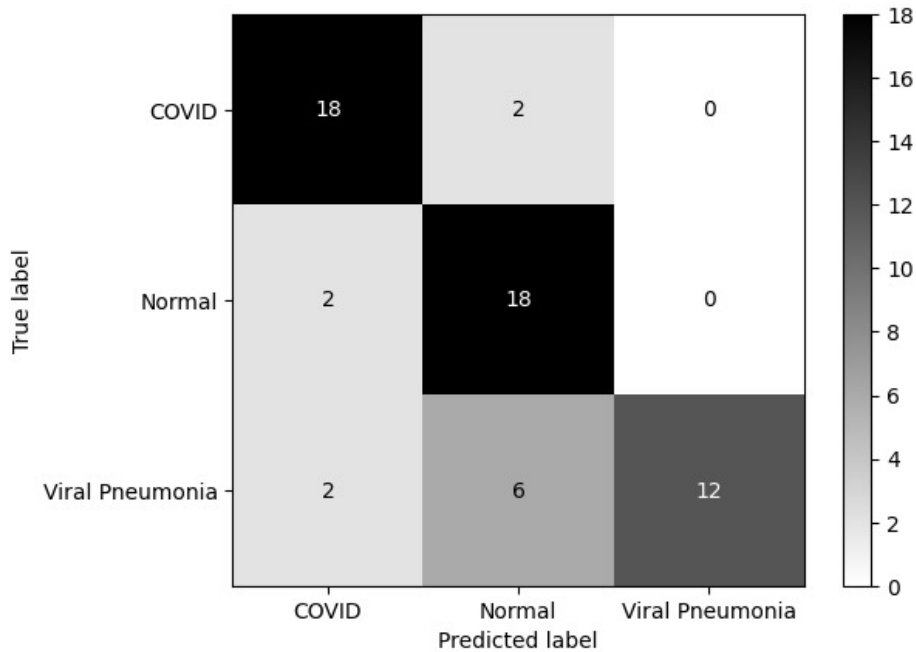


Figure 5.1: *Confusion Matrix for ResNet-50 model*

Table 5.1 presents the classification report for the RESNET-50 model. The classification report provides performance metrics such as precision, recall, F1-score, and support for each class in the classification task. The classification report table shown shows that the average accuracy obtained for each class is 0.80. This means that, on average, the classification model correctly predicts the class labels with an accuracy of 80%.

	Precision	Recall	F1-score	Support
COVID	0.82	0.90	0.86	20
Normal	0.69	0.90	0.78	20
Viral Pneumonia	1.00	0.60	0.75	20
accuracy	-	-	0.80	60
macro avg	0.84	0.80	0.80	60
weighted avg	0.84	0.80	0.80	60

Table 5.1: *Classification Report of ResNet-50 model*

5.1.4 Quantization

Quantization of the trained model weights using the Vitis-AI quantizer yields the following results:-

- **Time taken:** 491.09 seconds
- **Model size:** 95.8 MB
- **Evaluation of the Quantized model:**

After the quantization process, there have been slight changes in the true positive values of the model as depicted from the table 5.4. Specifically, for Class 1, the true positive count has changed from

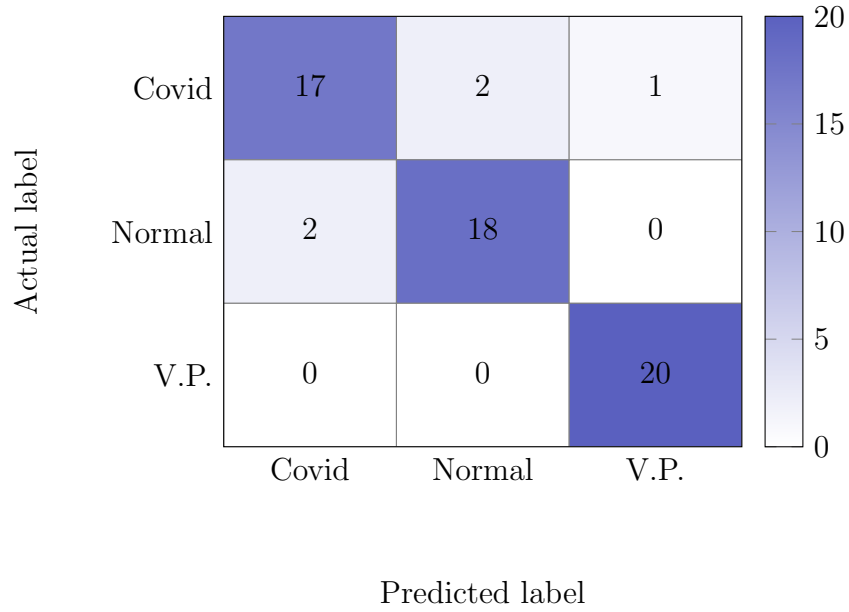


Table 5.2: *Confusion Matrix for Quantized ResNet-50 model*

18 to 17. For Class 2, the true positive count remains the same at 18. However, for Class 3, the true positive count has changed from 12 to 20. Higher true positive values after quantization indicate improved accuracy, as the quantized model correctly identifies more instances from each class.

	Precision	Recall	F1-score	Support
COVID	0.89	0.85	0.87	20
Normal	0.90	0.90	0.90	20
Viral Pneumonia	0.95	1.00	0.98	20
accuracy	-	-	0.92	60
macro avg	0.92	0.92	0.92	60
weighted avg	0.92	0.92	0.92	60

Table 5.3: *Classification Report of Quantized ResNet-50 model*

5.1.5 Compiling quantised model to xmodel

Compiling the quantized model to xmodel yields the following results: -

- **xModel size:** 25.1MB
- **Evaluation of the Compiled model:**

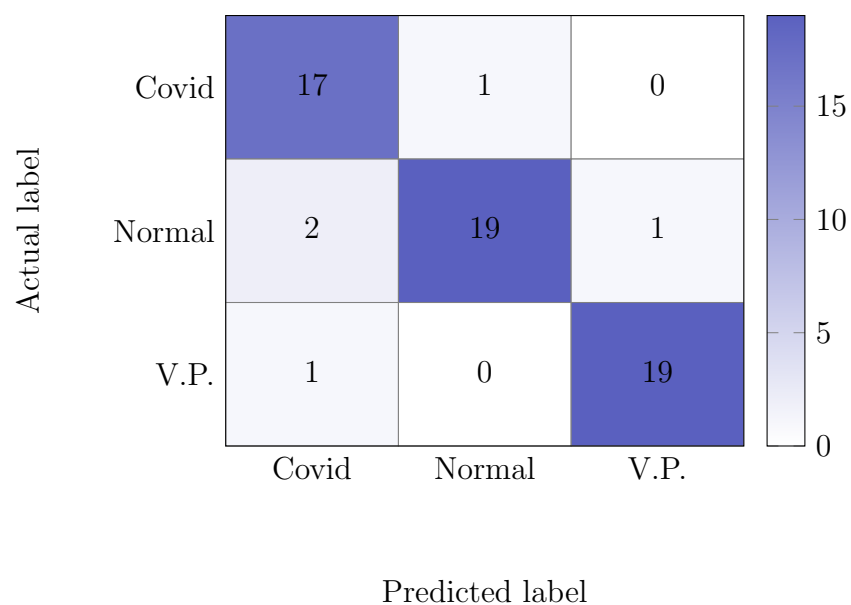


Table 5.4: *Confusion Matrix for Compiled ResNet-50 model*

	Accuracy	Precision	Recall	F1-score
COVID	0.93	0.85	0.94	0.9
Normal	0.93	0.95	0.86	0.90
Viral Pneumonia	0.97	0.95	0.95	0.95

Table 5.5: *Classification Report of Compiled Resnet-50 model*

5.2 DENSENET-169

5.2.1 Description

DenseNet-169 is a convolutional neural network architecture known for its dense connectivity pattern. It consists of 169 layers, making it a deep model. The key characteristic of DenseNet-169 is its dense blocks, where each layer is directly connected to every other layer within the block. This dense connectivity promotes feature reuse and gradient flow, enabling better information propagation throughout the network. By densely connecting layers, DenseNet-169 achieves efficient parameter usage and reduces redundancy. This architecture has shown impressive performance in various computer vision tasks, including image classification, due to its ability to effectively capture fine-grained details and exploit feature correlations.

5.2.2 Reason for choosing the model

DenseNet-169 is well-suited for chest X-ray classification due to its dense connectivity, efficient parameter usage, and ability to capture fine-grained details. Chest X-ray images often contain intricate patterns and subtle abnormalities, which can be effectively learned and represented by DenseNet-169's dense blocks. The dense connections promote feature reuse, enabling better gradient flow and information propagation. With its deep architecture and efficient parameter usage, DenseNet-169 can capture the nuances and correlations of chest X-ray abnormalities, leading to accurate classification results. Its success in various computer vision tasks makes it a promising model for chest X-ray classification, particularly in capturing the intricacies of chest pathologies.

5.2.3 Training/Fine-tuning the model

After undergoing the fine-tuning process, the pre-trained DenseNet-169 model yields the following results:-

- **Model size:** 155.9 MB
- **Evaluation of the fine-tuned model:** The confusion matrix represents the performance evaluation of a classification model with three classes. The diagonal elements of the matrix indicate the true positive values for each class. The confusion matrix for a 3-class problem shown in table 5.2 shows true positive values of 18, 20, and 20 for each class.

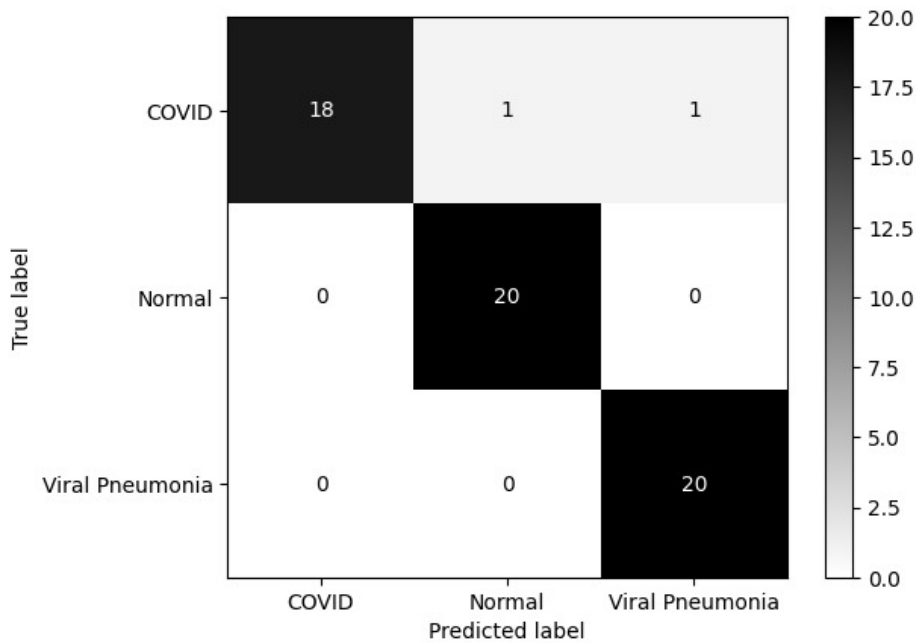


Figure 5.2: *Confusion Matrix for DenseNet-169 model*

The classification table 5.6 shows that the average accuracy obtained for each class is 0.97. This means that, on average, the classification model correctly predicts the class labels with an accuracy of 97%.

	Precision	Recall	F1-score	Support
COVID	1.00	0.90	0.95	20
Normal	0.95	1.00	0.98	20
Viral Pneumonia	0.95	1.00	0.98	20
accuracy	-	-	0.97	60
macro avg	0.97	0.97	0.97	60
weighted avg	0.97	0.97	0.97	60

Table 5.6: *Classification Report of DenseNet-169 Model*

5.2.4 Quantization

Quantization of the trained model weights using the Vitis-AI Quantizer yields the following results:-

- **Time taken:** 1025.25 seconds
- **Model size:** 53.9 MB
- **Evaluation of the Quantized model:**

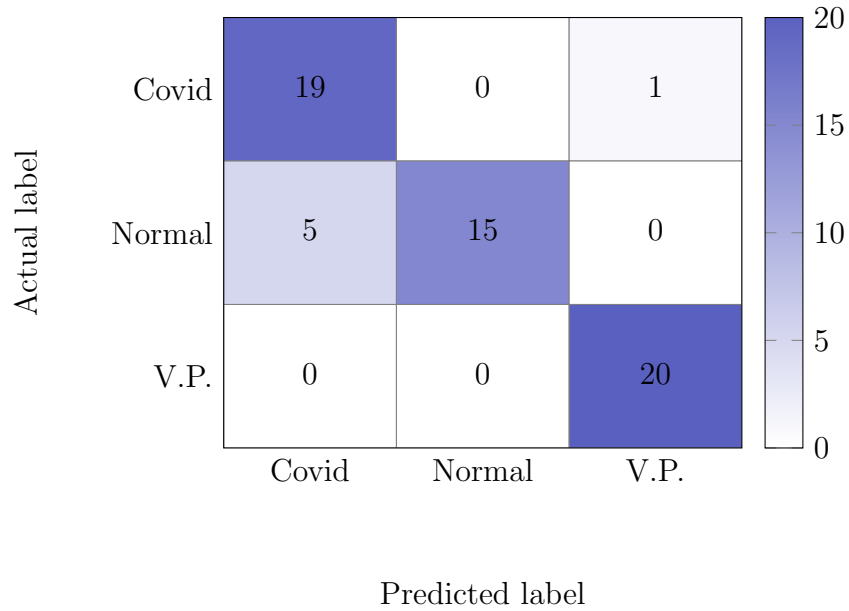


Table 5.7: *Confusion Matrix for Quantized DenseNet model*

	Precision	Recall	F1-score	Support
COVID	0.79	0.95	0.86	20
Normal	1.00	0.75	0.86	20
Viral Pneumonia	0.95	1.00	0.98	20
accuracy	-	-	0.90	60
macro avg	0.91	0.90	0.90	60
weighted avg	0.91	0.90	0.90	60

Table 5.8: *Classification Report of DenseNet-169 Quantized Model*

5.2.5 Compiling quantised model to xmodel

Compiling the quantized model to xmodel yeilds the following results: -

- **xModel size:** 14.2 MB
- **Evaluation of the Compiled model:**

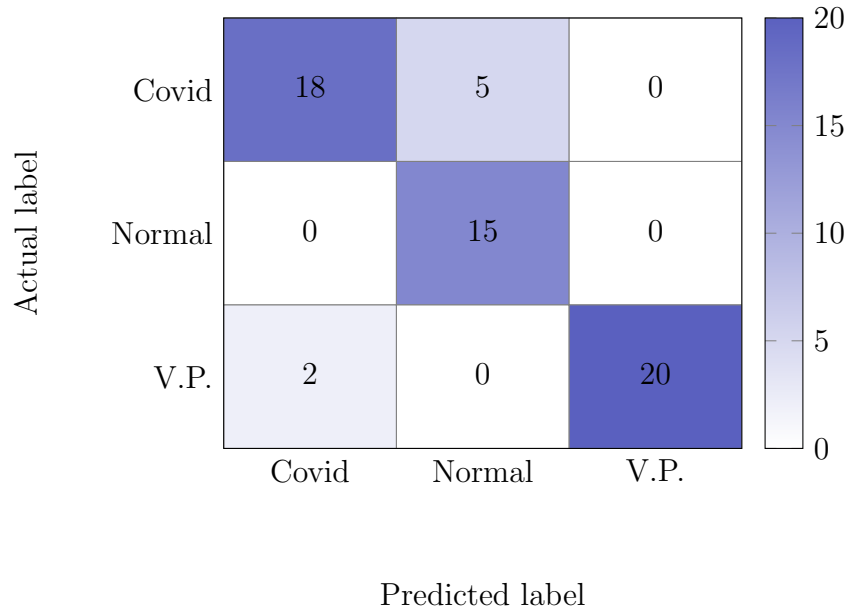


Table 5.9: *Confusion Matrix for Compiled DenseNet-169 model*

	Accuracy	Precision	Recall	F1-score
COVID	0.88	0.90	0.78	0.83
Normal	0.91	0.75	1.00	0.85
Viral Pneumonia	0.96	1.00	0.9	0.94

Table 5.10: *Classification Report of Compiled DenseNet-169 model*

5.3 VGG-16

5.3.1 Description

VGG-16 is a convolutional neural network architecture known for its deep and straightforward structure. It consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. VGG-16 follows a simple and uniform design philosophy, where convolutional layers use small 3x3 filters with stride 1 and max pooling layers with 2x2 filters and stride 2. This architecture enables VGG-16 to capture both low-level and high-level visual features effectively. With its deep layers, VGG-16 can learn intricate patterns in images, making it suitable for various computer vision tasks, including image classification, object detection, and segmentation.

5.3.2 Reason for choosing the model

VGG-16 can be a suitable model for chest X-ray classification due to its deep architecture and strong feature representation capabilities. Chest X-ray images often contain complex patterns and subtle abnormalities that require capturing fine-grained details. VGG-16's deep structure, with multiple stacked convolutional layers, allows it to learn intricate features and hierarchies of visual information. The use of small filter sizes and max-pooling layers helps capture local and global features effectively. Moreover, VGG-16 has been exten-

sively evaluated and proven successful in various computer vision tasks, including image classification, making it a reliable choice for accurate chest X-ray classification.

5.3.3 Training/Fine-tuning the model

After undergoing the fine-tuning process, the pre-trained ResNet-50 model yields the following results:-

- **Model size:** 177.6 MB
- **Evaluation of the fine-tuned model:** The confusion matrix for a 3-class problem in figure 5.3 shows true positive values of 14, 20, and 18 for each class.

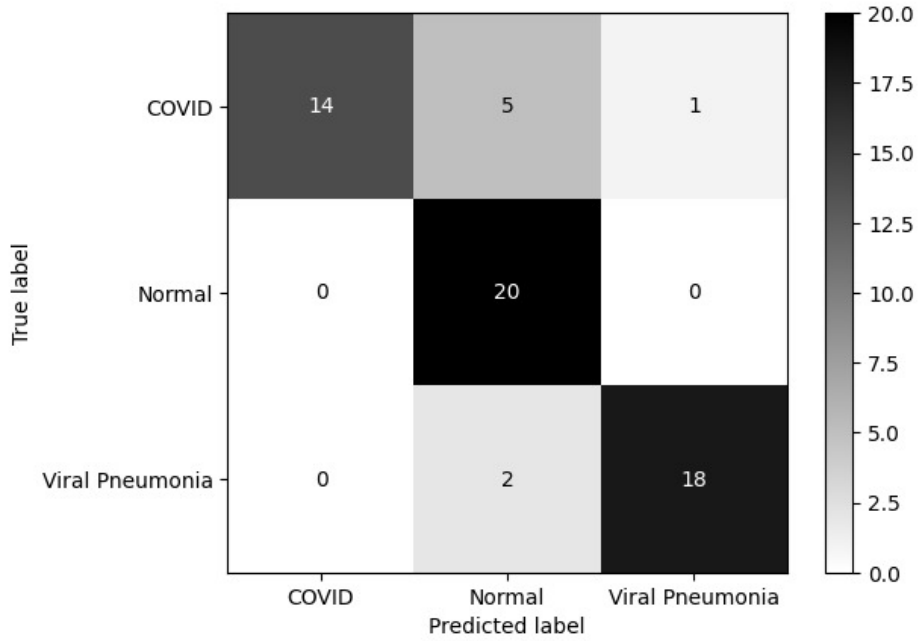


Figure 5.3: *Confusion Matrix for VGG-16 model*

The classification table 5.11 shows that the average accuracy obtained for each class is 0.87. This means that, on average, the classification model correctly predicts the class labels with an accuracy of 87%.

	Precision	Recall	F1-score	Support
COVID	1.00	0.70	0.82	20
Normal	0.74	1.00	0.85	20
Viral Pneumonia	0.95	0.90	0.98	20
accuracy	-	-	0.87	60
macro avg	0.90	0.87	0.87	60
weighted avg	0.90	0.87	0.87	60

Table 5.11: *Classification Report of Vgg-16 model*

5.3.4 Quantization

Quantization of the trained model weights using the Vitis-AI quantizer yields the following results:-

- **Time taken:** 437 seconds
- **Model size:** 59.4 MB
- **Evaluation of the Quantized model:**

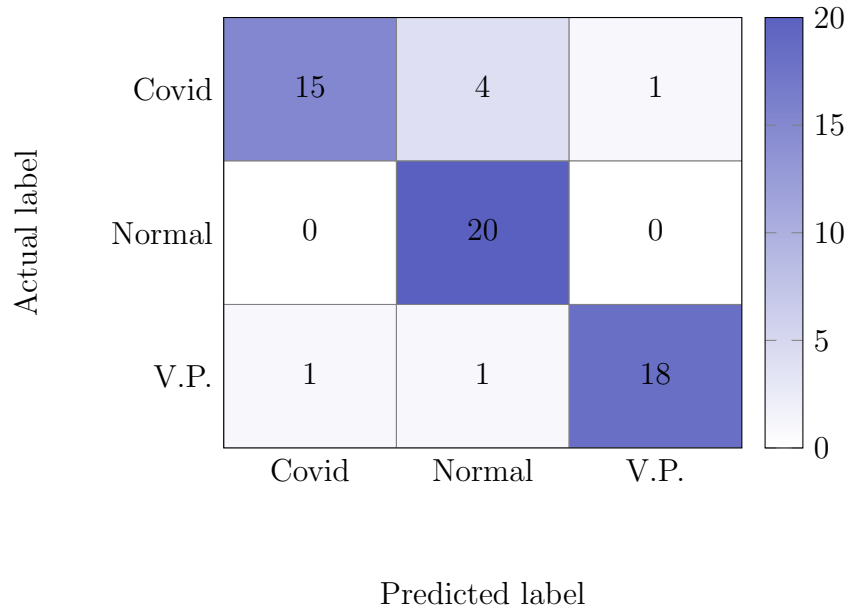


Table 5.12: *Confusion Matrix for Quantized Vgg-16 model*

	Precision	Recall	F1-score	Support
COVID	0.94	0.75	0.83	20
Normal	0.80	1.00	0.89	20
Viral Pneumonia	0.95	0.88	0.88	20
accuracy	-	-	0.88	60
macro avg	0.89	0.88	0.88	60
weighted avg	0.89	0.88	0.88	60

Table 5.13: *Classification Report of Quantized Vgg-16 Model*

5.3.5 Compiling quantised model to xmodel

Compiling the quantized model to xmodel yeilds the following results: -

- **xModel size:** 15.2 MB
- **Evaluation of the Compiled model:**

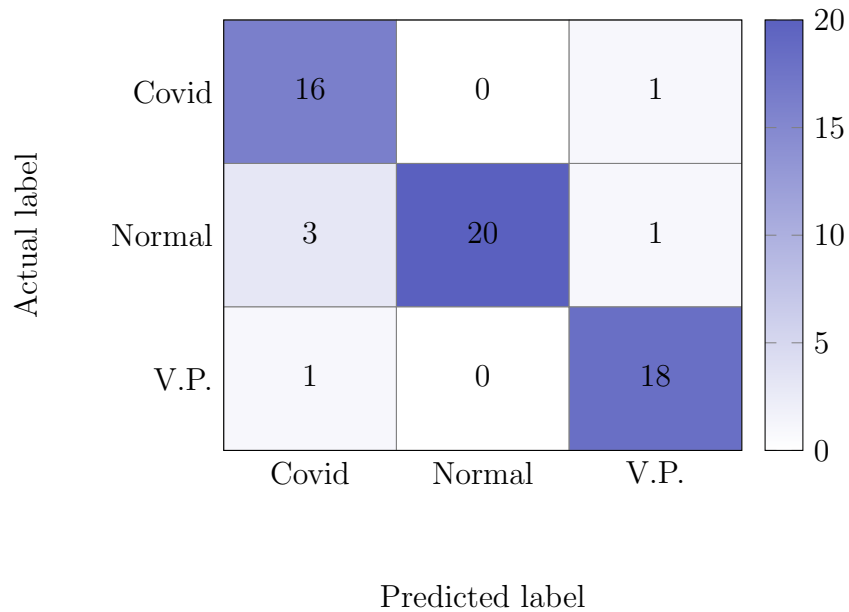


Table 5.14: *Confusion Matrix for Compiled Vgg-16 model*

	Accuracy	Precision	Recall	F1-score
COVID	0.91	0.80	0.94	0.86
Normal	0.93	1.00	0.83	0.90
Viral Pneumonia	0.95	0.9	0.94	0.92

Table 5.15: *Classification Report of Compiled Vgg-16 model*

5.4 MobileNet

5.4.1 Description

MobileNet is a lightweight convolutional neural network architecture designed for efficient and resource-constrained environments, such as mobile devices and embedded systems. It aims to strike a balance between model size and accuracy. It employs depth-wise separable convolutions, which separate the standard convolution operation into depth-wise and point-wise convolutions. This factorization significantly reduces the computational cost and number of parameters while preserving the model’s ability to learn meaningful representations. The architecture allows MobileNet to achieve high efficiency and fast inference times, making it suitable for real-time applications and scenarios where computational resources are limited.

5.4.2 Reason for choosing the model

MobileNet can be a suitable model for chest X-ray classification due to its lightweight architecture and efficient inference. Chest X-ray classification tasks often require real-time performance and are frequently deployed on resource-constrained devices. MobileNet’s design, with depth-wise separable convolutions, significantly reduces the computational complexity and model size while maintaining reasonable accuracy. This efficiency allows for faster inference times and

lower memory requirements, making it well-suited for real-time chest X-ray classification on mobile devices or embedded systems. Despite its compact size, MobileNet can still capture relevant features and patterns in X-ray images, enabling accurate classification of abnormalities with reduced computational overhead.

5.4.3 Training/Fine-tuning the model

After undergoing the fine-tuning process, the pre-trained MobileNet model yields the following results:-

- **Model size:** 40.7 MB
- **Evaluation of the fine-tuned model:** The confusion matrix for a 3-class problem in figure 5.4 shows true positive values of 17, 11, and 20 for each class.

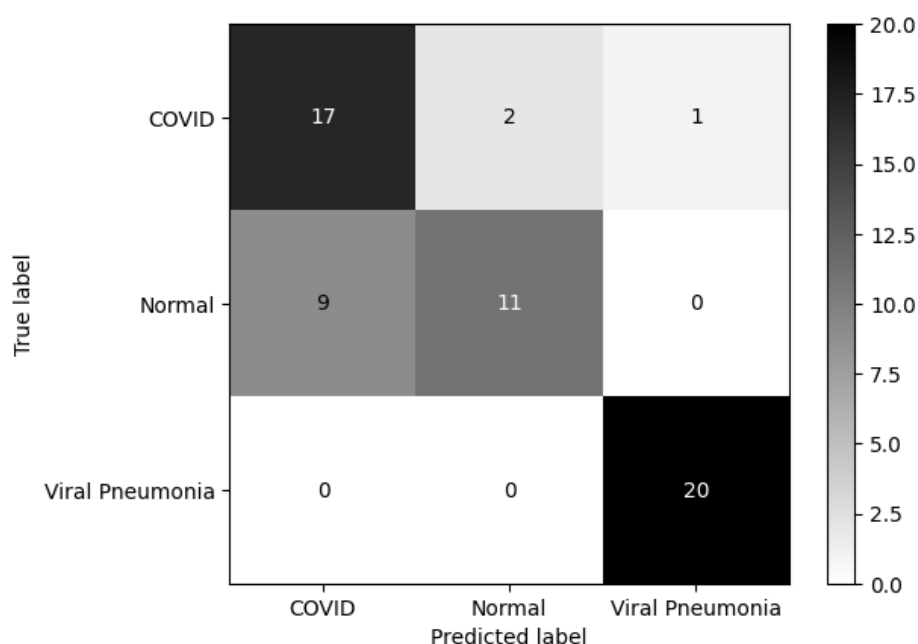


Figure 5.4: *Confusion Matrix for MobileNet model*

The table 5.16 below presents the classification report for the RESNET-50 model. The classification report provides performance metrics

such as precision, recall, F1-score, and support for each class in the classification task.

	Precision	Recall	F1-score	Support
COVID	0.69	0.90	0.78	20
Normal	1.00	0.60	0.75	20
Viral Pneumonia	0.91	1.00	0.95	20
accuracy	-	-	0.83	60
macro avg	0.87	0.83	0.83	60
weighted avg	0.87	0.83	0.83	60

Table 5.16: *Classification Report of MobileNet*

The classification table shown above shows that the average accuracy obtained for each class is 0.83. This means that, on average, the classification model correctly predicts the class labels with an accuracy of 83%.

5.4.4 Quantization

Quantization of the trained model weights using the Vitis-AI quantizer yields the following results:-

- **Model size:** 95.8 MB
- **Evaluation of the quantized model:**

During the evaluation of the quantized MobileNet model, it was observed that all the testing images were predicted for the label Covid [0], which is deemed inaccurate because of the usage of separable convolution in order to reduce parameters. Information can be lost due to the channel-wise convolution involved in separable convolution. In the case of TensorFlow quantization, it has been noted that this operation resembles the addition of noise to an already trained

network, with the expectation that the model has generalized well. While this approach tends to work effectively and maintain high accuracy for heavier models such as Inception and ResNet, it results in accuracy loss for lightweight models like MobileNet, given their simplicity and lightweight nature.

Chapter 6

Description of Workflow

This chapter provides comprehensive step-by-step instructions on utilizing Xilinx Vitis AI to execute multiple DNN models on the ZCU104 Ultrascale MPSoC board. The purpose of integrating various existing tutorials into a single guide is to create a comprehensive resource that covers everything from setting up the environment to running applications on the ZCU104 board. This ensures that users have a consolidated reference for all aspects of working with Vitis AI on the ZCU104 platform.

6.1 Tools Execution Process

While providers usually provide documentation for installing development tools, this section serves as a concise guide. Although it is possible to set up and use the tools without this guide, the author shares some observations from their own experience in setting up the tools. To determine the specific hardware platform used for the solution and the testing process, please refer to 4.2.

6.1.1 Vivado 2020.2 HLx edition

Installation instructions for Vivado HLx edition 2020.2 are widely documented and can be found at [20]. When choosing which software to install in this sample, just Vivado is necessary.

6.1.2 Vitis AI

Vitis AI can be installed using a quick start guide [13]. The majority of requirements for successfully installing Vitis AI on the host PC are covered in this article, however, there are a few factors the author highlights for the reader that may facilitate the installation process. The next section of this guide will cover how to set up ZCU104. For information about ZCU104, refer to section 4.1.5.

6.1.3 Petalinux

The "Petalinux Tools Documentation Reference Guide" [21] contains instructions for installing Petalinux. For installation, please refer to this manual. It is nicely written and offers advice on how to resolve common problems that may arise when installing Petalinux tools.

Launching a *settings.sh* or *settings.csh* script that additionally checks for the installation of Petalinux tools is recommended as the primary step in setting up the Petalinux working environment. The author notes that this script does not operate in a different terminal window and only applies Petalinux tools path variables to the current terminal session. These paths must be present in the Linux environment's \$PATH variable in order to permanently add Petalinux Tools.

6.1.4 Edge AI environment on the ZCU104 board

This subsection is composed of many software packages that are required in order to successfully accelerate the image classification process on the hardware platform and use shell scripts written to ease this process for developers.

It is of utmost importance to note that the installation process may not precisely adhere to the provided instructions. Due to the absence

of official guidelines from Xilinx, the hardware configuration employed in this project encountered several errors [22]. Consequently, it is recommended to install any extra dependencies that may not be covered in the current edition of the official Installation guide. It is possible that this situation may improve with subsequent official versions, leading to more comprehensive guidance.

6.2 Integration of fine-tuned models with ZCU104

The setup of the ZCU104 board and how to get fine-tuned models to run on ZCU104 are covered in this section and its subsections. The present guide is an updated version of Xilinx’s manual UG1414, with a few additional procedures highlighted [13]. As indicated in the introduction of this chapter, the Xilinx guide UG1414 is also replicated here. Each section of this guide includes prerequisites that specify what the user must do or have in order to follow it.

A simplified diagram illustrates the development process for executing fine-tuned models on the ZCU104 board, outlining the key steps and components involved, which can be found in the figure 6.1.

6.2.1 Hardware

The hardware design part employs a guide from Xilinx Vitis Tutorials [23]. Prerequisites for this subsection:

- Vivado Design suite 2020.2 HLx Edition (any edition works)

To create a DPU hardware platform using Vivado 2020.1, follow these steps:

- **Set the Vivado environment:** In the linux terminal, type:

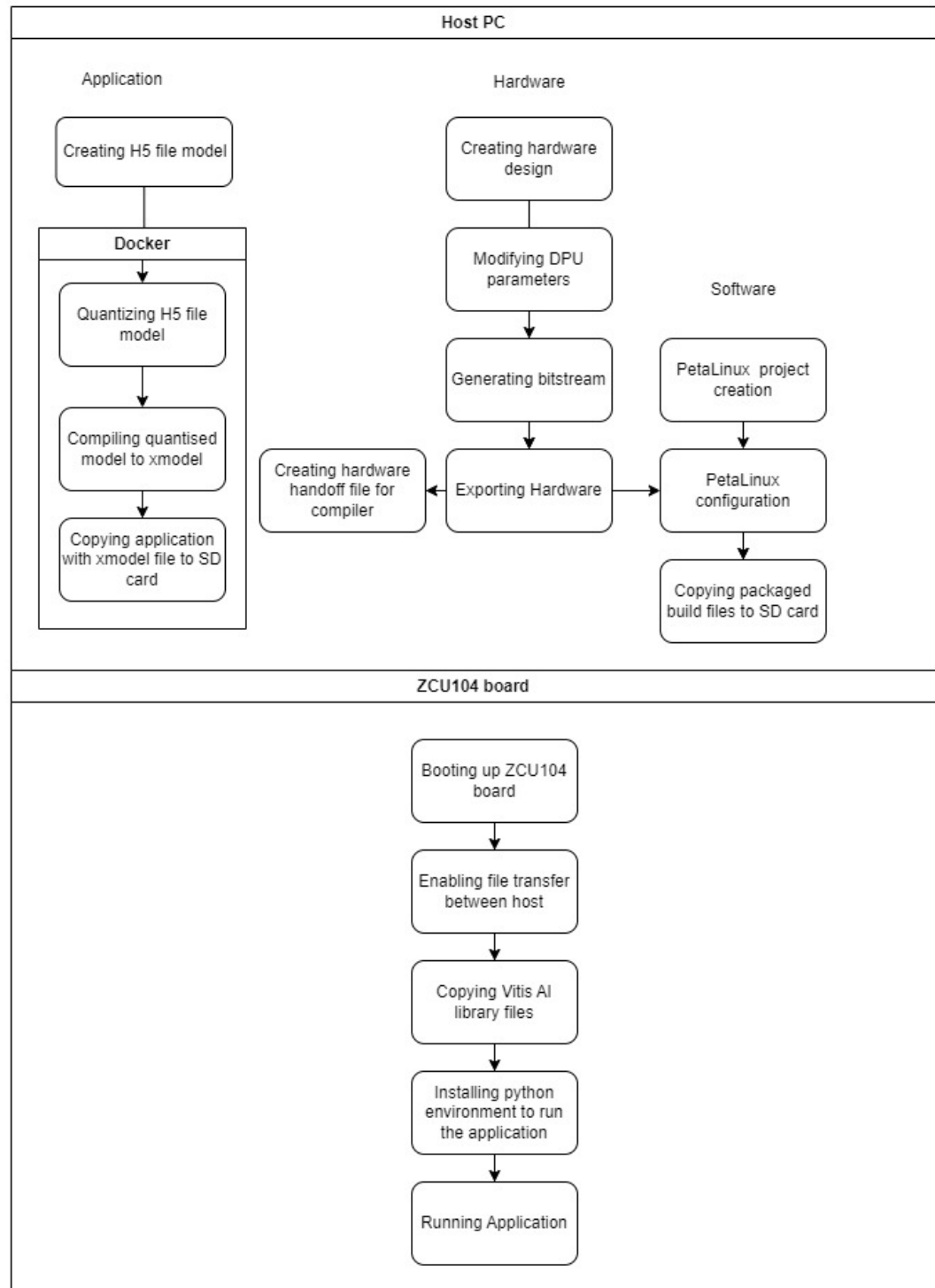


Figure 6.1: *Simplified view of development flow in proposed solution*


```
source <Vivado install path>/Vivado/2020.2/settings64.sh  
vivado
```

- **Launch Vivado Design Suite:** Open Vivado and create a new project or open an existing project to add the DPU hardware platform.
- **Create a new block design:** In the Flow Navigator pane, select "Create Block Design." and give the block design a name and click OK.
- **Add Zynq UltraScale+ MPSoC IP:** In the Diagram view, click on the "+" button to add IP. Search for "Zynq UltraScale+ MPSoC" and double-click on it to add it to your design. Configure the MPSoC IP according to desired settings, including selecting the appropriate part and configuring the PS (Processing System) settings.
- **Run Block Automation:** Right-click on the Zynq UltraScale+ MPSoC block and select "Run Block Automation." This will automatically add the necessary IP blocks and connections for the PS.
- **Add DPU IP:** In the Diagram view, click on the "+" button again and search for "DPU." Double-click on the DPU IP to add it to the design. Connect the DPU IP to the appropriate interfaces of the Zynq UltraScale+ MPSoC block, such as the AXI interconnects.
- **Configure DPU IP:** Double-click on the DPU IP block to open its configuration window. Configure the DPU settings, including the DPU model, clock frequency, and other parameters based on your requirements.

On the active window (Recustomize IP) make sure the DPU configuration is as seen in the figure 6.2.

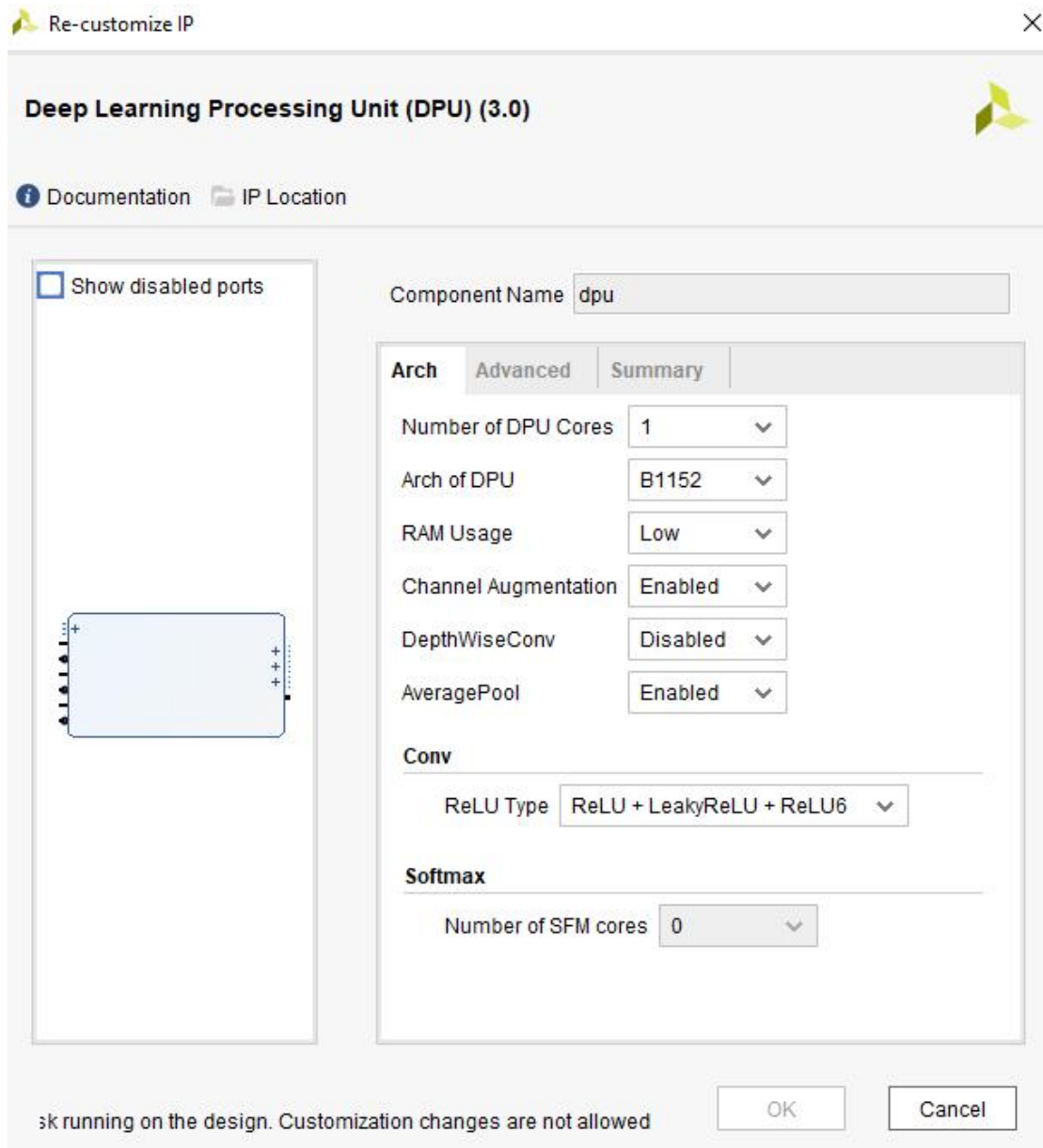


Figure 6.2: Screenshot of DPU configuration used in this project

- **Validate and Package IP:** Click on the "Validate Design" button to check for any errors or warnings. Resolve any issues if found. Once the design is validated, click on the "Package IP" button to create an IP package for the DPU.
- **Generate Bitstream:** In the Flow Navigator pane, click on "Generate Bitstream" to synthesize and implement the design. This will generate the bitstream file for programming the FPGA with the DPU hardware platform.

- **Export Hardware:** After the bitstream generation is complete, select "Export Hardware" from the File menu. Choose the option to include bitstream and specify the output directory.
- **Export Platform:** Finally, the hardware platform can be exported by clicking **File** → **Export** → **Export Platform** and following the wizard's instructions. The resulting XSA file will be named "**zcu104_custom_platform.xsa**". On opening the block diagram (file named as system.bd), one can see the following figure 6.3.

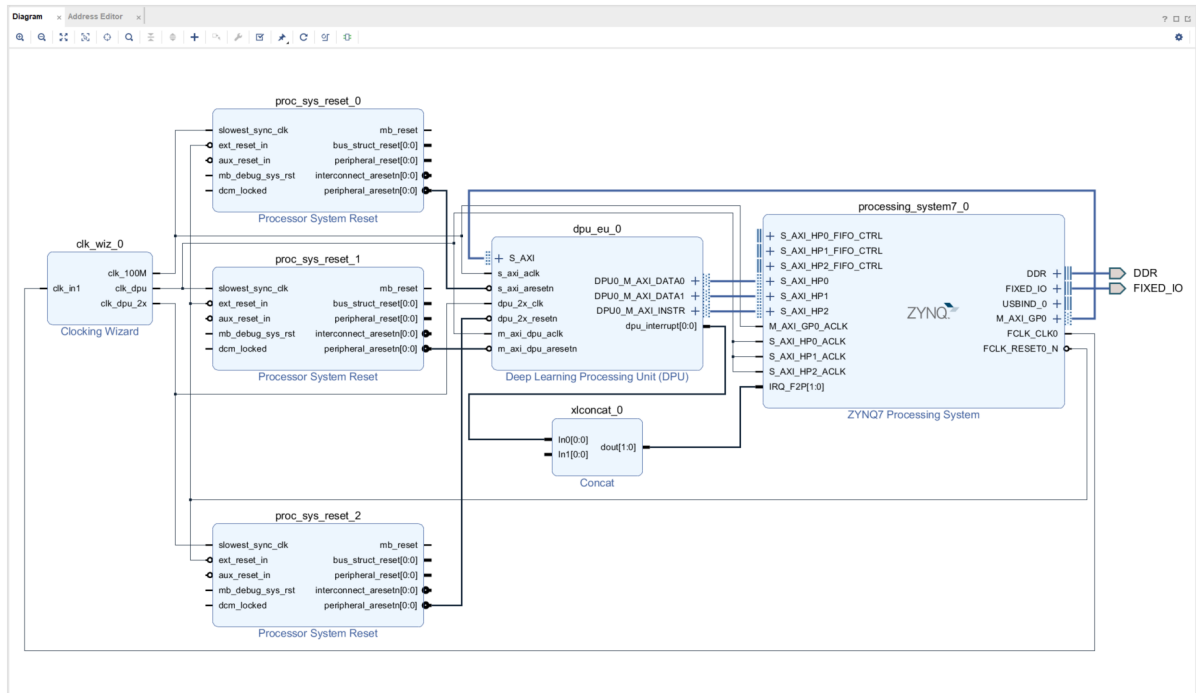


Figure 6.3: Screenshot of a block diagram of hardware configuration used in this project

The process described above outlines the steps to create a DPU hardware platform using Vivado 2020.2. If encountering any issues during the process, refer to [24]. Once the hardware platform is created, it is possible to proceed with software development and the integration of applications with the DPU for accelerated inference.

6.2.2 Software

This subsection demonstrates the use of Petalinux tools for software environment customizations, preparing an SD card, and launching Petalinux to verify that the installation was successful.

Perquisites of this subsection:

- Petalinux and its perquisites are installed and configured on the host computer. For installation, refer to 4.1.3

The following section is mostly copied from Xilinx’s guide UG1144 with additional comments and additions from the author. For reference, please refer to Xilinx’s guide UG1144 and its references [21]. For an overall explanation of the software section, refer to 6.1.3 Petalinux project creation from setting up an environment.

- **Setting up the Petalinux Environment**

```
source <petaLinux_tool_install_dir>/settings.sh
```

- **Creating a Petalinux Project**

```
petalinux-create --t project --template zynq --n zcu104_plnx  
cd zcu104_plnx
```

- **Configure the PetaLinux project with the HW design**

```
petalinux-config --get-hw-description=<vivado_design_dir>
```

- **Adding additional user packages to PetaLinux**

—In the current terminal, run

```
petalinux-config -c rootfs
```

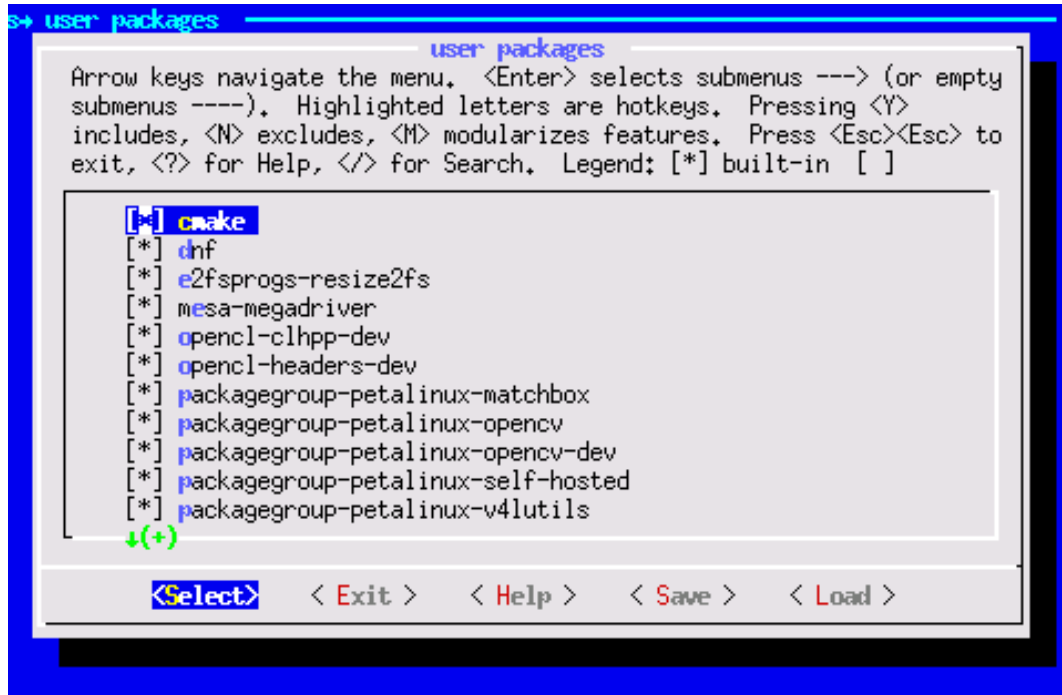


Figure 6.4: *Additional User Packages in PetaLinux*

A window is opened where the user can choose what packages can be installed on Petalinux. All necessary and desired packages must be selected before building the Petalinux project. Recommended packages for running and compiling image classification applications in Petalinux. Recommended packages for running image classification applications in Petalinux:

- Filesystem Packages → misc → packagegroup-core-ssh-dropbear and disable packagegroup-core-ssh-dropbear
- Filesystem Packages → console → network → openssh and enable openssh, openssh-sftp-server, openssh-sshd, openssh-scp.

Save changes and close the PetaLinux rootfs config window.

• Updating the Device tree

The **Device tree** describes the hardware components of the sys-

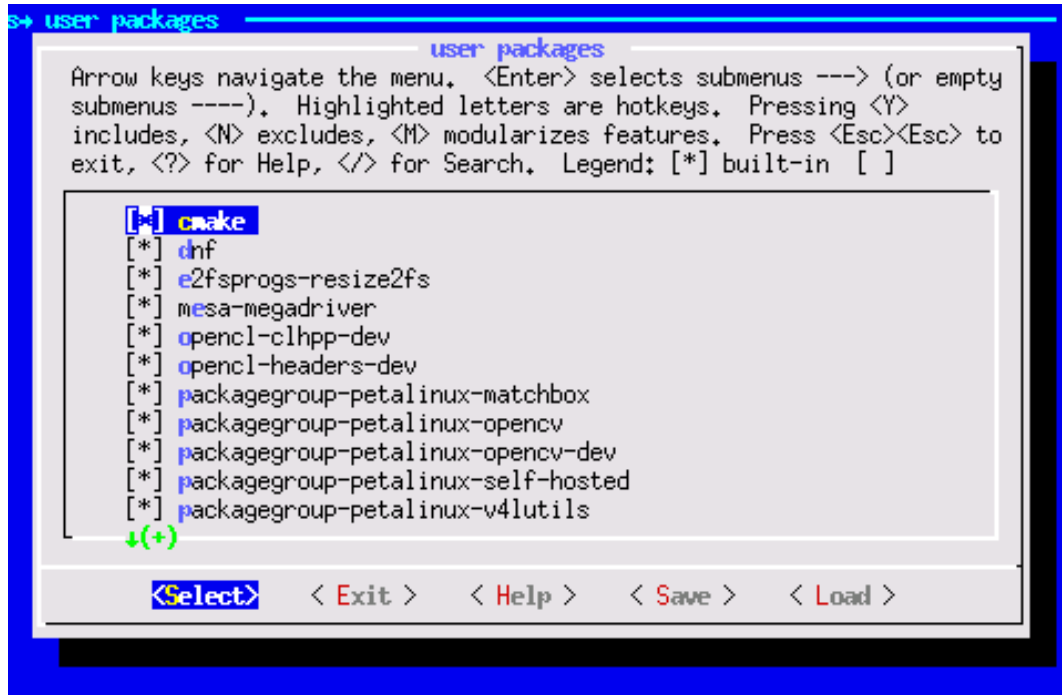


Figure 6.5: *Petalinux Root filesystem configuration window*

tem. The Xilinx Device Tree Generator (DTG) generates the device tree based on hardware configurations from the XSA file. To further customize the system in PetaLinux, users can modify the "system-user.dtsi" file. This allows users to add customization settings for driver nodes that do not have corresponding hardware or to override any auto-generated configurations provided by DTG. By utilizing "system-user.dtsi" file, users can tailor the device tree to their specific requirements in the PetaLinux environment. The contents of the system-user.dtsi file for ZCU104 is given in the figure 6.6. For further details, refer to [25].

★ Now that the Petalinux installation is so big, it will be impossible to get it to start from RAMDisk. So, the solution is to configure Petalinux *rootfs* to mount onto SD card itself. For that, insert:

```
petalinux-config
```

A Petalinux main configuration screen will appear as shown in the figure 6.7.

```

Open ▾ [icon] *system-user.dtsi
~/Downloads/Ranita/_zcu104_custom_pet/pro...ec/meta-user/recipes-bsp/device-tree/files

/include/ "system-conf.dtsi"
/ {

};
&amba {
    zyxclmm_drm {
        compatible = "xlnx,zocl";
        status = "okay";
        interrupt-parent = <&axi_intc_0>;
        interrupts = <0 4>, <1 4>, <2 4>, <3 4>,
            <4 4>, <5 4>, <6 4>, <7 4>,
            <8 4>, <9 4>, <10 4>, <11 4>,
            <12 4>, <13 4>, <14 4>, <15 4>,
            <16 4>, <17 4>, <18 4>, <19 4>,
            <20 4>, <21 4>, <22 4>, <23 4>,
            <24 4>, <25 4>, <26 4>, <27 4>,
            <28 4>, <29 4>, <30 4>, <31 4>;
    };
};

&axi_intc_0 {
    xlnx,kind-of-intr = <0x0>;
    xlnx,num-intr-inputs = <0x20>;
    interrupt-parent = <&gic>;
    interrupts = <0 89 4>;
};

&sdhci1 {
    no-1-8-v;
    disable-wp;
};|

```

Figure 6.6: *Device Tree for ZCU104*

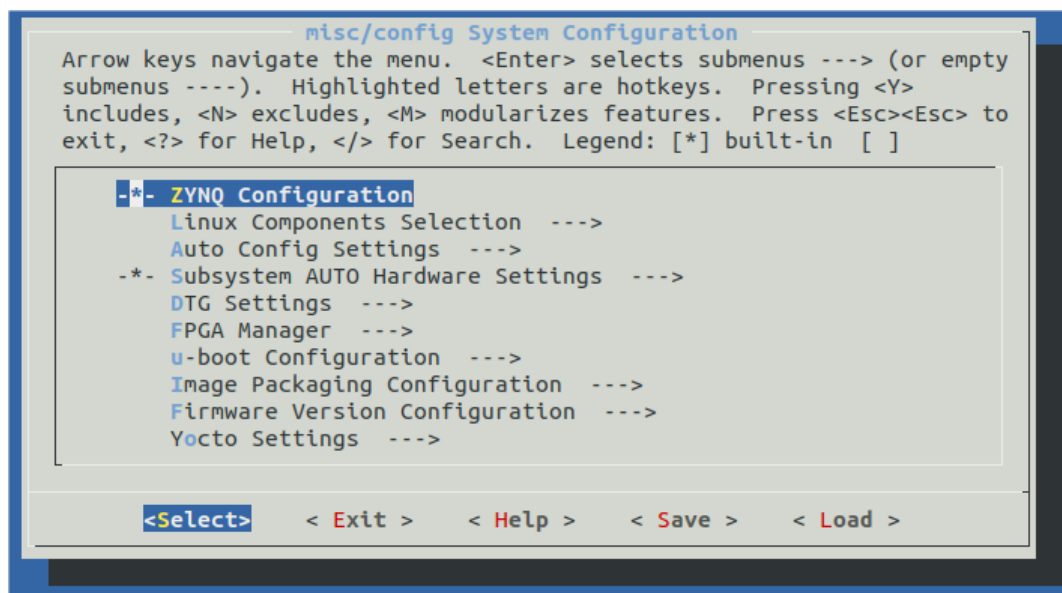


Figure 6.7: *Petalinux configuration window*

- ★ Navigate in the menu and select **EXT (SD/eMMC/QSPI/SA TA/USB)** in **Image packaging configuration** → **root filesystem** type.
- ★ In order to create filesystem as **ext4** type we will need to add

”ext4” into root filesystem formats in **Image packing configuration** → **root filesystem** formats.

★ Type ”ext4” at the end of the existing line as shown in the figure.

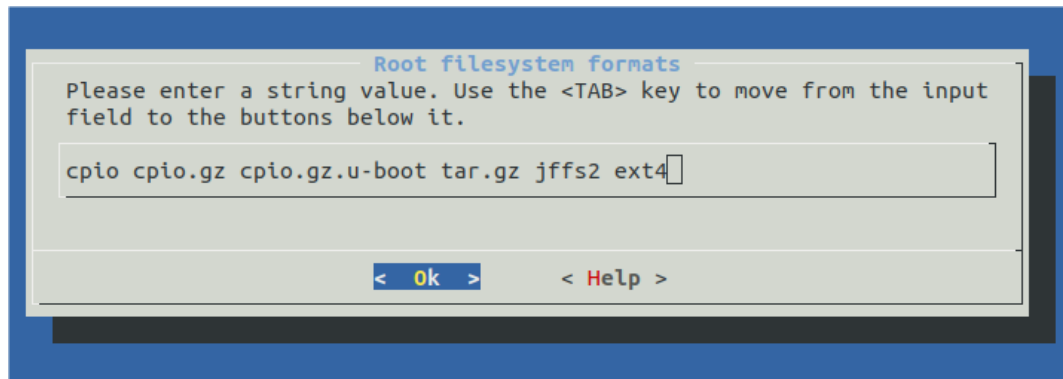


Figure 6.8: *Petalinux Root filesystem configuration formats window*

- **Build the Petalinux project. Insert:**

```
petalinux-build
```

This process could take several minutes.

6.2.3 Prepare Files for Platform Packaging

To facilitate the Vitis platform creation flow, we will create a folder named ”zcu104_custom_pkg” to store all the necessary files. Inside this folder, we will create a ”pfm” subfolder specifically for holding the platform creation source components.


- **Install sysroot**
 - Go to `/images/linux` directory.
 - Type ”`./sdk.sh -d [Install Target Dir]`” in the linux command terminal to install the PetaLinux SDK(Software Development Kit).

- Create "boot" directory and "image" directory inside "pfm" directory

```
cd zcu104_custom_pkg/pfm
mkdir boot
mkdir image
```

- Create BIF (linux.bif) to describe boot component structure for bootgen

A BIF file(linux.bif) is added to the "boot" directory with the contents shown below 6.9



```
/* linux */
the_ROM_image:
{
    [fsbl_config] a53_x64
    [bootloader] <fsbl.elf>
    [pmufw_image] <pmufw.elf>
    [destination_device=pl] <bitstream>
    [destination_cpu=a53-0, exception_level=el-3, trustzone] <bl31.elf>
    [destination_cpu=a53-0, exception_level=el-2] <u-boot.elf>
}
```

Figure 6.9: *Contents of linux.bif*

- Prepare the boot components for the SD card

- zynqmp_fsbl.elf
- pmufw.elf
- bl31.elf
- u-boot.elf

Note: These files are the sources of creating BOOT.BIN mentioned bin **linux.bif**.

- Prepare image directory

- boot.scr: script for u-boot initialization.

- system.dtb: the device tree blob that u-boot reads during boot to understand system setup.

- **Preparing SD card**

This article, "**Installing Ubuntu on Xilinx ZYNQ-7000 AP SoC Using Petalinux,**" by Chatura Niroshan, contains instructions on how to prepare an SD card for Petalinux booting. To read directly from the source, please go to the website specified in the citation [26]. The mentioned instruction will likewise be created here for the same reasons that most of UG1144 were duplicated there. The original creator is deserving of full credit for this.

The Linux programme *GParted* makes this task simple. Open a terminal in Ubuntu and enter the following command to launch *GParted*.

```
sudo gparted
```

The following command can be used to install *GParted* if it is not already present.

```
sudo apt-get install gparted
```

Connect the SD card to the PC using SD card reader and start *GParted* using above command. Follow the steps below to prepare the SD card.

→ Select the SD card in *GParted*.

→ Make sure it is unmounted and delete the partition of the SD card so that it displays 'unallocated' in *GParted*.

→ Right-click the unallocated space and create a new partition with the following settings.

- **Free Space Proceeding (MiB):** 4,
- **New Size (MiB):** 512,
- **File System:** FAT32,
- **Label:** BOOT.

Don't change other settings and click Add to finish.

→ Right-click the remaining unallocated space and create a new partition with following settings.

- **Free Space Proceeding (MiB):** 0,
- **Free Space Following(MiB):** 0,
- **File System:** ext4,
- **Label:** rootfs.

Don't change other settings and click Add to finish.

→ Apply all changes to create the partitions [26].

• Copying contents from host PC to SD card

→ Copy images files to SD card(BOOT.bin and image.ub to BOOT partition and extract rootfs.tar.bz2 into ROOTFS partion.)

→ Unmount the second partition of SD card (do not eject the card from the reader yet).

→ Copy contents from file into second partition on the SD card.

This can be done with the command:

```
$sudo tar xzf rootfs.tar.gz -C /media/{user}/ROOTFS/
$sync
$cp BOOT.bin image.ub /media/{user}/BOOT
```

NOTE! Make sure the selected destination partition corresponds to SD card's partition. This command could overwrite entire system partitions if used wrong.

→ Open *GParted*, select proper device from top right. When listing all partitions on SD Card a single partition might note an error message. Select this partition, right-click on it and select check.

→ Apply changes to check for filesystem errors. This will extend SD card's second partition to the full extent making it possible to use all this space on ZCU104 board.

→ SD Card is now ready to unmount and eject from the system.

To make sure the tutorial in this subsection has been completed successfully, Petalinux installation should be checked on Xilinx ZCU104 board.

To check the successful booting capability of Petalinux:

- Make sure ZCU104 board is configured as stated in ZCU104 getting started guide [19].
- If ZCU104 board is turned on, connect to ZCU104 board via USB_UART. Step by step guide can be found at [27] for host machines running Microsoft Windows 7 or newer or for host machine running Linux. In this project, a Linux host was used and after ZCU104 is configured, the Putty terminal can be opened using the command:

```
sudo putty
```

Configure the putty terminal as shown in figure 6.10.

Note that the device name varies by host and devices used. If Petalinux installation is successful a login screen appears in the terminal, as shown in figure 6.11. For logging in, the default username is "root" and the password is "root".

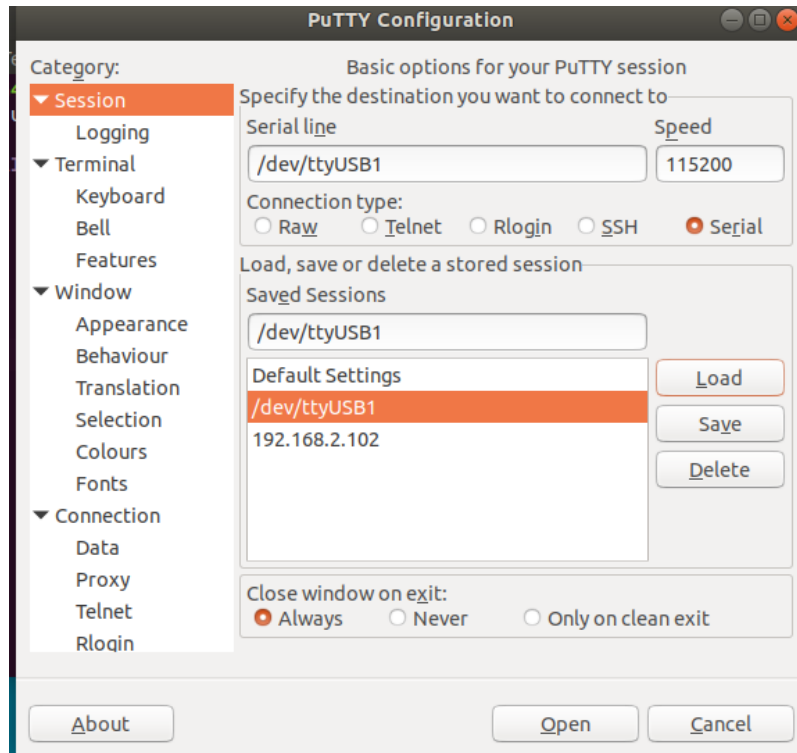


Figure 6.10: Screenshot of Putty Configuration Window

```

[ OK ] Started Getty on tty1.
[ OK ] Started Serial Getty on ttyPS0.
[ OK ] Reached target Login Prompts.
[ OK ] Started Target Communication Framework agent.
[ OK ] Reached target Multi-User System.
[ OK ] Started dpu-auto-config.service.
[ OK ] Reached target Graphical Interface.
Starting Record Runlevel Change in UTMP...
[ OK ] Finished Record Runlevel Change in UTMP.

PetaLinux 2022.1_release_S04190222 xilinx-zcu104-20221 ttyPS0

xilinx-zcu104-20221 login: root (automatic login)

root@xilinx-zcu104-20221:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 2E:D5:06:C8:6A:82
          inet addr:192.168.2.111  Bcast:192.168.2.255  Mask:255.255.255.0

```

Figure 6.11: Screenshot of terminal shortly after Petalinux has successfully booted greeting with a login screen.

- It is recommended to use **scp** for more intuitive file transfer between host and device. In Linux, this can be done by inserting:

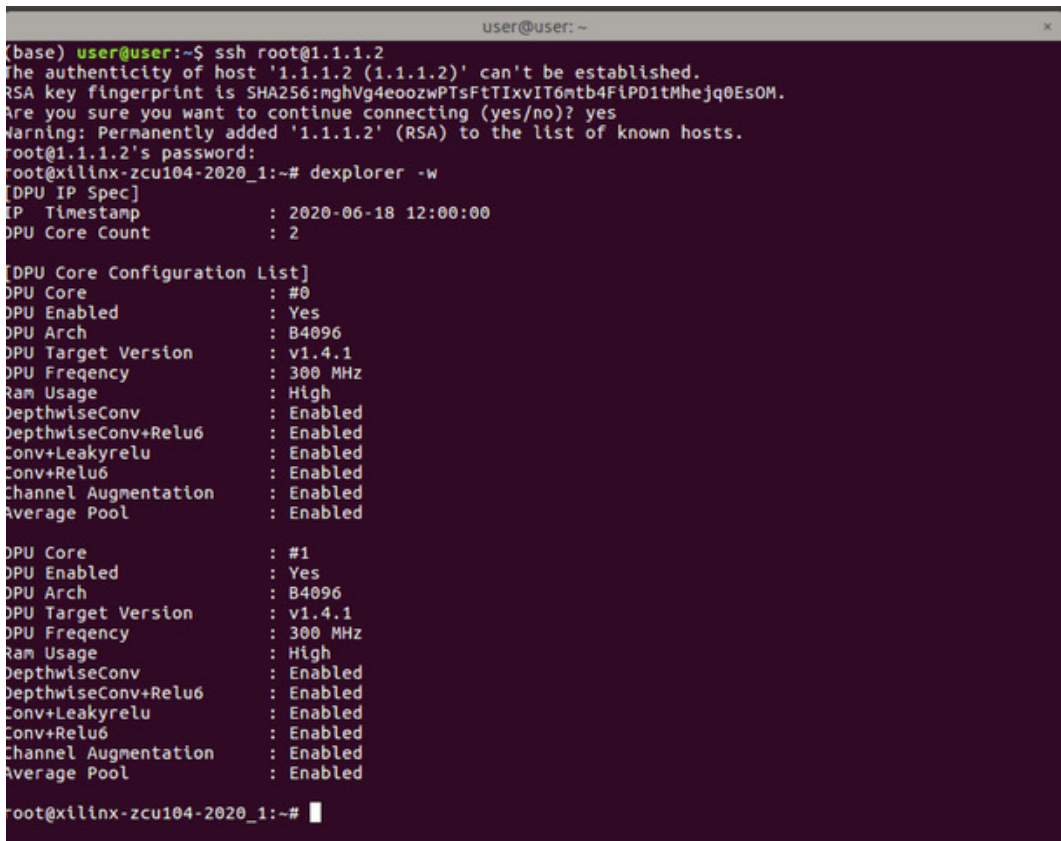
scp: //root@IP_ADDRESS

Note that IP_ADDRESS shall correspond to the IP address set to ZCU104 board and the network connection between the host and device has been established.

It is crucial to confirm that Petalinux kernel has identified DPU after a successful connection to the ZCU104 board. For detection insert:

```
dexplorer -w
```

To view DPU signature information for the selected board. If DPU is detected correctly then the terminal output should look something like in the figure 6.12.



```
(base) user@user:~$ ssh root@1.1.1.2
The authenticity of host '1.1.1.2 (1.1.1.2)' can't be established.
RSA key fingerprint is SHA256:mghVg4eoozwPTsFtTixvIT6mtb4FiPD1tMhejq0EsOM.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '1.1.1.2' (RSA) to the list of known hosts.
root@1.1.1.2's password:
root@xilinx-zcu104-2020_1:~# dexplorer -w
[DPU IP Spec]
IP Timestamp           : 2020-06-18 12:00:00
DPU Core Count         : 2

[DPU Core Configuration List]
DPU Core                : #0
DPU Enabled             : Yes
DPU Arch                : B4096
DPU Target Version      : v1.4.1
DPU Frequency           : 300 MHz
Ram Usage               : High
DepthwiseConv           : Enabled
DepthwiseConv+Relu6     : Enabled
Conv+Leakyrelu          : Enabled
Conv+Relu6              : Enabled
Channel Augmentation    : Enabled
Average Pool            : Enabled

DPU Core                : #1
DPU Enabled             : Yes
DPU Arch                : B4096
DPU Target Version      : v1.4.1
DPU Frequency           : 300 MHz
Ram Usage               : High
DepthwiseConv           : Enabled
DepthwiseConv+Relu6     : Enabled
Conv+Leakyrelu          : Enabled
Conv+Relu6              : Enabled
Channel Augmentation    : Enabled
Average Pool            : Enabled

root@xilinx-zcu104-2020_1:~#
```

Figure 6.12: *screenshot of DPU Signature Viewed ZCU104 with DExplorer*

6.3 Application on DPU

In this subsection, the focus is on implementing an image classification application on the ZCU104 board using four different models. The application will be accelerated using the Xilinx DPU, and the DPU configuration file compilation will be performed on a host PC.

While the majority of the work is done by the author, certain sections have been adapted from various Xilinx guides.

6.3.1 Docker Setup on the Host

To set up Docker on the host, follow these steps:

- **Clone the Vitis AI repository to obtain the examples, reference code, and scripts**

```
[Host]$ git clone https://github.com/Xilinx/Vitis-AI
[Host]$ cd Vitis-AI
```

If Docker is not already installed on the machine, please consult the official documentation <https://docs.docker.com/engine/install/> for guidance on installation.

- **Download the latest Vitis AI Docker**

```
[Host]$ docker pull xilinx/vitis-ai-<pytorch/tensorflow
/tensorflow2>-cpu:latest
```

6.3.2 Model Deployment

In this section, the fine-tuned model saved as "model.h5" will be quantized using Docker. After quantization, the resulting quantized model will be compiled to obtain the xmodel file necessary for deployment.

- **Quantising the model**
 - **Launch the docker image**

```
[Host]$ ./docker_run.sh xilinx/vitis-ai-tensorflow
-cpu:latest
```

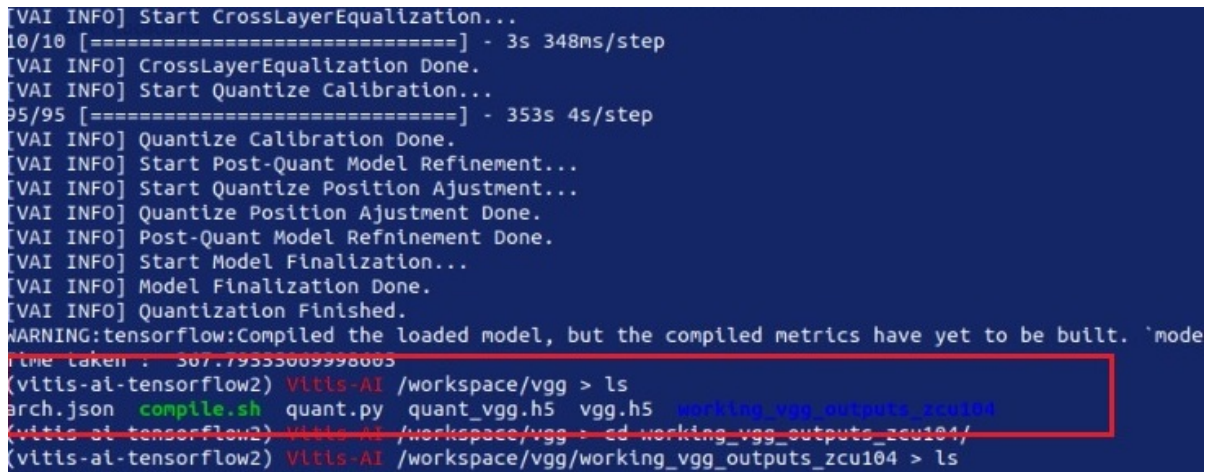
- **Quantize the model**

```
[Docker]$ conda activate vitis-ai-tensorflow
```

The "Conda" environment is set. Then run the source code "quant.py"

```
[Docker]$ python quant.py
```

The quantization calibration process will begin and typically takes a few minutes to finish. Once completed, the quantized model(quant.h5) will be accessible in the "[model_directory]/quantized" folder, as depicted in the figure 6.13.



```
[VAI INFO] Start CrossLayerEqualization...
10/10 [=====] - 3s 348ms/step
[VAI INFO] CrossLayerEqualization Done.
[VAI INFO] Start Quantize Calibration...
95/95 [=====] - 353s 4s/step
[VAI INFO] Quantize Calibration Done.
[VAI INFO] Start Post-Quant Model Refinement...
[VAI INFO] Start Quantize Position Ajustment...
[VAI INFO] Quantize Position Ajustment Done.
[VAI INFO] Post-Quant Model Refinement Done.
[VAI INFO] Start Model Finalization...
[VAI INFO] Model Finalization Done.
[VAI INFO] Quantization Finished.
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. 'mode
Time taken : 307.79555009998005
(vitis-ai-tensorflow2) Vitis-AI /workspace/vgg > ls
arch.json  compile.sh  quant.py  quant_vgg.h5  vgg.h5  working_vgg_outputs_zcu104
(vitis-ai-tensorflow2) Vitis-AI /workspace/vgg > cd working_vgg_outputs_zcu104/
(vitis-ai-tensorflow2) Vitis-AI /workspace/vgg/working_vgg_outputs_zcu104 > ls
```

Figure 6.13: Screenshot of Quantized model file

- **Compiling the Model**

After quantization, the model must be compiled to obtain the xmodel, which enables its execution on an FPGA. The following command will be run:

```
[Docker]$ compile.sh
```

Upon executing this command, the model will be compiled, and the resulting compiled xmodel file will be saved in the "[model_directory]/compiled" folder as shown in figure 6.14.

6.3.3 Deployment on Edge boards

To configure the ZCU104 board for access by the host PC and establish a serial connection, follow these steps:


```
(vitis-ai-tensorflow2) Vitis-AI /workspace/vgg > ls
arch.json compile.sh quant.py quant_vgg.h5 vgg.h5 working_vgg_outputs_zcu104/
(vitis-ai-tensorflow2) Vitis-AI /workspace/vgg > cd working_vgg_outputs_zcu104/
(vitis-ai-tensorflow2) Vitis-AI /workspace/vgg/working_vgg_outputs_zcu104 > ls
md5sum.txt meta.json vgg.xmodel
(vitis-ai-tensorflow2) Vitis-AI /workspace/vgg/working_vgg_outputs_zcu104 > ls -la
total 14824
drwxr-xr-x 2 vitis-ai-user vitis-ai-group 4096 May 28 11:17 .
drwxr-xr-x 3 vitis-ai-user vitis-ai-group 4096 Jun 7 04:30 ..
-rw-r--r-- 1 vitis-ai-user vitis-ai-group 33 May 28 11:17 md5sum.txt
-rw-r--r-- 1 vitis-ai-user vitis-ai-group 168 May 28 11:17 meta.json
-rw-r--r-- 1 vitis-ai-user vitis-ai-group 15162362 May 28 11:17 vgg.xmodel
(vitis-ai-tensorflow2) Vitis-AI /workspace/vgg/working_vgg_outputs_zcu104 > cd ..
(vitis-ai-tensorflow2) Vitis-AI /workspace/vgg > ls -la
total 231476
drwxr-xr-x 3 vitis-ai-user vitis-ai-group 4096 Jun 7 04:30 .
drwxrwxr-x 20 vitis-ai-user vitis-ai-group 4096 May 28 09:20 ..
-rw-r--r-- 1 vitis-ai-user vitis-ai-group 37 May 28 10:26 arch.json
-rwxr-xr-x 1 vitis-ai-user vitis-ai-group 202 May 28 11:17 compile.sh
-rw-r--r-- 1 vitis-ai-user vitis-ai-group 2496 Jun 7 04:30 quant.py
-rw-r--r-- 1 vitis-ai-user vitis-ai-group 59387920 Jun 7 04:38 quant_vgg.h5
-rw-rw-r-- 1 vitis-ai-user vitis-ai-group 177603680 May 26 12:41 vgg.h5
```

Figure 6.14: Screenshot of Compiled xmodel file

- Turn on the FPGA and access it through the UART port: The ZCU104 board is connected to the host PC using a USB cable.
- Power on the ZCU104 board.
- Open a terminal or command prompt on the host PC.
- Identify the serial port associated with the ZCU104 board. On Linux, the user can use the `ls "/dev/tty*"` command to list available serial ports.
- Once the serial port has been identified, configure the serial communication settings. Use a terminal emulator program such as PuTTY terminal to establish a serial connection. Set the baud rate, data bits, parity, stop bits, and flow control according to the board's specifications or default settings. Refer to 6.10.
- Open the serial connection. This will establish a connection between the host PC and the ZCU104 board, given in [28].
- To set a static IP address of the ZCU104 board, type:

```
ifconfig eth0 192.168.2.99 netmask 255.255.255.0
```

Once the DPU is integrated into the FPGA fabric, it is essential to have the relevant libraries to effectively utilize it. These libraries provide the necessary functions, APIs, and tools to facilitate the execution and management of the DPU, enabling seamless integration with the rest of the system.

- Transfer the package onto the FPGA via SCP:

```
scp /vitis-ai_v1.1_dnndk.tar.gz root@192.168.2.99:~/
```

- Then, using the terminal through the board:

```
tar -xzvf vitis-ai_v1.1_dnndk.tar.gz
cd vitis-ai_v1.1_dnndk
./install.sh
```

- Install Jupyter Notebook environment in the board

```
pip install jupyter notebook
```

Once the installation is complete, use the following command to generate a configuration file

```
jupyter notebook --generate-config
```

”/jupyter/jupyter_notebook_config.py” configuration file is generated. The following ip settings can be added to the file to make it accessible from the host PC by entering the following command:

```
search_str="# c.NotebookApp.ip = 'localhost'"
set_str="c.NotebookApp.ip = '0.0.0.0'"
sed -i -e "/^$search_str$/a $set_str"
~/jupyter/jupyter_notebook_config.py
```

Since it is used for access from the host PC side, memorize the IP address. Once the IP address has been obtained, start Jupyter Notebook by typing:

- `jupyter notebook --allow-root`

For further information, refer to [29] & [30]

In the Jupyter Notebook file, upload the dataset and the .xmodel file to the Test.ipynb file. Then, execute the image classification process using the following steps:

- **Load the dataset:** Import necessary libraries and load the Chest X-ray dataset into the notebook. Preprocess the dataset as required, including resizing images and normalizing pixel values.
- **Load the DPU model:** Load the .xmodel file that contains the trained DPU model. Use the appropriate function to initialize the DPU model and configure it for inference.
- **Perform image classification:** Iterate through the test images in the dataset and pass them through the DPU model for classification. Retrieve the predicted labels for each image.
- **Evaluate the results:** Compare the predicted labels with the ground truth labels of the test images. Calculate evaluation metrics such as accuracy, precision, recall, and F1 score to assess the performance of the image classification model as shown in section 7.1.
- **Visualize the results:** Display some sample images along with their predicted labels to visually inspect the accuracy of the model's classifications.

Ensure that the necessary dependencies, such as deep learning frameworks and image processing libraries, are installed and imported into the Jupyter Notebook.

Chapter 7

Result Analysis

In this chapter, we will dig into a comprehensive analysis of the results obtained from our work. We will explore the findings in three main segments:

1. Proof of concept and accuracy analysis, where we will assess the validity and precision of our proposed approach. 7.2
2. Model size and resource utilisation, where the analysis of model size and resource utilization consistently demonstrates a reduction in size for each model, resulting in improved resource efficiency. 7.3
3. Performance and evaluation of hardware acceleration, where we will evaluate the efficiency and effectiveness of hardware acceleration on deep learning model inference performance. 7.4

Through these detailed analyses, we aim to provide a comprehensive understanding of the outcomes and implications of our work.

7.1 Dataset

The dataset was divided into training and testing sets, organized into separate folders by class labels. Image data generators were used to preprocess the data using the "flow from directory" approach. The evaluation on the DPU utilized the same test dataset with three classes: COVID (1st class), Normal (2nd class), and Viral Pneumonia (3rd class). The accompanying figures illustrate accurate classification by each model. Furthermore, it is evident from the section 7.4 that the execution time achieved by the DPU is significantly lower compared to CPU execution. Further details regarding these findings are elaborated upon in the subsequent section 7.4.

The included figure 7.1 displays one of the 'COVID' class dataset samples utilized in this thesis.

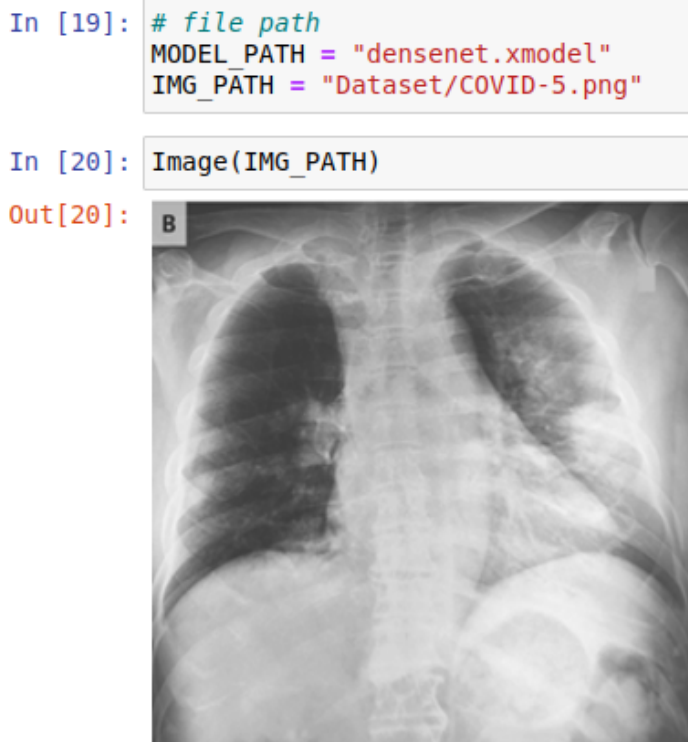


Figure 7.1: *Screenshot of COVID dataset image*

For class 1 (COVID), the corresponding figure 7.2 clearly illustrates that the model has accurately predicted the samples belonging to the 'COVID' class. The model's classification results align with the ground truth labels, indicating its ability to correctly identify COVID cases within the dataset. This successful prediction showcases the effectiveness of the model in distinguishing COVID instances from other classes.

```
print(resultList)
print("The Image belongs to class:",resultIdx)

end_time = time.time()
execution_time = end_time - start_time
print(f"The execution time is: {execution_time}")

[[[9.9823844e-01 1.5007908e-03 2.6079832e-04]]]
The Image belongs to class: 0
The execution time is: 0.010879278182983398
```

Figure 7.2: Screenshot of COVID class prediction

The provided figure 7.3 displays one of the 'Normal' class dataset images utilized in this thesis.

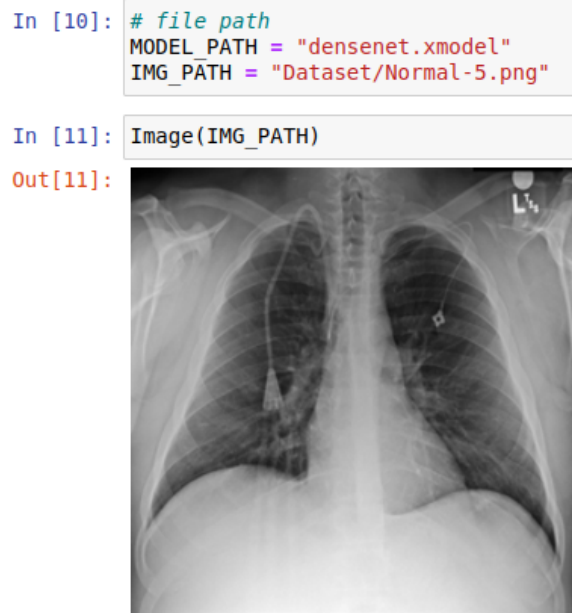


Figure 7.3: Screenshot of Normal class dataset image

For class 2(Normal), the corresponding figure 7.4 showcases the accurate prediction of the 'Normal' class by the model.

```
print(resultList)
print("The Image belongs to class:",resultIdx)

end_time = time.time()
execution_time = end_time - start_time
print(f"The execution time is: {execution_time}")

[[[0.01786798 0.9755587 0.00657326]]]
The Image belongs to class: 1
The execution time is: 0.009871959686279297
```

Figure 7.4: *Screenshot of Normal class prediction*

The below figure 7.5 represents one of the images from the 'Viral Pneumonia' class dataset images utilized in this thesis.

```
In [4]: # file path
MODEL_PATH = "densenet.xmodel"
IMG_PATH = "Dataset/Viral Pneumonia-5.png"
```

```
In [5]: Image(IMG_PATH)
```

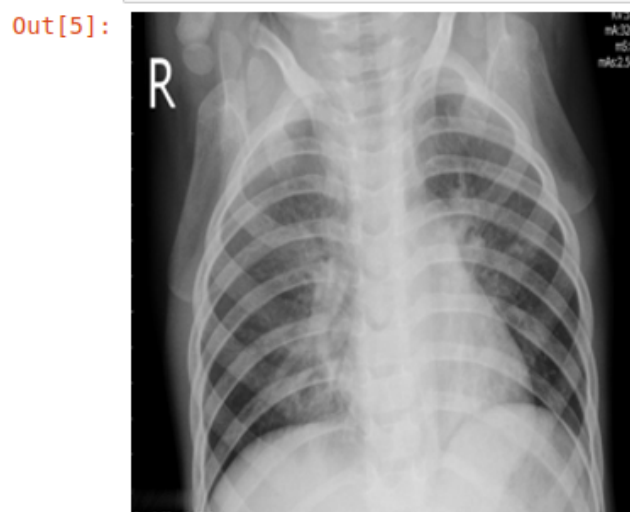


Figure 7.5: *Screenshot of Viral Pneumonia class dataset image*

For the Viral Pneumonia class (3rd class), the figure 7.6 clearly illustrates that the model has accurately predicted and classified instances belonging to the 'Viral Pneumonia' category. This demonstrates the model's proficiency in correctly identifying and distinguishing images associated with Viral Pneumonia within the dataset.

```
print(resultList)
print("The Image belongs to class:",resultIdx)

end_time = time.time()
execution_time = end_time - start_time
print(f"The execution time is: {execution_time}")
```

```
[[[0.00316265 0.00316265 0.99367476]]]
The Image belongs to class: 2
The execution time is: 0.010532617568969727
```

Figure 7.6: Screenshot of Viral Pneumonia class prediction

7.2 Proof of Concept and Accuracy Analysis

This section demonstrates that the proposed solution successfully achieves the intended goal of accelerating inference time on the DPU over the CPU for DNN image classification models. The evaluation process involved utilizing a ready-made open-source solution to calculate the mean average precision (mAP).

To adapt the evaluation to the image classification task using chest X-ray dataset containing labels for Covid, Viral Pneumonia, and Normal cases, modifications were made to the dataset and evaluation setup.

Trained/Fine-tuned the pre-trained models over Colab GPU notebook and then used the Vitis-AI docker platform to quantised the fine-tuned model weights which then is converted to Xmodel using the provided compiling tool within Vitis-AI docker platform itself. Which xmodel is later transferred to the custom-made DPU hardware platform for evaluating the inference.

The adapted evaluation process comprehensively assesses DNN im-

age classification models' performance and accuracy when utilizing DPU for accelerated inference time. This validates the effectiveness of the proposed solution for the given problem statement.

• RESNET-50

The table 7.1 comparing average accuracy values for the three classes in CPU and DPU for RESNET demonstrates that DPU achieves higher accuracy compared to CPU, indicating the effectiveness of hardware acceleration in improving classification performance.

Class Name	mAp(in CPU)	mAp(in DPU)
Covid	0.82	0.85
Normal	0.69	0.95
Viral Pneumonia	0.95	1.00

Table 7.1: Comparison of average accuracy values per class for ResNet-50 model

• DENSENET-169

The table 7.2 comparing the average precision values of the three classes in CPU and DPU for the DENSENET model. DPU achieves higher precision for all classes, indicating superior accuracy and performance compared to CPU inference.

Class Name	mAp(in CPU)	mAp(in DPU)
Covid	0.95	1.00
Normal	0.91	0.95
Viral Pneumonia	0.95	1.00

Table 7.2: Comparison of average accuracy values per class for DenseNet model

- **VGG-16**

From table 7.3 comparing the average precision values of the three classes in CPU and DPU for VGG, it can be concluded that utilizing DPU improves the precision significantly for all three classes, indicating the effectiveness of hardware acceleration for image analysis.

Class Name	mAp(in CPU)	mAp(in DPU)
Covid	0.97	0.98
Normal	0.74	1.00
Viral Pneumonia	0.95	0.95

Table 7.3: *Comparison of average accuracy values per class for Vgg-16 model*

- **MobileNet**

From table 7.4, it can be concluded that when using MobileNet on CPU, the average accuracy for the COVID class is 0.69, while it is 1.00 when using DPU. However, for other two classes (Normal and Viral Pneumonia), the average accuracy drops to 0 when using DPU.

Class Name	mAp(in CPU)	mAp(in DPU)
Covid	0.69	1.00
Normal	1.00	0
Viral Pneumonia	0.91	0

Table 7.4: *Comparison of average accuracy values per class for MobileNet*

This suggests that the DPU is highly effective in accurately classifying COVID cases, achieving a perfect average accuracy of 1.00.

However, it struggles to accurately classify the other two cases, resulting in an average accuracy of 0 for these classes.

Further analysis and investigation may be required to understand the specific reasons behind the lower accuracy for the Normal and Viral Pneumonia classes when utilizing the DPU in the MobileNet model. Factors such as model complexity, training data, or architectural limitations may play a role in this observation.

7.3 Model Size and Resource Utilization

From the below table 7.5, the analysis of size reduction after quantizing and generating xmodels for four different models, **RESNET-50**, **DenseNet-169**, **VGG-16**, and **MobileNet** reveals a consistent decrease in size for each model, leading to less resource utilization.

	Size of H5 model file	Quantized model size	Compiled xmodel size
ResNet50	286.8MB	95.8MB	25.1MB
DenseNet-169	155.9MB	53.9MB	14.2MB
Vgg-16	177.6MB	59.4MB	15.2MB
MobileNet	40.7MB	13.7MB	3.7MB

Table 7.5: *Comparison of Model Sizes and Resource Utilization*

Quantization, applied to all four models, reduces the precision of model parameters and activations, resulting in smaller model sizes compared to their original h5 files. This reduction occurs because quantized models require fewer bits to represent the same information, resulting in more compact representations. Consequently,

less resource utilization is achieved, enabling efficient storage, faster model loading times, and improved memory management for all four models.

A further reduction in size is observed when generating xmodels tailored for DPU implementation for each of the four models. The optimization process eliminates unnecessary components and focuses on extracting essential elements for inference on DPU. As a result, xmodels for **RESNET-50**, **DenseNet-169**, **VGG-16**, and **MobileNet** exhibit even smaller sizes compared to their respective quantized models. This further contributes to less resource utilization, allowing for improved utilization of computational resources on the DPU.

The consistent reduction in size for all four models, from the original h5 files to the quantized models and subsequently to the xmodels, underscores the advantages of both quantization and DPU implementation. These size reductions not only facilitate efficient storage and faster loading times but also enhance resource utilization, including memory and computational power. Overall, the analysis demonstrates that quantization and DPU implementation offer effective strategies for reducing the size of deep neural network models across different models, leading to less resource utilization and enabling efficient inference on hardware platforms.

7.4 Hardware Acceleration: Performance & Evaluation

In this section, we will evaluate the performance of our models and analyze the acceleration gains obtained by leveraging the DPU com-

pared to CPU. We will measure the inference times of each model on DPU and compare them to the corresponding CPU inference times. By analyzing the acceleration achieved, we can assess the effectiveness of utilizing the DPU for hardware acceleration. We will examine the acceleration gains for each model based on their individual inference times. This evaluation will provide us with valuable insights into the performance improvements achieved by utilizing the DPU for inference tasks.

	ResNet-50	DenseNet-169	Vgg-16
Inference time in CPU (in ms)	0.90	1.57	0.17
Inference time in DPU (in ms)	0.010	0.09	0.013
Acceleration	90x	17.45x	13.02x

Table 7.6: *Inference Time Comparison between CPU and DPU*

Table 7.6 illustrates the comparison of inference times for various models when executed on both the CPU and DPU platforms. The CPU inference time represents the time taken by the model to perform inference solely on the CPU, while the DPU inference time corresponds to the execution time when leveraging the DPU hardware acceleration.

- **ACCELERATION CALCULATION:**

To determine the acceleration achieved by utilizing DPU, the ratio of CPU inference time to DPU inference time can be calculated as :

$$\text{Acceleration} = \text{CPU Inference Time} / \text{DPU Inference Time} \quad (7.1)$$

This ratio provides an indication of the speedup achieved by leveraging the DPU hardware acceleration. The following list provides the acceleration calculations for each model:

- **RESNET-50:**

- Acceleration = 0.9 ms / 0.01 ms = **90**
- The DPU achieves an acceleration of **90 times** compared to the CPU.

- **DENSENET-169:**

- Acceleration = 1.57 / 0.09 ms = **17.45**
- The DPU achieves an acceleration of approximately **17 times** compared to the CPU.

- **VGG-16:**

- Acceleration = 0.17 ms / 0.013 ms = **13.02**
- The DPU achieves an acceleration of almost **13 times** compared to the CPU.

The acceleration values provide a clear measure of the speedup achieved through the utilization of the DPU for inference, surpassing the performance of the CPU alone. These values serve as a testament to the remarkable enhancements in system performance.

The higher the acceleration value, the more substantial the performance improvement. In this context, the acceleration values obtained signify the impressive reduction in execution time achieved by harnessing the power of the DPU’s hardware acceleration capabilities. This translates into a significant boost in efficiency, enabling faster and more efficient inference tasks compared to relying solely on the CPU.

By leveraging the DPU, the inference process is accelerated to a remarkable degree, offering a substantial improvement over traditional CPU-based inference. These acceleration values emphasize the remarkable strides made in performance and efficiency when employing the DPU for demanding inference tasks.

From the observed results, it can be concluded that the DPU architecture provides significant acceleration for deep neural network models compared to CPU. The ResNet-50 model achieves the highest acceleration of 90 times, indicating its compatibility with the DPU and its ability to exploit its hardware acceleration capabilities effectively. On the other hand, the VGG-16 model shows the lowest acceleration of 13 times, suggesting that it may not fully leverage the potential of the DPU for speeding up computations. These findings highlight the varying degrees of compatibility and performance gains achieved by different models when deployed on the DPU, emphasizing the importance of selecting suitable models for optimal acceleration.

Chapter 8

Conclusions

In a nutshell, this thesis has demonstrated the effectiveness of hardware acceleration using reconfigurable architectures for deep neural network-based image analysis. Through extensive experiments and analyses, it has been established that the integration of software and hardware components, facilitated by tools like Vitis AI and Vivado, enables seamless deployment and execution of deep neural networks on FPGA platforms. The utilization of various pre-trained models, including RESNET-50, DenseNet-169, VGG-16, and MobileNet, has showcased the versatility and applicability of the proposed solution. Furthermore, the analysis of model size and resource utilization has consistently shown reductions in size, leading to improved resource efficiency. The evaluation of performance has revealed substantial speed improvements, with accelerations ranging from 13 to 90 times compared to CPU execution. This research contributes to the field by highlighting the potential of reconfigurable architectures in image analysis tasks, while also suggesting avenues for future research to address technical challenges, enhance accuracy benchmarking, and explore additional model architectures. Overall, this thesis provides valuable insights for utilizing FPGA-based accelerators in efficient and high-performance deep neural network inference.

Chapter 9

Future Work

This thesis’s successful completion opens doors for future research in hardware-accelerated deep learning for image analysis, presenting numerous opportunities for further exploration and improvement.

One such area is the integration of more complex models beyond the ones studied in this thesis, such as Transformer-based architectures or larger-scale CNNs. By examining the acceleration and performance of these models using hardware acceleration, the proposed solution’s capabilities can be expanded in more room for utility.

Additionally, future research can explore DPU-based hardware acceleration for real-time video processing, expanding the proposed solution’s practicality and applicability to various multimedia applications beyond image analysis.

Future research can explore multi-model inference, running multiple DL models simultaneously on the DPU. Optimizing hardware resources for efficient multi-model inference enables the deployment of complex image analysis pipelines.

Future research can drive advancements in hardware-accelerated DL for image analysis, enabling efficient and scalable solutions.

Chapter 10

Summary

This thesis aims to leverage the Vitis Unified Software Platform by Xilinx to accelerate the inference of image classification applications on the ZCU104 board. The initial task involves conducting a comprehensive analysis of the Vitis platform, understanding its capabilities and benefits. Subsequently, four well-researched and easily modifiable models, namely RESNET-50, DenseNet-169, Vgg-16, and MobileNet, have been selected for the image classification task. These models were chosen due to their established performance and ease of customization.

After conducting the initial experiments, the author came to the realization that Vitis, although a powerful software platform, is not specifically designed for artificial neural network inference. Recognizing the need for a more specialized solution, the author then turned to experimenting with Vitis AI.

After numerous attempts and iterations, a satisfactory solution was ultimately discovered. The proposed solution incorporates Xilinx Vitis AI, along with a modified accelerator and the Petalinux operating system, to successfully execute image classification tasks on the ZCU104 board, which is mentioned in chapter 3.

The analysis section 7 of this thesis is divided into three major segments to comprehensively assess the proposed approach. Firstly, the proof of concept and accuracy analysis will validate the effectiveness and precision of the proposed approach. Secondly, the model size and resource utilization analysis will demonstrate a consistent reduction in model size, leading to enhanced resource efficiency. Lastly, the performance and evaluation of hardware acceleration will evaluate the efficiency and effectiveness of hardware acceleration on the inference performance of DL models. These segments collectively provide a comprehensive evaluation of the proposed approach, covering its validity, resource optimization, and acceleration capabilities.

Chapter 6 contains is a guide on replicating a proposed solution with tools listed in chapter 4.

In Chapter 9, the author identifies several important tasks that can further enhance the proposed solution.

The proposed solution holds significant potential for educational purposes, enabling the inference of various artificial neural networks without requiring developers to possess expertise in hardware design. This approach offers a user-friendly solution that simplifies the development flow, allowing for faster prototyping of new systems and solutions. By eliminating the need for specialized hardware design knowledge, the proposed solution empowers developers to focus on exploring innovative applications and rapidly iterating on their ideas. This accelerated development flow facilitates the exploration of new possibilities and encourages the efficient creation of prototypes for potential systems and solutions.

Bibliography

- [1] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).
- [2] Fahri Esen, Ali Degirmenci, and Omer Karal. “Implementation of the Object Detection Algorithm (YOLOV3) on FPGA”. In: *2021 Innovations in Intelligent Systems and Applications Conference (ASYU)*. 2021, pp. 1–6. DOI: 10.1109/ASYU52992.2021.9599073.
- [3] Jurgen Vandendriessche, Bruno Da Silva, and Abdellah Touhafi. “Frequency Evaluation of the Xilinx DPU Towards Energy Efficiency”. In: *IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society*. 2022, pp. 1–6. DOI: 10.1109/IECON49645.2022.9968811.
- [4] Jiang Zhu et al. “An Efficient Task Assignment Framework to Accelerate DPU-Based Convolutional Neural Network Inference on FPGAs”. In: *IEEE Access* 8 (2020), pp. 83224–83237. DOI: 10.1109/ACCESS.2020.2988311.
- [5] Aman Sharma, Vijander Singh, and Asha Rani. “Implementation of CNN on Zynq based FPGA for Real-time Object Detection”. In: *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2019, pp. 1–7. DOI: 10.1109/ICCCNT45670.2019.8944792.
- [6] Samuel C Leach. “Analysis of Hardware Accelerated Deep Learning and the Effects of Degradation on Performance”. In: (2021).
- [7] Xiaofan Zhang et al. “DNNE Explorer: a framework for modeling and exploring a novel paradigm of FPGA-based DNN accelerator”. In: *Proceedings of the 39th International Conference on Computer-Aided Design*. 2020, pp. 1–9.
- [8] Jin Wang and Shenshen Gu. “FPGA Implementation of Object Detection Accelerator Based on Vitis-AI”. In: *2021 11th International Conference on Information Science and Technology (ICIST)*. 2021, pp. 571–577. DOI: 10.1109/ICIST52614.2021.9440554.

- [9] Francesco Restuccia and Alessandro Biondi. “Time-Predictable Acceleration of Deep Neural Networks on FPGA SoC Platforms”. In: *2021 IEEE Real-Time Systems Symposium (RTSS)*. 2021, pp. 441–454. DOI: 10.1109/RTSS52674.2021.00047.
- [10] Yufan Lu et al. “FPGA based Adaptive Hardware Acceleration for Multiple Deep Learning Tasks”. In: *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoc)*. 2021, pp. 204–209. DOI: 10.1109/MCSoc51149.2021.00038.
- [11] Yutian Lei et al. “An Effective Design to Improve the Efficiency of DPUs on FPGA”. In: *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. 2020, pp. 206–213. DOI: 10.1109/ICPADS51040.2020.00036.
- [12] Xilinx. *Vitis Unified Software Platform*. Online accessed: July. 2022. URL: <https://www.xilinx.com/products/design-tools/vitis.html>.
- [13] Xilinx AMD. *Vitis AI User Guide*. Online accessed: September. 2022. URL: <https://docs.xilinx.com/r/en-US/ug1414-vitis-ai>.
- [14] Xilinx AMD. *Vitis AI Version 1.4*. Online accessed: August. 2022. URL: <https://github.com/Xilinx/Vitis-AI/tree/1.4>.
- [15] Xilinx. *DPUCZDX8G for Zynq UltraScale+ MPSoCs Product Guide (PG338)*. Online accessed: July. 2022. URL: <https://docs.xilinx.com/r/en-US/pg338-dpu/Customizing-and-Generating-the-Core-in-the-Vitis-IDE>.
- [16] *DPU Integration*. Online accessed: July. 2022. URL: <https://xilinx.github.io/Vitis-AI/docs/workflow-system-integration.html>.
- [17] Wikipedia contributors. *Xilinx — Wikipedia, The Free Encyclopedia*. 2022. URL: <https://en.wikipedia.org/w/index.php?title=Xilinx&oldid=950507093>.
- [18] *Model Quantization and Compilation*. Online accessed: September. 2022. URL: <https://xilinx.github.io/Vitis-AI/docs/workflow-model-development.html>.
- [19] Xilinx. *ZCU104 User Guide*. Online accessed: October. 2022. URL: <https://docs.xilinx.com/v/u/en-US/ug1267-zcu104-eval-bd>.
- [20] Wikipedia contributors. *Vivado Design Suite UserGuide Release Notes, Installation, and Licensing*. UG973 February 21, 2021. 2022. URL: <https://docs.xilinx.com/r/2020.2-English/ug973-vivado-release-notes-install-license/Revision-History>.

- [21] Wikipedia contributors. *PetaLinux User Guide*. UG1144, November 24, 2020. 2022. URL: <https://docs.xilinx.com/r/2020.2-English/ug1144-petalinux-tools-reference-guide/Revision-History>.
- [22] Xilinx. *Frequently asked Questions*. Online accessed: October. 2022. URL: https://support.xilinx.com/s/question/0D52E00006hplLuSAI/devicetree-error-in-petalinux-zcu104-board-for-bram?language=en_US.
- [23] Xilinx. *Custom Hardware Platform on ZCU104*. Online accessed: August. 2022. URL: https://xilinx.github.io/Vitis-Tutorials/2022-1/build/html/docs/Vitis_Platform_Creation/Design_Tutorials/02-Edge-AI-ZCU104/README.html.
- [24] Luu Nguyen. *Hardware Platform on ZCU104*. Online accessed: September. 2022. URL: <https://github.com/luunguyen97/DPU-TRD-ZCU104>.
- [25] Xilinx. *Vitis Software Platform Customization*. Online accessed: December. 2022. URL: https://xilinx.github.io/Vitis-Tutorials/2021-1/build/html/docs/Vitis_Platform_Creation/Introduction/02-Edge-AI-ZCU104/step2.html.
- [26] Chathura Niroshan. *Xilinx ZYNQ-7000 AP SoC Using Petalinux*. Online accessed: September. 2022. URL: <https://medium.com/developments-and-implementations-on-zynq-7000-ap/install-ubuntu-16-04-lts-on-zynq-zc702-using-petalinux-2016-4-e1da902eaff7>.
- [27] Xilinx. *Running Application on Board*. Online accessed: December. 2022. URL: https://xilinx.github.io/Vitis-Tutorials/2020-2/docs/build/html/docs/Vitis_Platform_Creation/Introduction/02-Edge-AI-ZCU104/step4.html.
- [28] *Setting up Host*. URL: <https://docs.xilinx.com/r/1.1-English/ug1414-vitis-ai/Using-the-Ethernet-Interface>.
- [29] *Running model on the DPU*. URL: <https://beetlebox.org/vitis-ai-using-tensorflow-and-keras-tutorial-part-9/>.
- [30] *Install Jupyter Notebook on edge boards*. URL: https://www.pixela.co.jp/products/pickup/dev/ai/vitisai_ai_2_env_en.html.