UNIVERSITY of RWANDA

ACEIoT

Website: www.aceiot.ur.ac.rw
Mail: aceiot@ur.ac.rw

COLLEGE OF SCIENCE AND TECHNOLOGY

AFRICAN CENTER OF EXCELLENCE IN INTERNET OF THINGS (ACEIoT)

Designing an FPGA-based System-on-Chip for X-ray Image Analysis and Tuberculosis Detection

A dissertation submitted in partial fulfillment of the requirements for the award of Master's of Science degree in Internet of Things: Embedded Computing Systems

Submitted By:

Name: Arthur Nathaniel Mwang'onda (Ref. No:221030693)

November 2023

UNIVERSITY of RWANDA

ACEIoT

Website: www.aceiot.ur.ac.rw

COLLEGE OF SCIENCE AND TECHNOLOGY

AFRICAN CENTER OF EXCELLENCE IN INTERNET OF THINGS (ACEIoT)

Designing an FPGA-based System-on-Chip for X-ray Image Analysis and Tuberculosis Detection

A dissertation submitted in partial fulfillment of the requirements for the award of Master's of Science degree in Internet of Things: Embedded Computing Systems

Submitted By

ARTHUR NATHANIEL MWANG'ONDA (REF.NO: 221030693)

Supervised by:

DR. JOEL MANDEBI

DR. OMAR GATERA

November 2023

Declaration

I **ARTHUR NATHANIEL MWANG'ONDA**, Masters' student from African Center of Excellence in Internet of Things, at University of Rwanda. I declare that this research thesis is my own original work, and it has never been presented before anywhere in the world.

Names: **ARTHUR NATHANIEL MWANG'ONDA**

Ref: 221030693

Signed: ……

Date: ..20./..11../2023

Bonafide certificate

This is to certify that this submitted Research Thesis work report is a record of the original work done by **ARTHUR NATHANIEL MWANG'ONDA (Ref. No: 221030693)**, MSc. IoT-ECS Student at the University of Rwanda / College of Science and Technology / African Center of Excellence in Internet of Things, the Academic year 2021/2023.

This work has been submitted under the supervision of **Dr. JOEL MANDEBI** and **Dr. OMAR GATERA**.

Main Supervisor: **Dr. JOEL MANDEBI** Co-Supervisor: **Dr. OMAR GATERA**

Date: 11/19/2023 Date:

Signature  Signature:

The Head of Masters and Training
Dr. James Rwigema
Signature:
Date:
Signature……

Acknowledgement

I extend my deepest gratitude to Dr. Joel Mandebi, my main supervisor, whose unwavering dedication, guidance, and expertise have been invaluable throughout the entire duration of my thesis. Dr. Mandebi's commitment to teaching, coordination, and supervision has been instrumental in shaping the quality and direction of this work. His insights and mentorship have been a constant source of inspiration.

I am also profoundly thankful to Dr. Omar Gatera, my co-supervisor, for his valuable input, support, and encouragement during this research. His perspective and suggestions have significantly enriched the depth and scope of this thesis.

I would like to acknowledge the management of the African Centre of Excellence in Internet of Things at the University of Rwanda for their support, resources, and conducive research environment. Their commitment to fostering academic excellence has provided an invaluable platform for my academic pursuits.

To my family, whose unwavering support, understanding, and encouragement have been the cornerstone of my academic journey, I am deeply grateful. Their love and belief in me have been a constant motivation to strive for excellence.

I also extend my appreciation to my colleagues Teleza Kanthonga and Sylvia Makupe, for their encouragement which has been a constant source of strength throughout this endeavor.

Lastly, to all those who have in various ways contributed to this thesis, whether through their time, knowledge, or support, I offer my heartfelt thanks.

Abstract

This thesis investigates the utilization of FPGA technology in tuberculosis (TB) detection through X-ray image analysis, focusing on designing an efficient system within the Kria KV260 FPGA-based System-on-Chip architecture. The methodology integrates hardware development emphasizing the Kria KV260 FPGA-based SoC and deep learning model training, transitioning from conventional computing to specific optimization within Vitis AI for deployment on the Kria KV260 platform. Comparative analysis assesses the FPGA-based SoC against CPU-based platforms, evaluating power efficiency, throughput, and latency. Results highlight the FPGA's superiority, notably the Kria KV260, demonstrating significantly lower power consumption during inference, emphasizing FPGAs inherent energy efficiency. Additionally, the Kria KV260 exhibits superior throughput and lower latency, crucial for efficient TB detection, underscoring FPGAs speed and responsiveness. In conclusion, the study extensively demonstrates FPGA-based architectures' prowess, particularly the Kria KV260, in enhancing energy efficiency and performance for TB detection from X-ray images. These findings emphasize substantial improvements in power consumption, throughput, and latency compared to conventional CPU-based platforms, particularly vital in healthcare applications reliant on deep learning inference. This research lays a foundation for future advancements leveraging FPGA technology for efficient and accurate TB diagnosis in healthcare settings.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Meaning
ACK	Adaptive Computation Kernels
AI	Artificial Intelligence
ALU	arithmetic logic unit
BRAM	Block Random Access Memory
BRAM	Block Random Access Memory
CLB	Configurable Logic Block
CNN	Convolution Neural Network
CPU	Central Processing Unit
DDR	Double Data Rate
DL	Deep Learning
DPU	Deep Learning Processing Unit
DSP	Digital Signal Processing
ECG	Electrocardiogram
FPGA	Field Programmable Gate Arrays
GPIO	General Purpose Input/Output
GPU	Graphical Processing Unit
HBM2	High Bandwidth Memory 2
I/O	Input/Output
I2C	Inter-Integrated Circuit
IP	Intellectual Property
LSTM	Long Short-Term Memory
LUT	Lookup Table
MAC	Multiply-Accumulate
ML	Machine Learning
MPSoC	Multi-processor System-on-Chip
PCB	Printed Circuit Board
PID	Proportional Integral Derivative

PL	Programmable Logic
PS	Processing System
RAM	Random Access Memory
SoC	System-on-Chip
SoM	System-on-Module
SPI	Serial Peripheral Interface
SRAM	static random-access memory
TB	Tuberculosis
UART	Universal Asynchronous Receiver-Transmitter
WHO	World Health Organization

Designing an FPGA-based System-on-Chip for X-ray Image Analysis and Tuberculosis Detection

CHAPTER 1

INTRODUCTION

In this chapter, we present the motivation behind the thesis research. Then, we introduce the problem statement. Next, we draw the hypothesis. Finally, we summarize the research contribution and conclude with the thesis organization.

1.1. Motivation

The bacterial disease known as tuberculosis (TB) is an illness that mostly affects the lungs. It is contagious and can spread through the air. According to the World Health Organization (WHO), TB is one of the main leading causes of death from a single infectious agent, with an estimated 10 million new cases in 2023, 1.4 million deaths [1] according to the WHO report of 2022. Early diagnosis and treatment of TB are crucial for successful outcomes [2]. There are multiple ways to diagnose TB, including the use of chest X-ray images. It generally consists in first obtaining the X-ray images, followed by an analysis relying on either manual or computer-aided methods. The process of obtaining these chest X-ray images is done by a radiologist using an X-ray machine, then they analyze it after either using manual or computer aided methods.

Emerging technologies such as Artificial Intelligence (AI), Machine Learning (ML), and X-Ray Technology play a significant role in diagnosis and early detection of Tuberculosis. Recent research highlighted the advancement of AI on early detection of tuberculosis [3] [4] [5]. The use of these emerging technologies in the context of automating X-ray image analysis has shown improved efficiency and accuracy in the process of detecting TB. For instance, Lakhani and Sundaram classified X-ray images for pulmonary TB manifestations by making use of deep learning techniques, more specifically AlexNet and GoogLeNet, and they were successful in doing so, achieving accurate and efficient results [6]. In addition, Rahman et al. reported a computer-aided method for detecting TB from chest X-ray images that was based on intelligent pattern recognition and used Convolutional Neural Networks (CNNs) [2]. This method had an accuracy

of 88.76%. In addition, Acharya et al. introduced progressive resizing for training models to perform automatic inference of TB using chest X-ray images [7]. This demonstrated the potential for automation in TB detection. Moreover, the use of advanced tools and technology for automatic analysis and classification of chest X-rays into TB and non-TB has been highlighted by Nafisah & Muhammad as a reliable alternative to subjective assessment performed by healthcare professionals [8]. Furthermore, the study by Heo et al. emphasized the potential of deep learning methods, including demographic information, to detect TB in chest radiographs, indicating the broad applicability of automated TB detection methods [9].

According to recent statistics, close to 60% of the African population live in remote areas with limited access to healthcare services [10]. Booi & Breetzke emphasized that those with the least access to healthcare facilities are at a greater risk of tuberculosis [11]. Because of this, diagnosing and treating diseases like tuberculosis is even more difficult than it already is. To address this healthcare gap and combat the prevalence of tuberculosis, it is essential to bring diagnostic capabilities closer to populations in remote areas. Traditional diagnostic tools and hospital-centric approaches are inadequate in these settings due to the limited access to healthcare services, and unreliable power supply. There is a need for a solution with a small form factor, low power consumption, meaningful classification accuracy and high throughput. Considering these challenges, traditional computing systems may turn out to be inadequate since they will require constant power. The rationale for exploring System-on-Chip (SoC) architectures, particularly those based on Field-Programmable Gate Arrays (FPGAs), could be explored in this aspect. FPGAs offer a unique combination of small form factor, low power consumption, and high-throughput capabilities. Through this exploration, we aspire to contribute to the advancement of healthcare technology, making strides towards more inclusive and accessible healthcare in resource-constrained environments.

1.2. Background

This chapter presents the background concepts of this research in two major sections. Section 1.2.1 provides a brief perspective on the System-on-Chip Architectures, on different types of System-on-Chips and the benefits of FPGA. Next, Section 1.2.2 describes Machine Learning in general, different machine learning algorithms. Finally, Section 1.2.3 concludes the background concepts.

1.2.1. System-on-Chip Architectures

System-on-Chip refers to a highly integrated electronic circuit that can fit all or most of the necessary components of a computer or electronic system onto a single semiconductor chip [12]. These components include microprocessor or microcontroller, memory, input/output interfaces, and various peripheral devices as you can see in figure 1.

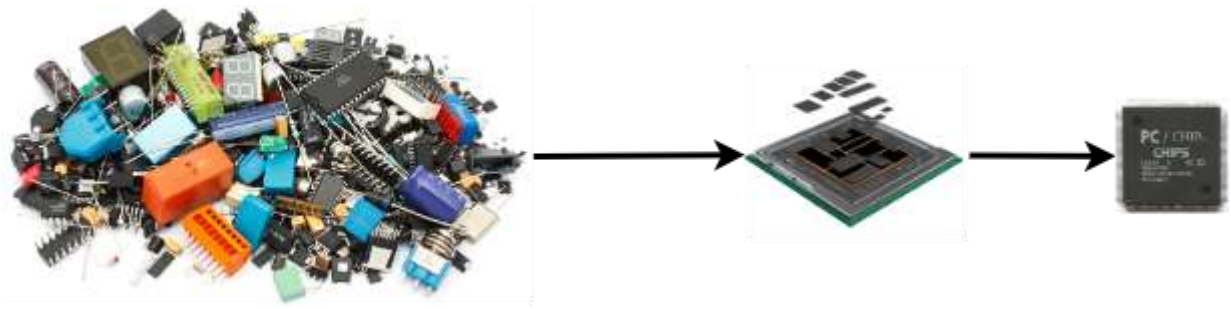

Figure 1: Miniaturization potential of System-on-Chips

SoC consolidates these elements, which results in increased efficiency, decreased footprint, and improved performance in comparison to traditional systems, which use separate chips for each of their respective functions. The integration of several different components onto a single chip has emerged as the standard method for developing embedded systems in the modern era [13]. This integration has resulted in the widespread utilization of reusable hard IP cores, which has helped to address the increasing gap between design productivity and chip complexity [14].

The need for smaller and more powerful electronic devices has been a driving force behind the development of SoC technology. Because of the rapid development of technology, it became necessary to pack more features into ever-decreasing amounts of space, particularly for use in applications such as mobile phones, tablets, and embedded systems. As semiconductor manufacturing processes became more sophisticated, it became possible to include a wider variety of components on a smaller scale. This gave rise to the idea of integrating entire systems onto a single chip, which gained popularity in recent years [15]

1.2.2. SoC Trends

Several notable trends in computer systems have been characterized by the evolution of SoC Technology advancements in semiconductor manufacturing and the reduction in size of transistors have made it possible to integrate more components on a single chip, which has increased functionality and enhanced power efficiency. However, Moore's law and Amdahl's law have made significant contributions to this field.

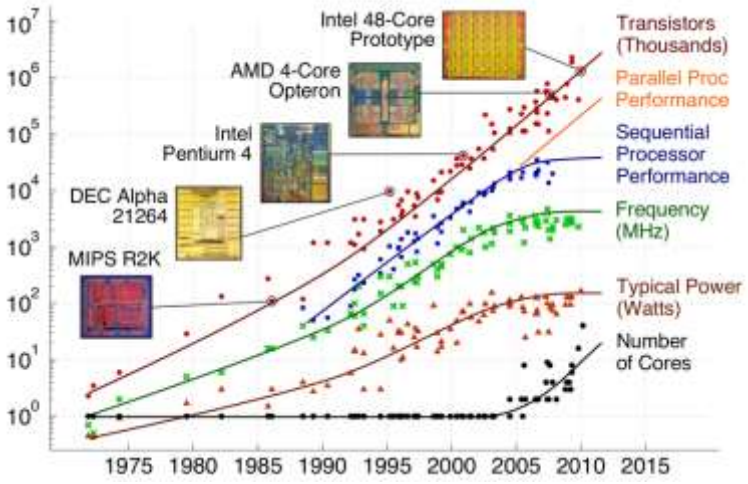

Figure 2: Moore's Law *[16]*

Moore's Law: The first prediction was made by Gordon Moore, the co-founder of Intel in 1965. Which states that "the number of transistors on a semiconductor chip doubles approximately every two years" [17]. This observation has been a driving force behind the rapid pace of innovation that has been taking place in the semiconductor industry. Moore's Law has been very beneficial to SoCs because according to figure 2, we observe that with the increasing number of years there are more and more transistors being integrated together. This has made it possible for more components to be integrated onto a single chip, which has led to improved performance and capabilities.

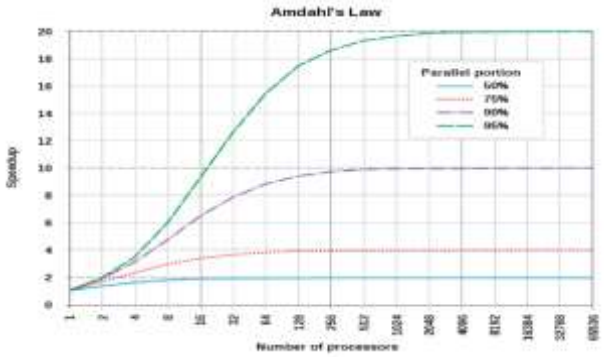

Figure 3: Amdahl's law [88]

Amdahl's Law: Another trend in computer systems was presented by Gene Amdahl in 1967. This law is very important in computer architecture, and parallel computing. It emphasizes that the speedup of a system is limited by the proportion of the program that cannot be parallelized, as it is shown in figure 3. In the context of SoCs, Amdahl's Law encourages designers to identify and optimize the critical and often sequential components of a system to achieve significant overall performance improvements [18]. As SoCs integrate a variety of components, understanding and addressing bottlenecks become crucial for optimizing performance [19].

1.2.3. Benefits of using SoC

The use of SoCs offers several benefits in many domains, such as reduced size, cost savings, low power consumption, high performance, increased functionality, and simplified PCB design. Let's explore each of these benefits in detail:

Reduced Size: The integration of various functionalities, such as processors, memory, and peripherals, into a compact form factor, leads to a significant reduction in the size of the electronic system. This is a key advantage of SoC designs, which can integrate many functions into the same chip, reducing the need for separate components [20], [21]. The immense level of integration in SoCs offers more advantage in applications where space is limited, such as mobile devices, IoT devices, consumer electronics, and biological applications, leading to a reduction in the overall size of electronic systems [22].

Reduced Overall System Cost: There is typically a reduction in costs because of integrating multiple components onto a single chip. It is possible to lower the costs of manufacturing and

assembly by doing away with the need for separate chips, connectors, and other associated components [23]. Additionally, system-on-chips typically result in lower development costs because engineers can concentrate on designing and optimizing a single integrated solution rather than coordinating several separate components. This saves the engineers time and money.

Lower Power Consumption: The power consumption of SoCs can be optimized through careful management of the interactions between integrated components [24]. The reduction in power consumption is especially important for devices that run on batteries, such as smartphones, tablets, and Internet of Things devices. SoCs help extend the life of batteries and improve energy efficiency by reducing the amount of power that they need to operate.

Increased Performance: SoCs have the capability to incorporate advanced processing units, such as multicore CPUs and GPUs, which ultimately results in an improvement in the performance of the system [25]. The integration of specialized components onto a single chip makes it possible to have effective communication between various modules, which in turn helps to reduce latency and improves the responsiveness of the system.

Increased Functionality/Performance in Reduced Footprint: The integration of a wide range of functionalities onto a single chip is made possible by SoC technology, which enhances the capabilities of electronic devices without increasing their size. This is particularly advantageous in scenarios where space is limited, and users require enhanced features and performance without compromising the form factor. SoCs combine high-performance CPUs and the processing power of programmable logic, delivering superior parallel processing, computing power, real-time performance, and versatile connectivity [26].

Simplified PCB Design: The overall design of the Printed Circuit Board (PCB) can be made more straightforward by integrating multiple components onto a single chip [27]. When there are fewer components, there are also fewer interconnections, which can simplify the design and reduce the possibility that there will be problems with the signal's integrity. Simplified PCB designs can help make the manufacturing, testing, and maintenance processes simpler.

1.2.4. Common components of SoC and layers

In this section, we will discuss the common components of a SoC and the layers of integration, and I'll describe each component in detail.

1.2.4.1. Common Components of SoC

The common components that make up System-on-Chip includes processors, Memory, Interconnect, Peripheral components, and finally Integration. Figure 4 shows the integration of the main components that make up SoC.

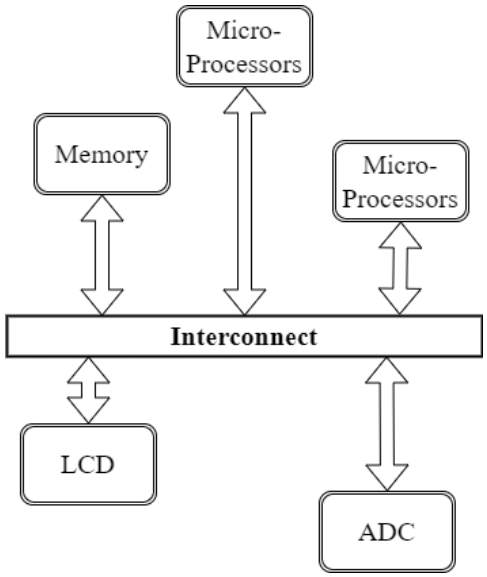

Figure 4: Component of SoC

Processor: The processor is the primary computational component of the SoC, and it may consist of both general-purpose computing units known as central processing units (CPUs) and specialized units known as graphics processing units (GPUs) to handle graphics and parallel processing, respectively. The overall performance of the SoC can be significantly impacted by the selection of the processor architecture as well as the number of cores.

Memory: Memory components of a SoC can be either volatile (RAM) or non-volatile (Flash), depending on the application's requirements. Flash memory is used for the storage of non-volatile data such as firmware, operating systems, and application code, while RAM, is used for storing

data that the processor actively uses while it is operating. Memory management is essential for the effective operation of the SoC, and the incorporation of memory onto the chip allows for a reduction in latency as well as an improvement in data transfer rates.

Interconnect: SoC contains several different components, all of which need to communicate with one another. The interconnect is a network that makes this possible. It consists of buses and communication protocols that make it possible for data to be transferred between the processor, memory, and other peripherals. To maximize the amount of data that can be transferred and reduce the amount of time that it takes for that data to be transferred, the design of the interconnects must be as efficient as possible.

Peripheral Components: Peripheral components include a wide range of functionalities and can include interfaces, controllers, and specialized components such as sensors and communication modules (Wi-Fi, Bluetooth, etc.). Some examples of peripherals include the following: I/O Interfaces: GPIO (General Purpose Input/Output), UART (Universal Asynchronous Receiver-Transmitter), I2C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface). Communication Modules: Ethernet,

Integration: The integration layer is the point at which all of the individual components are assembled and connected to one another. This layer is responsible for the physical design of the SoC, which includes arranging the components on the silicon die in the appropriate fashion.
In addition to its critical function in maximizing the effectiveness of power distribution and maintaining signal integrity, the integration layer is accountable for ensuring that the various components are able to effectively communicate with one another.

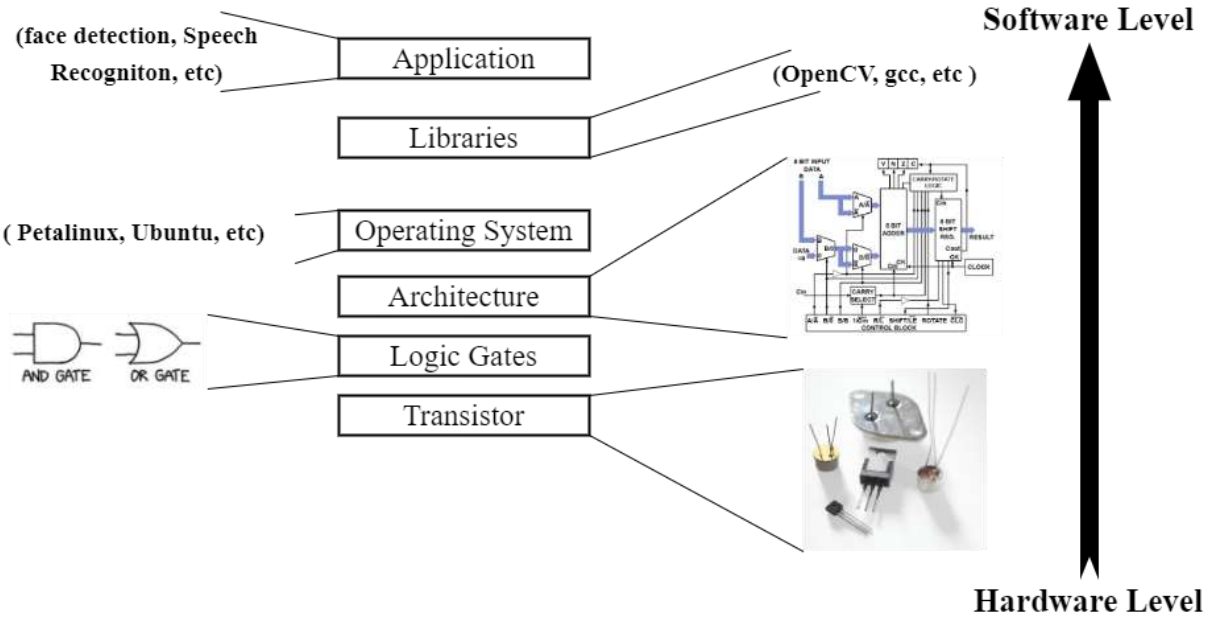

Figure 5: The SoC Layers

The layers of a SoC range from the level of hardware to the level of software. The development process of a SoC typically begins at the lowest layers, which are concerned with the physical components, and works its way up to the highest layers, which are concerned with the software layer as it is indicated in figure 5 above. The following describe in detail the layers in figure 5.

Transistor Level: Engineers begin their work with individual transistors and the connections between them at the lowest level. This involves designing the physical layout of transistors on the silicon die, taking into consideration a variety of factors including power consumption, heat transfer, and signal integrity.

Logic Gates: This layer builds upon the previous one, the transistor layer, and involves the design of logical functions through the combination of logic gates (such as AND, OR, and NOT). This is the stage at which the implementation of more complex functional blocks takes place, such as arithmetic logic units (ALUs) and memory cells.

Architecture Level: The architecture of the entire system is the primary focus of this layer. Engineers are responsible for making decisions regarding the architecture of the processor (for example, ARM or x86), the number of cores, the cache hierarchy, and the layout of memory

subsystems. It is an essential step in the process of defining the hardware structure of the SoC at this stage.

Operating System: After the hardware architecture has been established, the SoC needs to be able to support an operating system such as Ubuntu, Petalinux. Integration at this layer entails the incorporation of necessary components and drivers to ensure compatibility with the operating system that was selected. The operating system is responsible for the management of hardware resources, the facilitation of communication between software and hardware, and the provision of a platform on which applications can be run.

Library: This layer is responsible for the integration of libraries and middleware, both of which offer pre-built functions and routines that make application development easier. Software developers make use of these tools to get more out of the functionality that already exists, thereby reducing the complexity of their code. Examples of this kind of software include graphics libraries, communication stacks, and sensor driver software.

Applications Layer: Developers are responsible for creating user applications, which are in the top layer. The underlying layers are utilized by these applications so that specific functions can be carried out. Depending on the market and the application that is being developed for, the application development process can range from developing low-level firmware for embedded systems to developing high-level software for consumer devices.

1.2.5. SoCs Examples

There are various SoC on the market, and they are developed by different manufacturers based on their specific purpose. They want that SoC to be used.

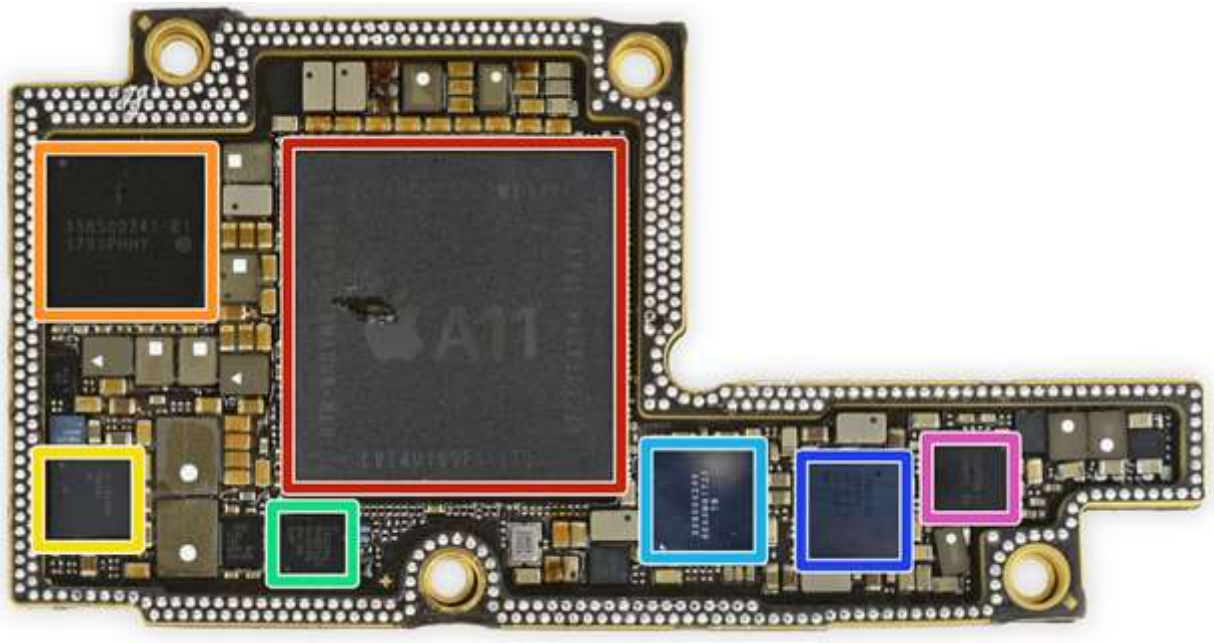

Figure 6: Apple A11 Bionic [22]

Figure 6 shows Apple A11 Bionic which was developed by Apple Inc., and it was created for their products. It was first shown to the public in September 2017, and it has since been integrated into several products manufactured by Apple, including the iPhone 8, the iPhone 8 Plus, and the iPhone X. When it comes to performance, energy efficiency, and the integration of specialized hardware components, the A11 Bionic represents a significant advancement over previous generations. It has the following features: 6 CPU Cores (Frequency: 2390 MHz, Transistor count: 4.3 billion, Instruction set: ARMv8), GPU: Apple GPU (FLOPS: 325 Gigaflops), Memory: LPDDR4 (Frequency: 1866 MHz, Bus: 2x 16 Bits, Max size 3GB)

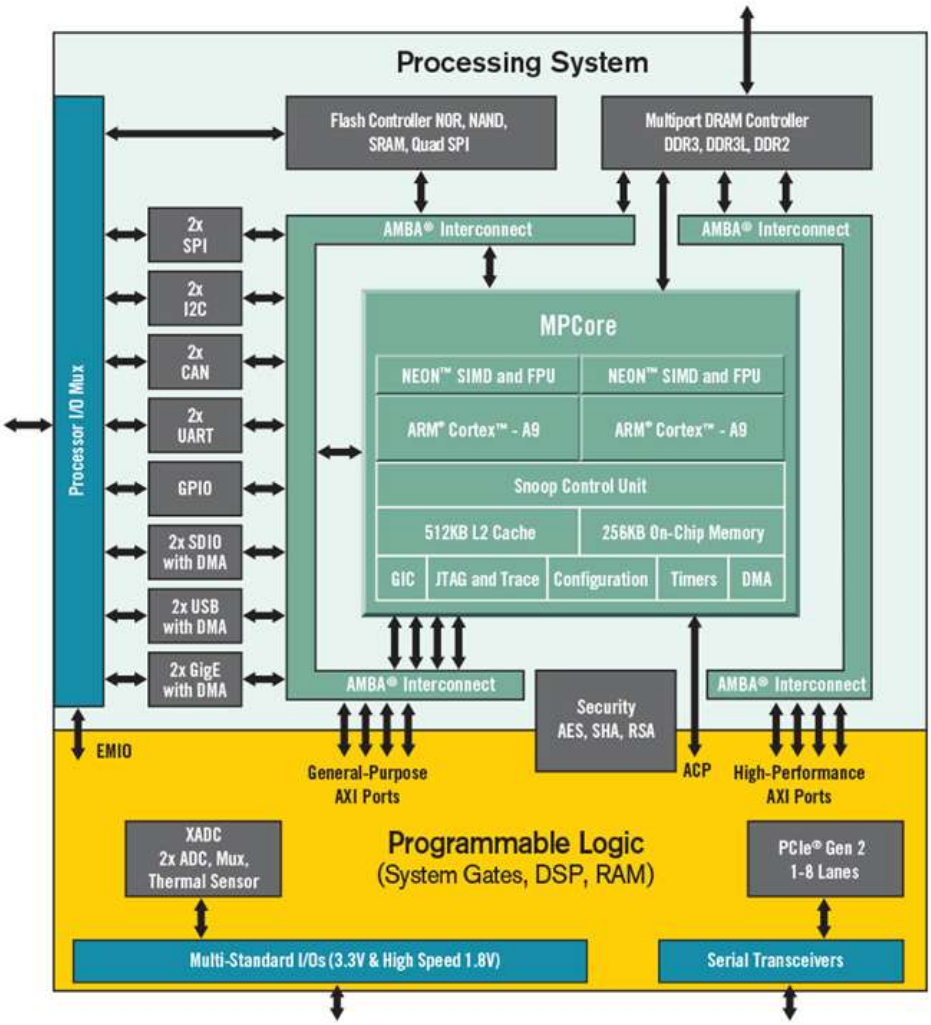

Figure 7: Xilinx Zynq SoC

The Xilinx Zynq is an example of a SoC design, which was first released in the year 2011. The Processing System (PS) and the Programmable Logic (PL) are the two primary components that it is made up of. The CPU that is a part of the Processing System can reach clock frequencies of up to 1 GHz and runs on the ARMv7 instruction set. Additionally, it has 256KB of on-chip RAM, which enables it to process data quickly and effectively. The PS has several different IO peripherals and interfaces, as well as an external memory interface, to make it easier to communicate with other devices. On the other hand, Zynq's Configurable Logic Blocks (CLB), Digital Signal Processing (DSP) blocks, and 36 Kb Block RAMs are all included in its Programmable Logic (PL) component. Because of this, there is a great deal of room for flexibility and adaptability in the

configuration of the hardware. The system-on-chip's capacity to communicate with a wide variety of systems and devices located on the outside of the chip has been significantly bolstered by the incorporation of programmable input/output blocks.

1.2.6. Introduction to FPGA

An FPGA, or Field-Programmable Gate Array, is a type of integrated circuit that can be configured or programmed after it is manufactured [28]. In contrast to traditional processors, which are designed to execute a specific set of instructions, FPGAs are highly flexible and can be customized to perform specific tasks or functions. FPGAs are made up of configurable logic blocks (CLBs), interconnect, programmable input/output blocks, and configurable custom hard IPs such as DSP blocks and BRAMs.

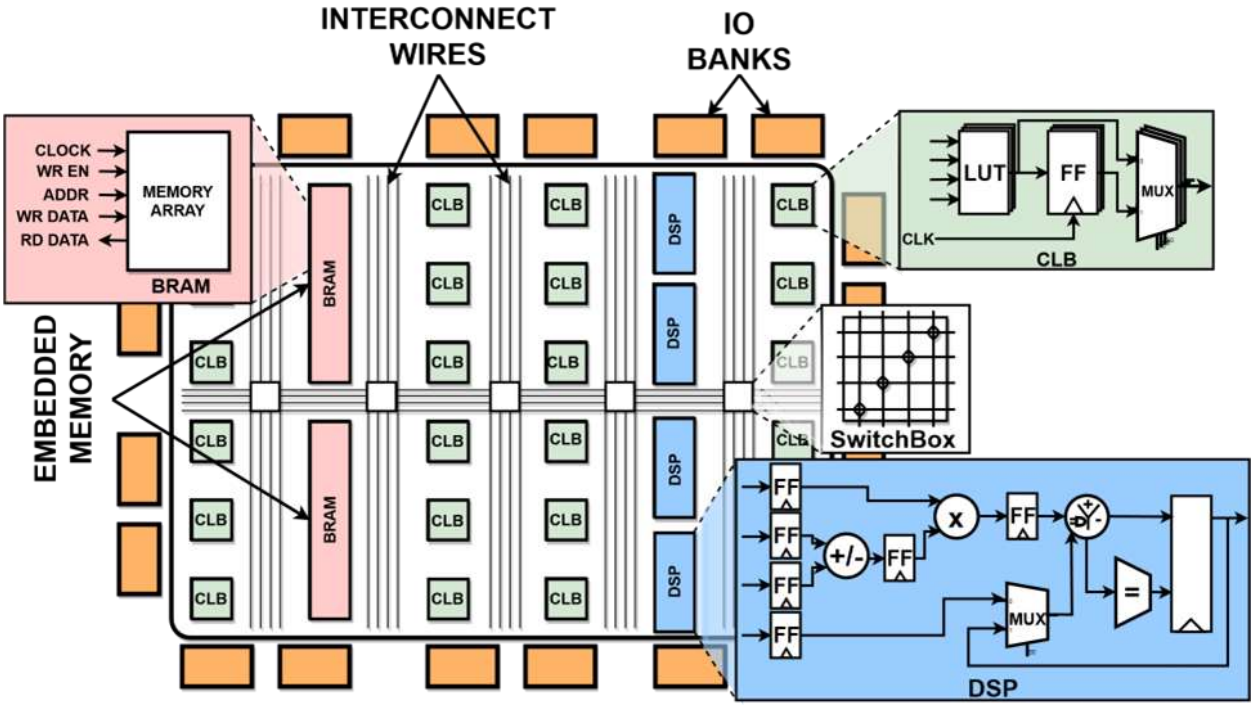

Figure 8: FPGA layout overview [29]

The use of **embedded memory** on FPGA, also known as on-chip Block Random Access Memory (BRAM) memory as it is shown in figure 8, enables distributed high bandwidth memory access within an FPGA chip. When compared to Double Data Rate (DDR) SDRAM memory, BRAM

provides greater flexibility since many BRAM blocks are scattered on an FPGA chip interleaved with other components such as DSP blocks and LUTs. This allows memory to be accessed in a highly parallelized and customized manner. For instance, the peak bandwidth of BRAM on an FPGA could potentially reach terabits per second, whereas the highest level that DDR memory typically reaches is gigabits per second. However, the price of on-chip BRAM is higher, and as a result, there is no capacity that is comparable to that of off-chip DDR memory.

Digital Signal Processing (DSP), blocks found in FPGAs are specialized hardware components designed to accelerate multiply-accumulate (MAC) operations. These operations are fundamental to most signal processing algorithms. Multiplying two numbers, then accumulating the product, and finally adding the sum to an accumulator are the three steps that make up the MAC operation. This operation is particularly common in applications such as filtering and convolution, as well as various mathematical transformations that are prevalent in digital signal processing.

Programmable Interconnects in an FPGA act as a dynamic bridge between the device's processing and memory components, and its name comes from its primary function. This essential component enables the customization of data paths, which in turn facilitates effective communication between the programmable logic blocks, memory elements, and other functional units that are contained within the FPGA architecture. The programmable interconnect plays a pivotal role in optimizing data flow, facilitating parallel processing, and ensuring seamless communication between the various components of the FPGA by providing a flexible network of interconnections that can be configured and reconfigured as needed. This helps to enhance the device's adaptability and performance in a wide range of applications.

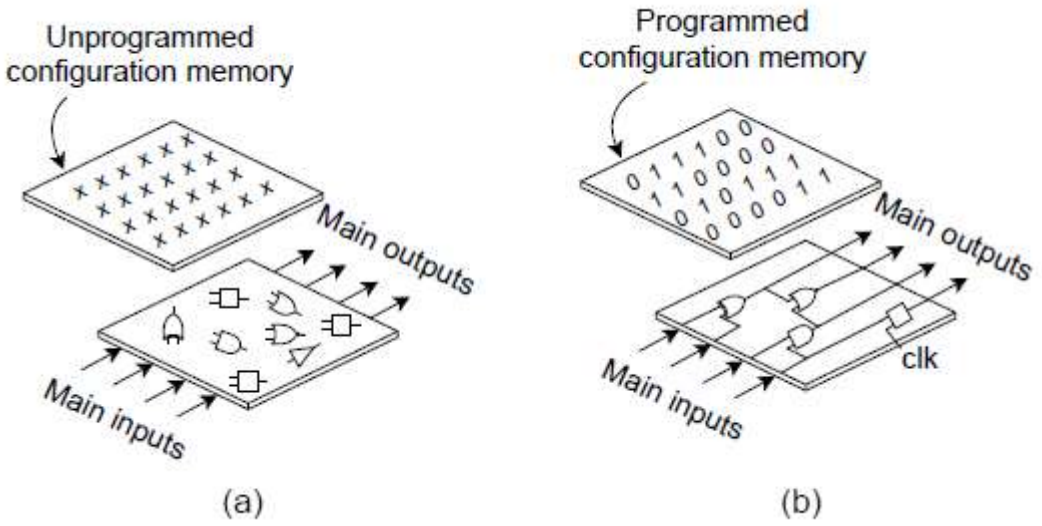

Figure 9: FPGA Programmability. (a) Unconfigured logic Circuit. (b) Logic circuit configured using a bitstream [29]

FPGA (Field Programmable Gate Array) programmability. In figure 9 there are two memories, (a) and (b). (a) shows an unconfigured logic circuit with a configuration memory, main inputs, and main outputs. (b) shows a logic circuit configured using a bitstream with a configuration memory, main inputs, main outputs, and a clock. The configuration memory in (a) is not configured, while in (b) it is configured.

Configurable Logic Block (CLB) is a basic building block that is a key part of making both combinational and sequential circuits work. The CLB is made up of logic gates, flip-flops, and other elements that can be changed. It gives users a flexible way to create and alter digital circuits to meet the needs of specific applications. When used with combinational circuits, the CLB allows complex logic functions by linking its parts together. When flip-flops are used in sequential circuits, they let data be stored and processed over many clock cycles. Because CLBs are programmable, users can set them up to do a lot of different logical operations. This makes them an important part of how flexible and adaptable FPGAs are for many different digital design tasks.

IO banks of a FPGA are dedicated regions that serve the purpose of facilitating communication between the FPGA and other components or systems that are external to the FPGA. The programmable logic contained within the chip is connected to the outside world by means of a

bridge that is provided by these banks. These banks are purposefully designed to manage input and output operations. Each input/output (IO) bank will typically consist of a set of input and output pins, and the arrangement of these pins can be modified to suit the requirements of a particular application.

The **lookup table** (LUT) is the most important piece of hardware in the FPGA platform, as it is what enables its reconfigurability and flexibility. In its most basic form, it is a static random-access memory (SRAM) block that is addressed by the data that is fed into it. By writing memory content into LUTs, configuration of LUTs can be accomplished, which ultimately leads to the implementation of configurable logic functionality in FPGA. On today's FPGA chips, you can find more complicated LUT block structures than ever before.

1.2.7. Benefits of using FPGAs.

FPGAs are specialized integrated circuits that are flexible and programmable, which makes them useful in many situations. In contrast to regular processors, FPGAs can be changed to carry out specific tasks and adjust to changing needs.

Figure 10: The Benefits of FPGA

Based on figure 10 showing the benefits of using FPGA, we are going to explain the above benefits in detail.

Performance: FPGAs are not bound by the Von Neumann architecture, allowing for a high degree of adaptability. This means that the architecture can be tailored to the specific algorithms being implemented, rather than the algorithms having to conform to a fixed architecture. FPGAs can exploit both data parallelism and pipeline parallelism. This enables them to perform multiple tasks simultaneously, leading to enhanced overall system performance.

Low Latency: FPGAs are great at data parallelism, which means they can process multiple pieces of data at the same time. This ability to process multiple tasks at once cuts down on latency when running complicated algorithms. Also, it can use pipeline parallelism to process data continuously without any downtime, which cuts latency even more.

Adaptability: When it comes to applications, FPGAs are very general chips that can be programmed and reconfigured. They can be used for many different tasks because they are flexible, such as computing and networking algorithms. The FPGA is great for situations where needs may change over time because it can be configured differently at any time.

Power Consumption: FPGAs can achieve reduced power consumption compared to traditional Central Processing Units (CPUs) and Graphics Processing Units (GPUs). This effectiveness shows most in power-hungry environments, such embedded systems, and battery-operated gadgets. By dynamically re-configuring themselves, FPGAs may maximize resource efficiency while minimizing power consumption for any given application.

1.2.8. FPGA-Based SoC

The integration of FPGAs into System-on-Chip designs has attracted substantial interest due to the combined benefits of flexibility and integration. FPGAs, which are known for their programmability and ability to construct custom digital circuits, are being integrated into SoCs to harness the benefits of traditional fixed-function SoC [30]. This integration is leading to the creation of heterogeneous SoCs with multi-cores and FPGAs on the same die, linked by an on-chip interconnect [30]. The trend of incorporating FPGA IP cores in SoCs is developing, increasing

the flexibility of the SoC chips [31]. Furthermore, the rising popularity of heterogeneous computing, notably in cloud data centers and flexible SoCs, is driving FPGA incorporation into SoCs [32]. FPGA integration with SoCs is also being investigated in a variety of applications, including industrial control, real-time video tracking, and state-of-charge prediction for lithium-ion batteries [33][34][35][36]. Furthermore, the use of FPGAs in SoCs has been proved to meet real-time requirements for applications such as hyperspectral image compression and background subtraction methods [37][35]. The incorporation of FPGAs into SoCs enables the construction of custom embedded SoCs and hybrid SoCs, which provide flexibility and customizability for specific applications [38][39]. Overall, the integration of FPGAs into SoCs is a burgeoning area of research and development, providing a versatile platform for a wide range of applications, from high-performance computation to real-time control and estimation systems.

1.2.8.1. Examples of FPGA-Based SoCs

There are different FPGA-based SoCs on the Market that are developed by different companies such as AMD, Intel, and Microchip. They are developed with various features and functionality to address the needs in several classes of applications. Some of the examples include: Zybo, Versal, Kria KV-260, PolarFire, and Intel Stratix10. Zybo is an educational and prototyping board powered by a Xilinx Zynq-7000 series SoC, Versal is a versatile family of heterogeneous FPGA-based SoCs for a variety of applications, and Kria KV-260 is an edge computing System-on-Module powered by a Zynq UltraScale+ Multi-processor System-on-Chip (MPSoC), specifically designed for AI inference and vision processing.

Table 1: FPGA-Based SoC examples.

Name	Features
Zybo	• FPGA: Xilinx Zynq-7000 series • ARM Cortex-A9 processor • On-board peripherals and connectors • HDMI, VGA, Ethernet, USB, etc.
Versal	• FPGA: Xilinx Versal ACAP • Heterogeneous multi-core architecture • AI Engine for machine learning acceleration • High-speed interfaces (PCIe, Ethernet, etc.) • Versatility for a wide range of applications
Kria KV-260	• FPGA: Xilinx Kria K26 series • Quad-core ARM Cortex-A53 processor • AI Engine for AI/ML acceleration • High-speed connectivity (Ethernet, PCIe) • Designed for vision and edge AI applications
PolarFire	• Company: Microchip • Fabric Type: FPGA fabric integrated with CPU cores. • Process Technology: Manufactured using 28nm process technology. • CPU Cores: Up to 4x RISC-V CPU cores (64-bit RV64 MAFDC). • AI/ML Acceleration: Embedded AI/ML acceleration hardware for optimized machine learning tasks. • Memory Support: DDR4 memory interfaces for system memory. • High-Speed Interfaces: Supports PCIe, Ethernet, and other high-speed interfaces for connectivity.
Intel Stratix 10	• Company: Intel • FPGA Fabric: High-performance FPGA fabric with up to 2.8 million logic elements (LEs) • Process Technology: Manufactured using Intel's 14nm Tri-Gate process technology. • HBM2 Memory Interface: High Bandwidth Memory 2 (HBM2) interface for high-speed memory access • Embedded Hard Processor: ARM Cortex-A53 processor cores integrated for system control. • Transceiver Support: High-speed transceivers supporting various protocols (e.g., PCIe, Ethernet)

1.2.9. Rationale for Choosing Kria KV260

The choice to use the Kria KV260 as a prototyping hardware platform was driven by a combination of factors that align with the specification of this project. We have considered the Kria KV260, because of its features and capabilities in the context of this project's needs. Here are some reasons why we have chosen the Kria KV260:

Versatile FPGA Platform: The Kria KV260 is built around Xilinx's Kria system-on-module (SOM), which features a Zynq UltraScale+ MPSoC with an FPGA fabric. This makes it a versatile platform for a wide range of applications, as FPGAs can be reconfigured to perform specific tasks efficiently [40].

Processing Power: The ARM Cortex-A53 and Cortex-R5 cores, as well as the FPGA fabric, are integrated together in the Zynq UltraScale+ MPSoC. This gives us the ability to run complex software tasks on the ARM cores while offloading computationally intensive or custom functions to the FPGA [41]. As a result, performance can be optimized for workloads.

AI/ML Acceleration: Acceleration capabilities for artificial intelligence and machine learning are built into the KV260's programmable logic. Because of this, we have designed custom accelerators in the FPGA fabric, which is useful because this project involves machine learning inference, image processing, and other AI-related tasks such as prediction accuracy.

Reconfigurability: FPGAs are reprogrammable, which means we have adapted the hardware to evolving project requirements without changing the physical hardware. This reconfigurability has a significant advantage for this project.

Connectivity and I/O: The KV260 offers a variety of I/O interfaces, including PCIe, Gigabit Ethernet, USB, and more. This is crucial for interfacing with external devices or networking, making it suitable for a wide range of applications.

2.1.8.1. Hardware architecture

The Kria KV260 is an FPGA-based system-on-module (SoM) developed by Xilinx, designed for use in embedded vision applications. It features a powerful Xilinx Zynq UltraScale+ MPSoC,

which combines programmable logic (FPGA) with Arm Cortex-A53 processors and Arm Cortex-R5 processors. Below is an overview of the hardware architecture of the Kria KV260:

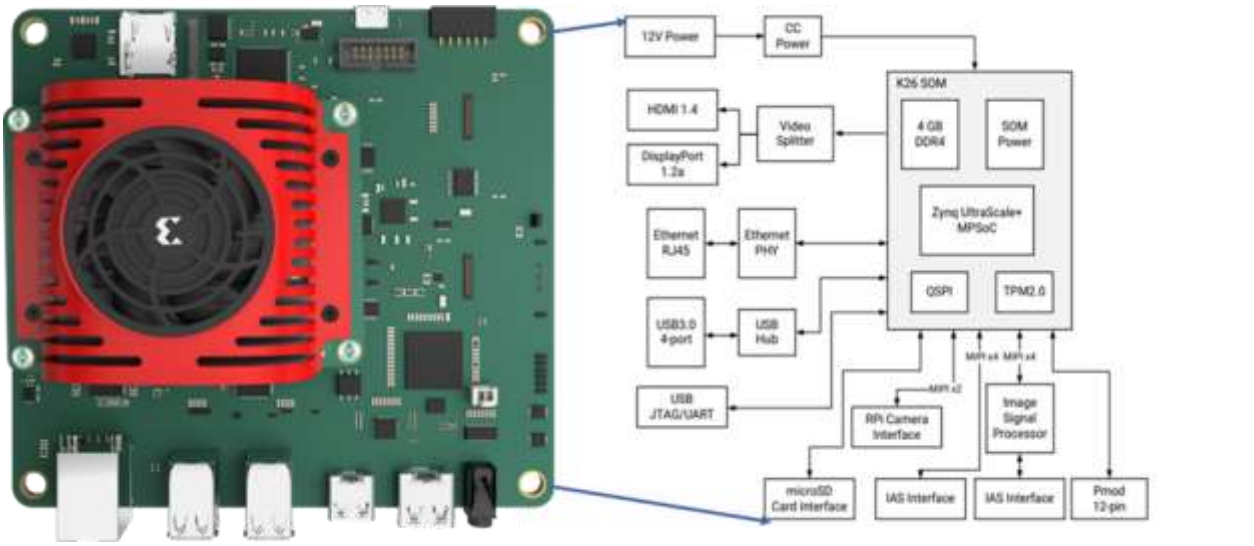

Figure 11: Hardware Architecture of Kria KV260

Zynq UltraScale+ MPSoC: The heart of the Kria KV260 is the Zynq UltraScale+ MPSoC, which integrates several key components into a single chip. This includes:

- Quad-core Arm Cortex-A53 processors: These application processors are used for running general-purpose software and high-level tasks.
- Dual-core Arm Cortex-R5 processors: These processors are designed for real-time processing and control tasks.
- Programmable Logic (FPGA): The FPGA fabric allows for hardware acceleration and customization of specific functions.

Programmable Logic (FPGA)

- The FPGA portion of the Zynq UltraScale+ MPSoC is highly configurable and allows the design of custom hardware accelerators and processing units to offload specific tasks.
- This FPGA fabric is used to implement image processing, machine learning, and other custom hardware functions for embedded vision applications.
- We can also use tools like Vivado and Vitis to design, program, and configure the FPGA portion.

Memory Hierarchy: The Kria KV260 includes various levels of memory to support different tasks and data storage needs, including:

- DDR4 memory: Used for high-capacity and high-speed data storage.
- On-chip memory: Including both cache and internal memory for fast data access.
- SD Card and eMMC options for additional storage.

I/O Interfaces: The Kria KV260 features a range of I/O interfaces for connecting to external devices and networks, including:

- HDMI ports for video output.
- Gigabit Ethernet for network connectivity.
- USB ports for peripheral devices.
- GPIO and UART interfaces for control and communication.
- MIPI CSI-2 and DSI interfaces for camera and display connectivity.

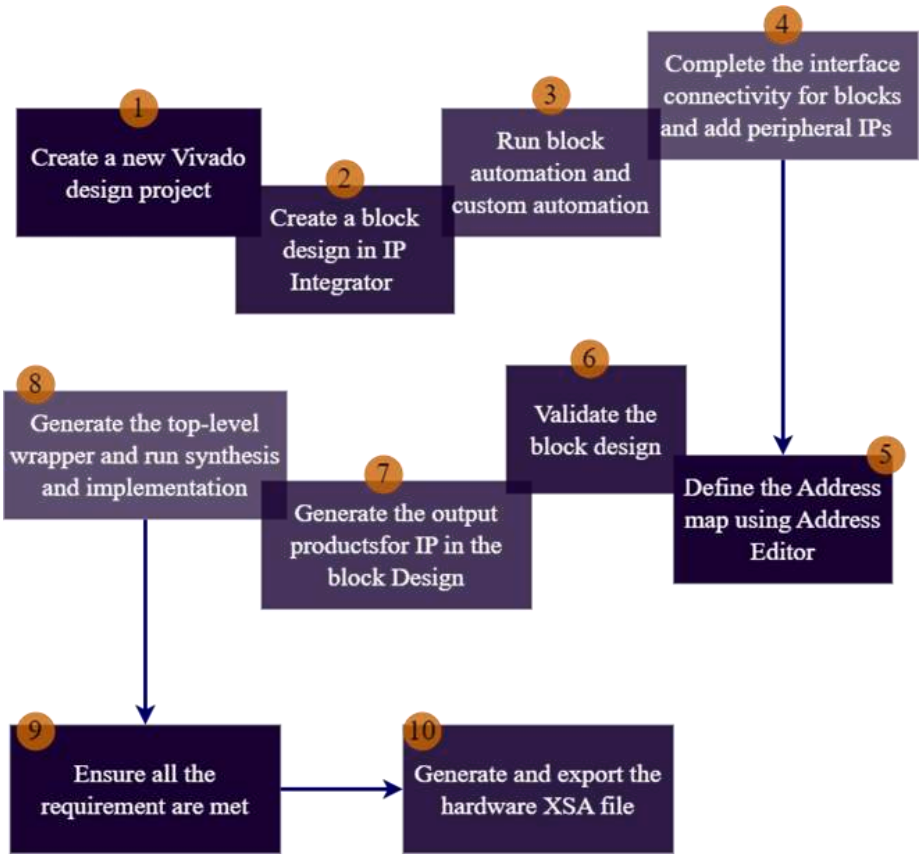

Figure 12: Steps for Building Hardware Architecture in Vivado Design Suite

Figure 12 above shows how hardware architecture is being designed. The hardware design is required for applications running on the Kria KV260. We use the Vivado Design Suite. At the end of the steps, the hardware design is generated in the form of a **.XSA** file which later is needed when we are building the software architecture.

1.2.9.1. Software architecture

An FPGA-based System-on-Chip like the Kria KV260 often has numerous layers of software architecture, especially when using Xilinx toolchain. As it is shown in figure 11, after the hardware has been designed. An XSA file is generated, in this XSA file is where the hardware platforms are encapsulated meaning to say it contains different components of the hardware. XSA includes hardware information, additional IP blocks, metadata to support kernel connectivity and software part containing acceleration kernel requirement for target Linux such as Petalinux, Xilinx Runtime (XRT), kernel interface declaration, and metadata.

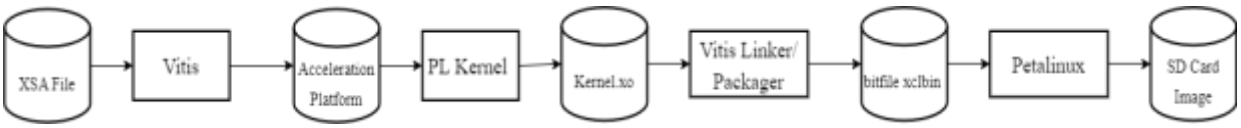

Figure 13: Software Architecture

Software architecture is being built in Vitis. In figure 13 we see an XSA file which was generated from Hardware architecture above. Then it is imported in the Vitis toolchain, so that Vitis Acceleration platform should be created. After the Acceleration platform has been generated this will be fed to the Programmable Logic Kernel where C/C++ code together with RTL are being compiled using Vitis compiler. Then Kernel.xo file will be generated, after that kernel.xo file will be fed into Vitis Linker/Packager to produce a bit file with an extension of .xclbin. Then finally this file will be compiled with petalinux image and other libraries to produce an image which will be later transferred in SD Card to be used in Kria KV260. Since Kria KV260 comes with other firmware utilities such as (xmutil) as tool run commands such as checking the power usage and other applications.

1.2.10. Introduction to Machine Learning

Machine learning is a subfield of artificial intelligence (AI) that focuses on the development of algorithms and models that enable computers to learn patterns and make decisions or predictions without being explicitly programmed in the process [42]. There are three main categories of ML, these types include supervised learning, unsupervised learning, and reinforcement learning, with each category encompassing various specific algorithms [43].

Figure 14: Types of Machine Learning

In **supervised learning**, a model is "trained" on a dataset that has been labeled, meaning that the input data is matched up with the appropriate answer. Finding patterns in the relationship between input and output is how the model learns to make predictions for the future. algorithms such as decision trees, support vector machines, logistic regression, and random forests are frequently used in supervised learning.

Unsupervised learning, on the other hand, works with data that has not been labeled and seeks to discover intrinsic structures or patterns that are hidden within the data that is input. In unsupervised learning, clustering algorithms like K-means and EM (Expectation-Maximization) are frequently used to group together similar data points based on the features they share.

Reinforcement learning is a type of machine learning in which an autonomous agent acquires the ability to make decisions through iterative interactions with an environment, wherein it receives evaluative feedback in the form of rewards or penalties. Reinforcement learning algorithms, such as Q-learning and policy gradient methods, are commonly employed in situations where an autonomous agent must acquire a sequence of actions to optimize the overall sum of rewards.

1.2.10.1. Neural Network

Neural Networks are a type of algorithm that mimics the way the brain works. They are made up of stacked layers of interconnected neurons [44]. Typically, a neural network will have an input layer, a few hidden layers, and an output layer [45], with the feedforward neural network being the most popular. Where **Input layer** takes in the initial data or features, the **few hidden layers** which is an intermediate layer between input and output, where processing and training take place and it is at this layer where data is mathematically transformed at each hidden layer node. Finally, the Output **layer produces** the results or predictions.

During the training process, neural networks use weights and biases to fine-tune the strength of connections between neurons. Forward propagation involves making predictions, and backward propagation involves correcting those errors by adjusting weights [46].

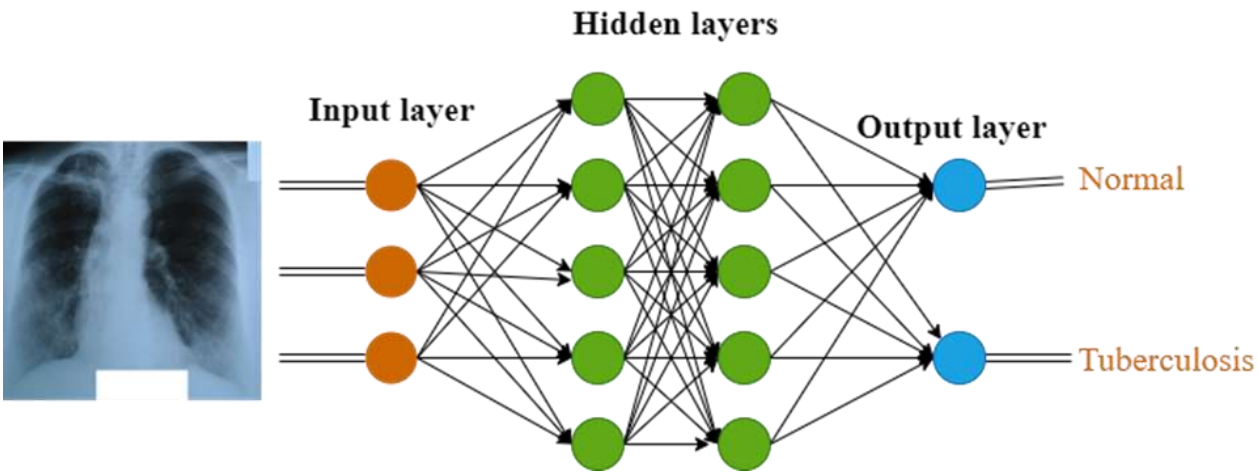

Figure 15: Examples of Neural Network for Tuberculosis Classification

In figure 15, we see the chest x-ray image as an input being taken in at the input layer, this is based on some specific features from the image, and it is sent to hidden layers where the training of the network takes place. From there it is sent to the output layer where prediction is made where the chest x-ray image contains tuberculosis, or it is normal.

1.2.10.2. Deep Learning

Deep Learning is a subfield of machine learning that emphasizes complex networks of neurons that contain many hidden layers. The term "deep" refers to the depth of the network, i.e., the number of hidden layers [47]. The ability to automatically learn hierarchical representations from data has contributed to deep learning's rapid rise in popularity. Deep neural networks have the capability of automatically extracting relevant features from raw data, thereby reducing the need for manual feature engineering [48]. They can grasp complicated and abstract representations of information.

1.2.10.2.1. Common Architectures in Deep Learning

There are different architectures of Deep Learning, some of them include Convolution Neural network most suitable for image recognition and computer vision tasks, and Recurrent Neural Network mostly used sequence data, such as time series or natural language.

Convolution Neural network

Convolutional Neural Networks are a subcategory of deep neural networks that have been developed specifically for the purpose of processing structured grid data, like that found in images. They have achieved a great deal of success in many different computer vision tasks, such as image recognition, object detection, image segmentation, and many others.

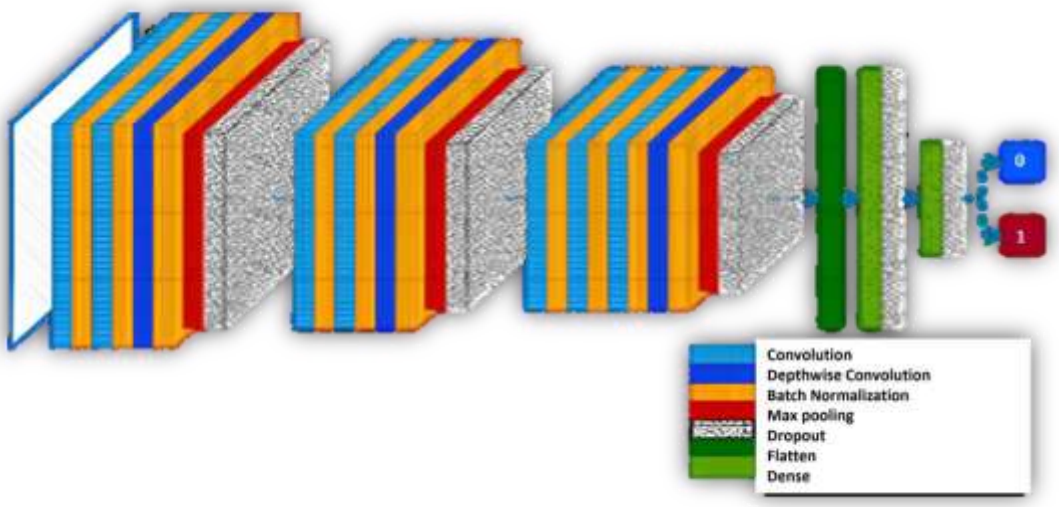

Figure 16: Convolution Neural Network [49]

The colors of the blocks in figure 16. represent different types of layers in the network, including:

Table 2: Description of CNN layers

Layer	**Description**
Convolution layers	These are the main building blocks of CNN. They apply a set of learnable filters to the input image to create feature maps
Depth wise convolution layers	A variant of the standard convolution layer, it applies a single filter per input channel (as opposed to the standard convolution layer, which applies multiple filters to each input channel).
Batch normalization layers	These layers normalize the activations of the previous layer, which can help speed up learning and improve the final performance of the model.
Max pooling layers	These layers sample the input along its spatial dimensions (width and height) by taking the maximum value over a window of inputs.
Dropout layers	These layers randomly set a fraction of input units to 0 during training, which can help prevent overfitting.
Flatten layers	These layers flatten the input into a one-dimensional array.
Dense (or fully connected) layers	These layers connect every neuron in one layer to every neuron in the next layer.

The network appears to have two output nodes, labeled 0 and 1, which suggests that it's designed for binary classification tasks. That is, it's probably being used to classify input data into one of two categories.

The background concepts of the research emphasize System-on-Chip architectures' integration benefits like efficiency and footprint reduction. Flexible FPGAs are highlighted in SoCs. The study's conceptual groundwork highlights the advantages of integration through System-on-Chip architectures, such as increased efficiency and a smaller system footprint. In SoCs, programmable FPGAs take center stage. Along with introducing Machine Learning, it provided a brief overview of Deep Learning, Neural Networks, and the most common Deep Learning architecture.

1.3. Problem statement

Tuberculosis is a chronic lung disease that is one of the top 10 leading causes of death worldwide [50]. Early detection and precise diagnosis of tuberculosis are crucial for the successful implementation of treatment and for controlling the spread of the disease. X-ray imaging is a widely used diagnostic tool for TB detection, but X-ray images interpretation is time-consuming and requires trained radiologists. Therefore, there is a need for automated systems that could speed up the analysis of X-ray images while maintaining the accuracy of the diagnosis compared to currently widespread manual diagnostics.

However, recent research exploring the automation of X-ray image analysis does not emphasize the specific application case of TB using FPGA technology. While some of the previous work discuss FPGA-based systems for image processing and classification, they focus on different applications such as non-destructive detection systems for conveyor belts [51][52], the use of FPGA in ultrasound imaging systems [53], Electrocardiogram (ECG) signal generation and the use of FPGA in real-time ethernet for motion control [54]. Other references discuss FPGA-based systems for image edge detection [55], face detection [56], and use of models developed using deep learning algorithms for analyzing chest X-ray images [57].

The detection and analysis of X-ray images for TB is a critical task in the field of medical imaging analysis. Deep learning (DL) techniques have shown promising results in detecting various medical conditions, including TB, from chest X-rays [58]. However, the vast amount of data

collected for medical diagnosis requires efficient and high-performance computing platforms to enable real-time deployment of TB classification with low latency [49].

One of the key challenges in designing such a system is achieving low-power consumption while maintaining high throughput and low latency. FPGA-based systems offer advantages in terms of low power consumption, parallel computation, and reconfigurability [59]. FPGA accelerators have been used to speed up the inference of various deep learning models, including Long Short-Term Memory (LSTMs) [60]. Additionally, FPGA-based systems have been shown to achieve low latency and high-throughput performance in applications such as high-frequency trading [61].

To address the problem of low latency and high throughput in X-ray image analysis and TB detection, the proposed system will leverage the capabilities of FPGA-based hardware accelerators. These accelerators will be designed to efficiently execute the computationally intensive tasks involved in deep learning inference, such as CNNs and LSTMs The FPGA-based system will also incorporate techniques to optimize memory bandwidth and minimize data communication overhead to further reduce latency [60].

The design of the FPGA-based system will involve mapping the deep learning models onto the FPGA platform and developing novel hardware accelerators tailored to the specific requirements of X-ray image analysis and TB detection. The system will utilize techniques such as adaptive computation kernels (ACK) to execute various computation kernels with low latency and support both dense and sparse computation kernels [62]. Additionally, optimization methods for implementing CNNs and binary CNNs on FPGA will be explored to satisfy the low-latency requirement and enable the use of low-cost compact FPGA devices [59].

The performance of the FPGA-based system will be evaluated in terms of latency, throughput, power consumption, and accuracy. Comparative studies will be conducted to assess the performance of the FPGA-based system against other computing platforms, such as GPUs, in terms of inference speed, energy efficiency, and accuracy [49]. The goal is to demonstrate if the FPGA-based system can provide high throughput, low latency, and low power consumption while maintaining high accuracy in X-ray image analysis and TB detection.

In conclusion, the problem statement involves designing an FPGA-based System-on-Chip for X-ray image analysis and tuberculosis detection. The system aims to achieve low-power

consumption, low latency, and high throughput by leveraging FPGA-based hardware accelerators and optimizing the design for efficient execution of deep learning models. The performance of the system will be evaluated in terms of latency, throughput, power consumption, and accuracy, and comparative studies will be conducted to assess its superiority over other computing platforms.

1.4. Study Objectives

1.4.1. General Objective

The main objective of this research is to design a SoC for X-ray image analysis and TB detection using FPGA-based architecture and machine learning.

1.4.2. Specific Objectives

To achieve the main goal the following specific objectives will be used as guides:

- To Design and verify an FPGA-based SoC architecture for X-ray image classification using AMD-Xilinx devices and toolchain: In the context of this study, we will use **Kria KV260 AI starter kit.**
- To train a machine learning model for TB detection using an existing dataset.
- To integrate the machine learning model into the FPGA-based SoC for hardware acceleration.
- To compare the performance of running a machine learning model on an FPGA-based SoC over a regular computer in terms of power consumption, latency, and throughput.

1.4.3. Kria KV260 AI starter Kit

In the context of this study, we use the Kria KV260 Vision AI Starter Kit which is a comprehensive development platform designed to accelerate the development and evaluation of vision AI applications. The KV260 is based on the Zynq UltraScale+ MPSoC architecture and comes with a fully equipped carrier card for rapid prototyping and design [63]. The following are detailed breakdown of its features and capabilities:

Key Features:

- System on Module (SOM): Xilinx Zynq UltraScale+ MPSoC ZU5EV FPGA with integrated ARM Cortex-A53 and ARM Cortex-R5F processors.
- Vision Accelerator: Deep Vision Processing Unit (DPU) for efficient hardware-accelerated AI inference.
- Multi-camera Support: Up to 8 camera interfaces, including 3 MIPI CSI-2 interfaces and USB cameras.
- Built-in Image Signal Processor (ISP): Pre-processes camera data for optimal performance.
- Connectivity: Gigabit Ethernet, USB 3.0, HDMI, and I2C.
- Software Support: Petalinux tools for building custom Linux images, Vitis AI libraries for AI development.
- Pre-built Applications: Demonstrations for various vision AI applications, such as object detection and image classification.

Technical Specifications:

- Processor: Zynq UltraScale+ MPSoC ZU5EV
- FPGA: Xilinx Artix-7 AC701
- DPU: 2x DPUv3 cores
- RAM: 16GB DDR4
- Flash Storage: 32GB eMMC
- Cameras: Up to 8 (3 x MIPI CSI-2, USB)
- Connectivity: Gigabit Ethernet, USB 3.0, HDMI, I2C
- Power Supply: 12V DC

1.4.4. Regular Computer

In the context of this research, **Regular computer** refers to the computer with the following features: Dell G3 3500, Processor Intel(R) Core (TM) i5-10300H CPU @ 2.50GHz, 2496 MHz, 4 Core(s), 8 Logical Processor(s)

1.5. Hypothesis

The general hypothesis of this study is that the implementation of a SoC architecture using FPGA technology for X-ray image analysis can significantly enhance performance in terms of inference latency, power consumption, and throughput with minimal to no accuracy degradation compared to using regular computers.

It is hypothesized that the FPGA-based SoC design will have a significantly lower power consumption during the inference phase in comparison to when the deep neural network is being run on a regular computer.

When compared with regular computing platforms, it is hypothesized that the FPGA-based SoC architecture will provide high throughput and low latency for the analysis of X-ray images.

1.6. Study Scope

The study focus of this research includes:

1. Design and verify a system-on-chip (SoC) architecture that is based on field-programmable gate arrays (FPGAs) and utilizes AMD-Xilinx devices and toolchain. This involves designing the hardware components and integrating them into a cohesive architecture.
2. Training a machine learning model for TB detection using an existing dataset on a regular computer. This involves selecting an appropriate dataset, preprocessing the data, and training a machine learning model using techniques such as deep learning.
3. Integrating the trained machine learning model into the FPGA-based SoC for hardware acceleration which involves converting the trained model to be compatible with the Kria KV260 and be able to perform the X-ray Image classification for TB.
4. Comparing the performance of running the machine learning model on the FPGA-based SoC with running it on a regular computer. The performance metrics to be considered are power consumption, latency, and throughput.

1.7. Contributions

Contributions of this thesis are listed in this section. This work aims to design a SoC for X-ray image analysis and TB detection using FPGA-based architecture and machine learning. However, we focus our research on developing a hardware architecture and training ML model which will be deployed on the FPGA-based SoC. The contributions of this work are summarized below:

1. Dataset evaluation for X-ray classification: we evaluate several existing datasets and provide a comparative analysis detailing the models best suited to classify X-ray images.
2. Machine Learning Model Pruning: we describe the necessary steps to optimize machine learning models for the KV 260 with the goal of achieving significant performance.
3. Description of the proposed SoC architecture enabling efficient X-ray image classification
4. Comparative analysis of performance achieved with the proposed SoC solution, as opposed to using regular workstations.

1.8. Organization of the Study

There are six chapters in this research as illustrated in Figure 1. In chapter 1, the project's introduction is presented, along with the motivation and contribution of the research that is based on the prototype. Chapter 2 of this study explores background information on topics that will be discussed in the chapters that follow. Chapter 3 touches on the literature review and the identification of gaps in existing research to fulfill the objectives of this project. In Chapter 4, look at the methodology employed in conducting the research. Chapter 5 investigates the topics of system design, simulation, and prototype development. Chapter 6 provides an in-depth analysis of the outcomes and discoveries derived from the conducted study. Subsequently, chapter 7 explores challenges encountered during the research process. limitations, recommendations, and finally draw the conclusion.

In summary, this chapter presented the introduction of this study, problem statements and justification for this study. The objective of the project is to integrate hardware acceleration with machine learning techniques for the purpose of TB detection. The integration of machine learning

functionalities when combined with FPGA acceleration will demonstrate improved diagnostic procedures. This study demonstrates the many potentials that arise from integrating hardware and machine learning, highlighting the potential for enhanced medical diagnostics that are efficient in terms of time taken to process the image, low latency, and energy consumption.

Chapter 2

Literature Review

This chapter presents the state of the art based on what has been done so far based on this field. This chapter contains two sections, section 2.1, we will begin by concentrating on the use of deep learning for the classification of chest X-ray images for tuberculosis. To accomplish this, a comprehensive investigation of the deep learning models, datasets, and methods that have been used to tackle the crucial problem of tuberculosis diagnosis in the medical field is required. After that, we will proceed to the next section 2.2, which will investigate the use of FPGA technology for the purpose of accelerating the inference process of deep learning models. Afterwards, we will conclude this section.

2.1. Classification chest X-ray Images for Tuberculosis Using Deep Learning

This section discusses how deep learning can be applied to the classification of chest X-ray tuberculosis images to enhance diagnostic precision and speed up turnaround times. The use of Field-Programmable Gate Arrays (FPGAs) to speed up the execution of complex machine learning models is the focus of the second area, which enables real-time or near-real-time inferences for applications such as medical image analysis. One recent study by Rahman et al. [2] proposed a reliable TB detection method using chest X-ray images. The authors suggest using chest X-rays to detect tuberculosis. The study uses deep learning, segmentation, and visualization for accurate results. The authors measured accuracy, precision, sensitivity, F1-score, and specificity to evaluate their method. ChexNet, the best model, had 96.47% accuracy. This means the model classified 96.47% of dataset tuberculosis cases correctly. Additionally, the model had 96.62% precision. Precision is the percentage of predicted positive cases that are true positives. Thus, 96.62% of model-predicted tuberculosis positive cases were true positives. The model's sensitivity, which measures how many true positive cases it correctly identified, was 96.47%. The model found 96.47% of dataset tuberculosis cases. The model's F1-score, which considers precision and sensitivity, was 96.47%. It measures the model's precision and sensitivity, with higher values

indicating better performance. Finally, the model achieved 96.51% specificity in identifying true negative cases. The model classified 96.51% of non-tuberculosis cases correctly.

Another study by Huy & Lin [64] built upon the work of and Rajaraman et al. to develop an improved deep neural network model for TB detection using chest X-ray images. The authors presented an enhanced Dense net deep neural network model for tuberculosis detection using chest X-ray images. The study aims to improve the accuracy and performance of existing models in tuberculosis detection. The authors evaluated the performance of their proposed model by measuring various metrics, including accuracy, sensitivity, precision, specificity, and F1 score. The results demonstrate that the proposed model outperforms other models in terms of these metrics. The accuracy of the proposed model was reported to be 98.80%. This indicates that the model correctly classified 98.80% of the tuberculosis cases in the dataset, showcasing its high accuracy in detecting tuberculosis from chest X-ray images. The sensitivity of the model, which measures the proportion of true positive cases correctly identified by the model, was found to be 94.28%. This means that the model successfully detected 94.28% of the tuberculosis cases present in the dataset. The precision of the model, which refers to the proportion of true positive cases out of all the cases predicted as positive, was measured to be 98.50%. This indicates that out of all the cases predicted as tuberculosis positive by the model, 98.50% were true positive cases. The specificity of the model, which measures the proportion of true negative cases correctly identified by the model, was reported to be 95.7%. This means that the model accurately classified 95.7% of the non-tuberculosis cases in the dataset. Then, F1 score, which is a measure of the model's accuracy that considers both precision and sensitivity, was determined to be 96.35% [2]. This score indicates the balance between the precision and sensitivity of the model, with higher values indicating better overall performance.

Manohar et al.[65] evaluated different convolutional neural network (CNN) architectures and training strategies for identifying drug-resistant TB in chest radiographs. Their study focussed on the identification of drug-resistant tuberculosis in chest radiographs. The study evaluates different CNN architectures and training strategies to improve the accuracy of detection. The authors highlighted the challenge of drug resistance in tuberculosis, which makes it more difficult to control the disease. To address this issue, they propose the use of CNN architectures and machine learning techniques to accurately identify drug-resistant tuberculosis from chest radiographs. The

paper discussed the evaluation of various CNN architectures and training strategies to determine the most effective approach for detecting drug-resistant tuberculosis. The authors compared the performance of different models based on metrics such as accuracy, precision, sensitivity, specificity, and F1 score. However, the study by Manohar et al. contributed to the field by evaluating different CNN architectures and training strategies for the identification of drug-resistant tuberculosis in chest radiographs. The results of the study provided insights into the effectiveness of different approaches and help in the development of more accurate and reliable detection methods.

Livieris et al. [66] proposed an ensemble semi-supervised learning algorithm for the classification of chest X-rays of tuberculosis. They classified chest X-ray images using labelled and unlabelled data. Their algorithm is proven effective by experiments and statistical tests. The algorithm focuses on developing reliable and robust prediction models with few labelled and many unlabelled data. This is done with semi-supervised learning (SSL). The SSL machine learning method uses labelled and unlabelled data to improve classification models. Their proposed ensemble approach to predict using multiple base classifiers. Base classifiers are trained on labelled and unlabelled data subsets. Voting is used to combine base classifier predictions to get the final classification result. The authors tested their algorithm using chest X-ray images. Expert-annotated labelled and unlabelled images were in the dataset. The algorithm was trained on a subset of labelled data and tested on the rest and unlabelled data. The experiments showed that the proposed algorithm classified chest X-ray images accurately. The algorithm's prediction models were reliable and robust when tested using statistical nonparametric tests.

Xu & Yuan developed a pulmonary tuberculosis classification algorithm based on CNNs [67]. The authors presented a CNN with coordinated attention for the automatic detection of pulmonary tuberculosis images on chest X-rays. The authors conducted experiments using 5-fold cross-validation to evaluate the performance of their proposed model. According to the results reported in their paper, the average accuracy achieved by the CNN with coordinate attention model was 92.73%. This indicates that the model was able to correctly classify pulmonary tuberculosis images on chest X-rays with a high level of accuracy. In addition to accuracy, other performance metrics were also reported. The area under the curve (AUC) was found to be 97.71%, which suggests that the model had a high discriminatory power in distinguishing between tuberculosis and non-

tuberculosis cases. The precision, recall, and F1 score were all reported to be 92.73%, 92.83%, and 92.82% respectively. These results demonstrate the effectiveness of the proposed CNN model with coordinate attention for the automatic detection of pulmonary tuberculosis images on chest X-rays. The high accuracy and other performance metrics indicate that the model has the potential to assist in the early and accurate diagnosis of tuberculosis, which can contribute to improved patient outcomes and more efficient healthcare delivery.

The study which was conducted by Guia et al. where they presented a deep learning approach for the detection of tuberculosis using chest X-ray image classification [68]. The authors conducted experiments and evaluated their model on a test set to assess its performance. According to the results reported in the paper, the model achieved a score of 0.9992 for all metrics, including accuracy, precision, recall, and F1-score. Although the paper does not explicitly mention the accuracy in terms of percentages, a score of 0.9992 indicates an extremely high level of accuracy. This suggests that the deep learning model was able to accurately classify tuberculosis cases from chest X-ray images with a very high degree of precision. The high-performance metrics achieved by the model demonstrate its effectiveness in tuberculosis detection using chest X-ray image classification. The high accuracy, precision, recall, and F1-score indicate that the model has the potential to assist in the early and accurate diagnosis of tuberculosis, which can contribute to improved patient outcomes and more efficient healthcare delivery.

An et al. presents a study on the use of a light deep neural network called E-TBNet for the automatic detection of tuberculosis using X-ray DR imaging. The authors employed stacking ensemble learning to improve the performance of the network. The accuracy of the E-TBNet model was evaluated as one of the performance metrics [69]. According to the paper, the accuracy achieved by the model was 0.941, which corresponds to 94.1% accuracy. This indicates that the E-TBNet model was able to accurately detect tuberculosis in X-ray DR images with a high level of accuracy. In addition to accuracy, the paper also mentions the Area Under the Curve (AUC) as another performance metric. The AUC value achieved by the E-TBNet model was reported as 0.995. A high AUC value indicates that the model has a high discriminatory power and can effectively distinguish between tuberculosis and non-tuberculosis cases. Overall, the E-TBNet model demonstrated strong performance in terms of accuracy, achieving an accuracy rate of 94.1%.

This suggests that the model has the potential to be a valuable tool for the automatic detection of tuberculosis using X-ray DR imaging.

Another study by Akhari et al [70] built a convolutional neural network model for tuberculosis detection using chest X-ray images. presents a study on the development of a CNN model for the detection of tuberculosis using chest X-ray (CXR) images. The authors aimed to leverage deep learning techniques to accurately predict tuberculosis based on the analysis of CXR images. According to the paper, the CNN model achieved an accuracy rate of 97% in detecting tuberculosis using CXR images. This means that the model was able to correctly classify tuberculosis cases with a high level of accuracy, reaching a 97% accuracy rate. The high accuracy achieved by the CNN model indicates its effectiveness in tuberculosis detection using CXR images. This suggests that the model has the potential to be a valuable tool in assisting healthcare professionals in the diagnosis of tuberculosis, providing accurate and reliable results. The study demonstrated the successful development of a CNN model for tuberculosis detection using CXR images, achieving an accuracy rate of 97%. This highlights the potential of deep learning techniques in improving the accuracy and efficiency of tuberculosis diagnosis.

Xie et al. [71] focused on the development of a computer-aided system for the detection of multicategory pulmonary tuberculosis in radiographs. The authors aimed to create a system that could assist in the early screening and diagnosis of tuberculosis, which is crucial for effective control and treatment of the disease. According to the paper, the computer-aided system achieved a sensitivity of 92.3% and a specificity of 92.5% in the detection of multicategory pulmonary tuberculosis in radiographs. These values indicate that the system was able to accurately identify both positive and negative cases of tuberculosis with a high level of accuracy. Furthermore, the AUC-ROC value achieved by the system was reported as 0.964. This suggests that the system had a strong discriminatory power and was effective in distinguishing between different categories of pulmonary tuberculosis in radiographs. Their study demonstrated promising performance in the detection of multicategory pulmonary tuberculosis in radiographs. The system achieved high sensitivity (92.3%) and specificity (92.5%), indicating its ability to accurately identify positive and negative cases of tuberculosis. Additionally, the AUC-ROC value of 0.964 suggests that the system had a strong discriminatory power.

Liebenlito et al. [72] presented a study on the classification of tuberculosis and pneumonia in human lungs using chest X-ray images and CNNs. The authors aimed to develop a model that could accurately identify cases of tuberculosis and pneumonia based on these images. The study utilized a CNN architecture to train and test the model. The dataset used in the study consisted of chest X-ray images of patients with tuberculosis and pneumonia. The authors employed a technique called under sampling to balance the dataset, ensuring that both classes were represented equally. This approach helps prevent bias towards the majority class and improves the model's ability to accurately classify both tuberculosis and pneumonia cases. The results of the study showed promising accuracy rates for the classification of tuberculosis and pneumonia. The best model achieved an accuracy of 86% for identifying cases of tuberculosis and 96% for identifying cases of pneumonia. These accuracy percentages indicate that the CNN model developed in this study was effective in distinguishing between tuberculosis and pneumonia based on chest X-ray images.

Another study published in the American Journal of Applied Sciences in 2023 [73] focused on an algorithm for identifying tuberculosis cavities using an adaptive region growing approach combined with a support vector machine (SVM) classifier. The study conducted an evaluation of the proposed algorithm and compared its performance with an existing technique. The results showed that the proposed algorithm achieved an accuracy of 85%, which was higher than the accuracy of the existing technique, which was reported to be 78%. This improvement in accuracy is significant as it demonstrates the effectiveness of the adaptive region growing algorithm with SVM classifier in accurately identifying tuberculosis cavities. The higher accuracy of the proposed algorithm suggests that it has the potential to enhance the diagnosis and treatment of tuberculosis by providing more accurate and reliable identification of cavities. The use of an adaptive region growing approach in combination with an SVM classifier allows for the identification of tuberculosis cavities based on specific characteristics and patterns. This approach considers the variability and complexity of cavity structures, enabling the algorithm to adapt and accurately identify cavities in different cases.

Another study by Hooper aimed to develop a computer-aided system for the detection of tuberculosis in chest radiographs [74]. The methodology involves the use of image features and deep learning algorithms to automatically detect active pulmonary tuberculosis. The study utilizes

radiological and CT findings of pulmonary tuberculosis in infants to inform the development of the classification system. Additionally, the deep learning-based automatic detection algorithm for active pulmonary tuberculosis provides insights into the development and validation of the detection algorithm used in the study. Furthermore, the computer-aided system for the detection of multicategory pulmonary tuberculosis in radiographs contributes to the understanding of the technology and methodology used in the paper. The results of the study demonstrated the successful development and validation of a computer-aided system for the detection of tuberculosis in digital chest radiographs. The system utilizes image features and deep learning algorithms to achieve accurate and automatic detection of active pulmonary tuberculosis. The study leverages the insights from radiological and CT findings of pulmonary tuberculosis in infants to enhance the classification system's performance. The deep learning-based automatic detection algorithm for active pulmonary tuberculosis provides a robust foundation for the development of the computer-aided system. Overall, the paper presents a comprehensive approach to tuberculosis classification in digital chest radiographs, leveraging advanced imaging techniques and deep learning algorithms to improve diagnostic accuracy and efficiency.

2.2. FPGA Inferencing

Hardware acceleration has become increasingly popular because of its potential to satisfy the stringent requirements that are imposed on real-time applications, such as medical image analysis.

Zakariah et al conducted a study focusing on tuberculosis classification using deep learning and FPGA inferencing [49]. The authors employed a methodology that involved the deployment of a model across various computing devices to infer deep learning technology with FPGA. The study likely utilized deep learning algorithms to analyze medical images for the classification of tuberculosis. Additionally, the authors incorporated FPGA inferencing, which suggests the use of field-programmable gate arrays for efficient and high-performance inference of the deep learning model. The results of the study demonstrated the effectiveness of the deep learning model in accurately classifying tuberculosis from medical images. The authors presented metrics such as accuracy, precision, recall, and F1 score to quantify the performance of their model. Furthermore, the study compared the performance of the model when deployed on different computing devices, highlighting the efficiency of FPGA inferencing. They also went on further, to check performance

evaluation of the deep learning model in classifying tuberculosis images, particularly on FPGA, including comparative results on inference speed and power consumption. Insights into practical implications such as real-time classification in resource-constrained settings.

Another study by Serrano-Gotarredona et al introduced a chip that is capable of performing image convolutions in real time with programmable kernels of arbitrary shape [75]. The technique called Address Event Representation (AER) is what the convolution processing is built on. This technique is a based-on Spike image and video representation technique that is biologically inspired. It allocates greater communication bandwidth to pixels that contain more information. This chip is the first experimental prototype of a smaller size, and its purpose is to validate the circuits that have been constructed as well as the system-level procedures.

The authors Zhang et al. suggested an architecture for system-on-chip that can be reconfigured, Their main intention was about serving imaging-related medical applications [76]. They were able to construct a tiny system by merging regular processors such as general-purpose with FPGA technology and then integrating the whole system into one single FPGA chip. This allowed them to avoid sacrificing processing power and flexibility while yet maintaining a small footprint. To test the viability of the suggested architecture, a prototype circuit board was developed and then tested using a digital X-ray imaging system.

An image capture and processing system prototype for the ERPSat-1 Pico Satellite was developed by Neji et al, who used FPGA technology in their work. Reconfiguration and advancement of hardware designs are both made possible by the FPGA technology [77]. This technology can be used to prototype a variety of applications, such as Multiprocessor System on Chip and Network on Chip. These applications have the potential to be utilized on board the ERPSat-1 pico satellite to reduce the amount of time spent on data processing and handling.

In another work of research, Jung and Kim built an artificial neural network controller hardware using an FPGA-based general-purpose processor and a digital signal processing (DSP) board [78]. This was done to handle nonlinear system control challenges. Real-time control of the backpropagation learning algorithm of a neural network is within the capabilities of the intelligent control hardware that was created. An FPGA chip contains the implementation of the fundamental Proportional Integral Derivative (PID) control algorithms, and a DSP board has the whole process of implanting a neural network controller.

Chapter 3

Research Methodology

In this research, our methodology comprises several stages to achieve the objective of this research study. Initially, we explore a comprehensive Tuberculosis dataset for model training and evaluation. The dataset selection is a critical step, ensuring its representativeness and relevance to the problem. Subsequently, we employ various deep learning frameworks such as TensorFlow and PyTorch, experiment with platforms like Google Colab and Jupyter notebook, and leverage libraries including Keras and Scikit-learn to train different models. The models are rigorously evaluated to assess their accuracy, and the best-performing model is selected for further optimization.

In the subsequent phase, the chosen model undergoes conversion for deployment on the Kria KV260 platform. This involves quantizing the model, evaluating the quantized version, and compiling it for compatibility with Kria KV260. Simultaneously, another workflow focuses on hardware design, encompassing the creation of a platform, PL kernel, Vitis Linker and Packager, and Petalinux BSP. Importantly, we integrate the compiled model into the Kria KV260 environment, ensuring compatibility with the Petalinux BSP loaded onto an SD card. Finally, we execute TB classification on the Kria KV260, affirming the seamless interaction between the designed hardware and the machine learning model for efficient and accurate Tuberculosis diagnosis.

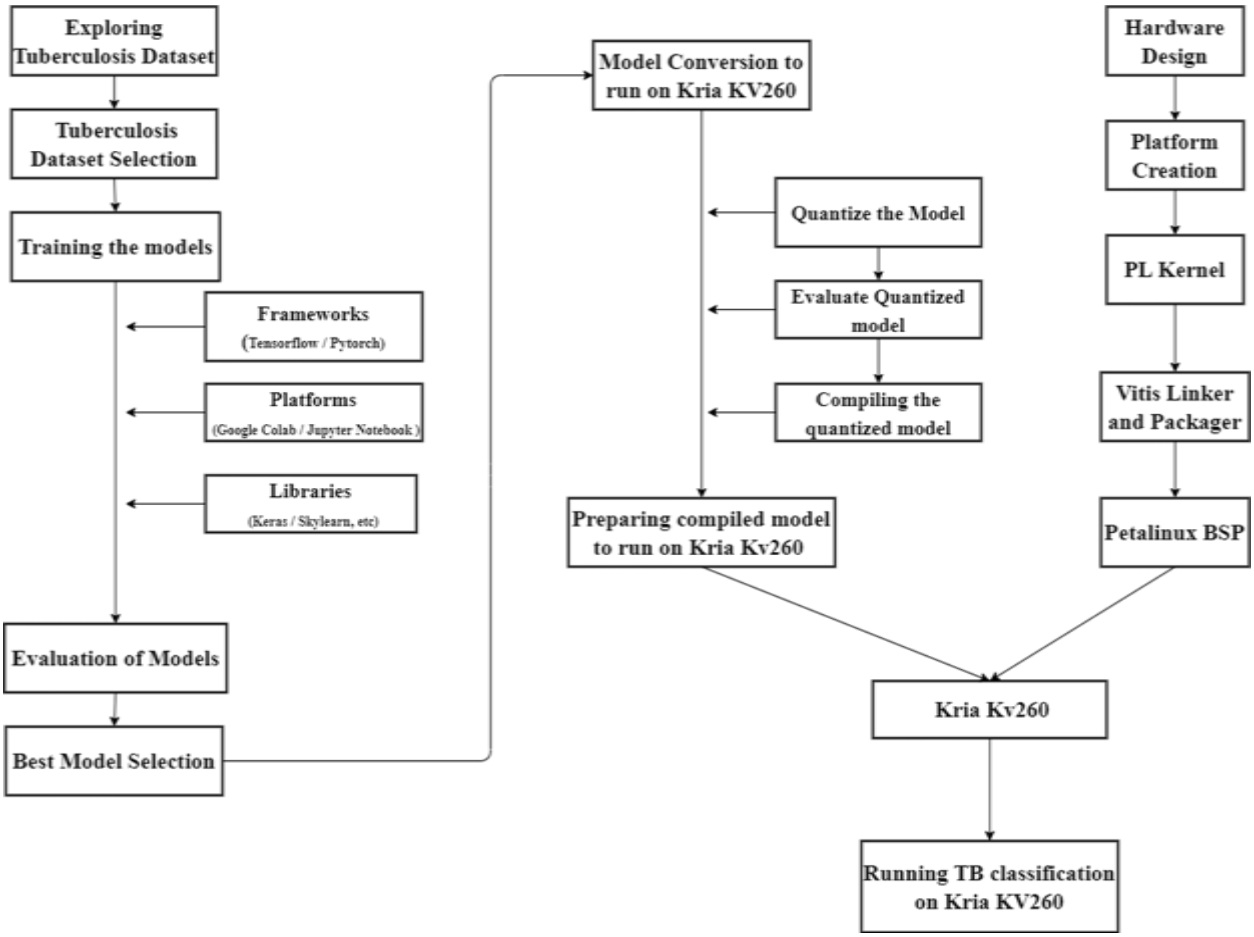

Figure 17: Proposed Methodology

3.1. Rationale for Choosing Methodology

We considered the methodology illustrated in Figure 17 based on the following reasons:

Comprehensive Dataset Exploration: The research ensures that the models are trained on diverse and representative data by extensively exploring a Tuberculosis dataset. This enhances the models' ability to generalize to different cases by providing them with a stronger foundation.

Model Diversity: Training models with different kinds of frameworks (such as TensorFlow and PyTorch), platforms (such as Google Colab and Jupyter notebook), and libraries (such as Keras and Scikit-learn) enables a comprehensive evaluation of several different approaches. This ensures

that the model that is selected does not have a bias toward a particular tool or framework and that it can be applied to a wider range of situations.

Evaluation of Model Performance: The thorough analysis of model performance, with accuracy testing, contributes to the process of determining which model is the most suitable for achieving the criteria for tuberculosis classification.

End-to-End System Integration: The methodology covers all parts of the pipeline, from the development of models and their optimization to the design of hardware and its deployment. This end-to-end approach guarantees a smooth integration of the machine learning model into the hardware environment that has been specified, which contributes to the system's ability to be used in real-world scenarios.

Flexibility and Adaptability: The flexibility and adaptability of the methodology are ensured by the utilization of popular frameworks such as TensorFlow and PyTorch. This is important because improvements in machine learning and hardware technologies are still being made, and the tools that were chosen to make it easy to integrate these improvements.

In Section we presented the research methodology we have adopted to achieve the research objectives. The chosen methodology was based on a strategic combination of considerations, aiming to ensure a robust and effective Tuberculosis classification system while considering both software and hardware aspects.

Chapter 4

System Design and Analysis

In this section we review the proposed solution that enables fast and efficient classification of tuberculosis Chest X-ray images using a SoC architecture. Section 4.1, we start by providing a general overview of the proposed architecture, then using a bottom-up approach, we detail the various layers going from the hardware layer up to the application layer implementing machine learning models.

4.1. Overview of the proposed solution

The proposed solution in figure 18 has hardware and software stack for training and deploying machine learning models. It consists of hardware components such as a processing system and programmable logic, and software components such as libraries and platforms.

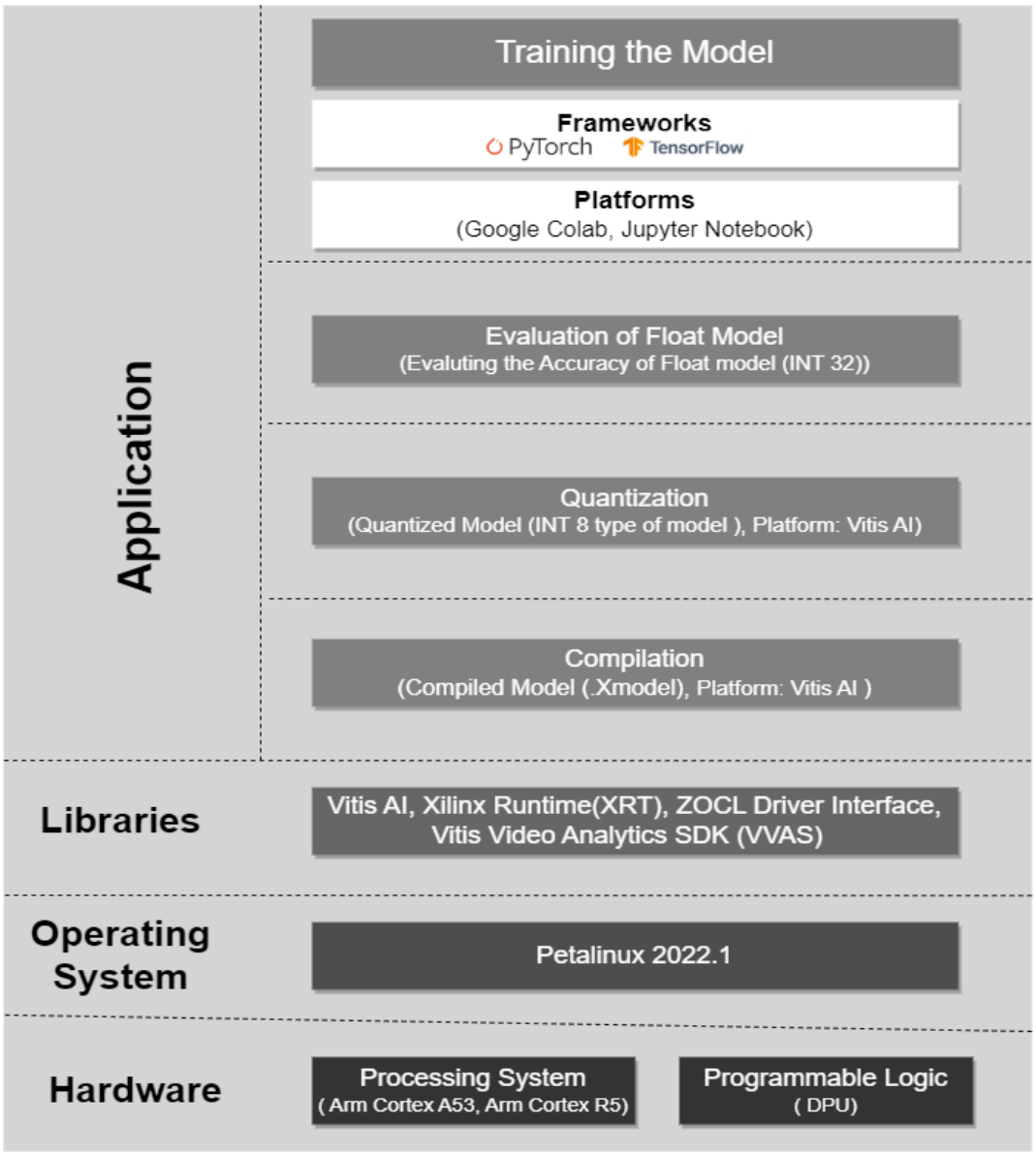

Figure 18: Overview of Proposed Solution

The stack is divided into four layers: hardware, operating system, libraries, and application and it has been arranged in SoC architecture. The hardware layer consists of a processing system which is composed of CPUs such as Arm Cortex A53 for performing other tasks, Arm Cortex R5 for performing real time tasks and programmable logic where Deep Learning Processing Unit resides, and it is responsible for Accelerating all AI tasks. The operating system layer, in this layer Petalinux 2022.1 is used. The libraries layer consists of Vitis AI, Vitis Video Analytics, and SDK Driver Interface called ZOCL driver. The application layer consists of platforms such as Google

Colab and Jupyter Notebook, and a compiled mode. The stack also includes a training workflow with PyTorch and TensorFlow, and an evaluation workflow with Float model and Quantized model.

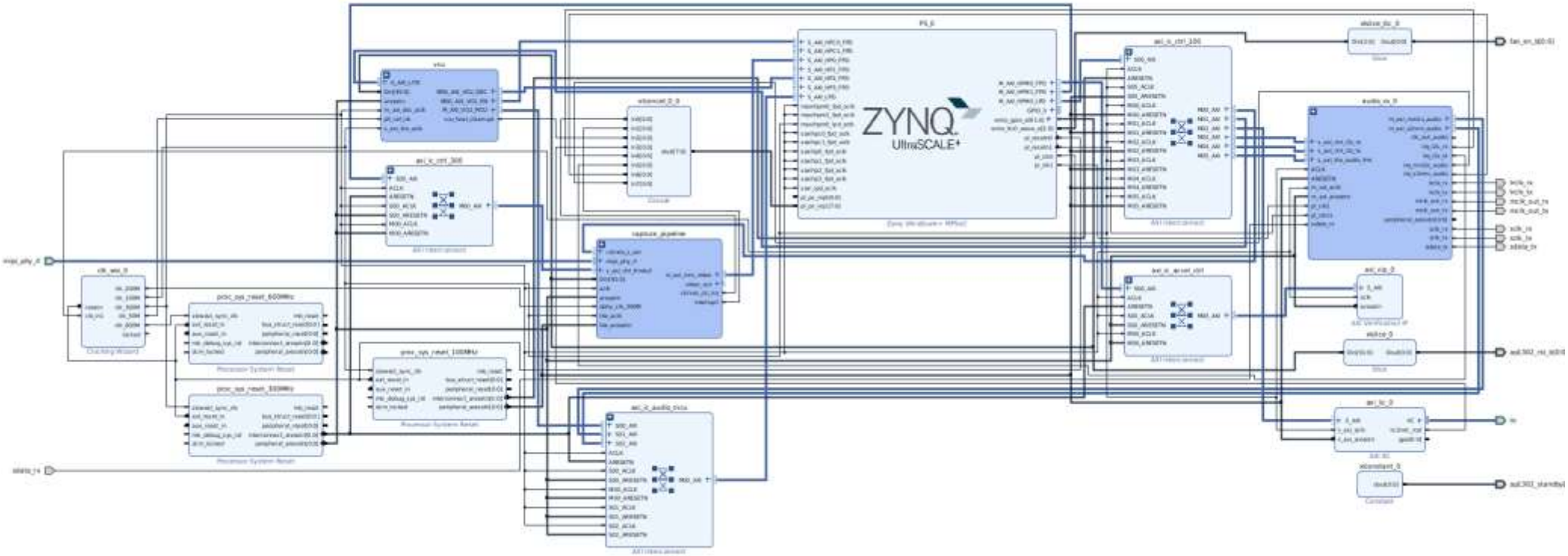

Figure 19: Hardware Architecture of the proposed Architecture

The nature of an IP (master or slave) refers to its role on the AXI bus. A master IP can initiate transactions on the AXI bus, while a slave IP can only respond to transactions initiated by a master IP for example: The Zynq UltraScale+ MPSoC is a master IP because it can initiate transactions on the AXI bus to read and write data to other IPs, such as the DPU and ISP. The DPU is a slave IP because it can only respond to transactions initiated by the Zynq UltraScale+ MPSoC.

Table 3: Description of IPs

IP Name	**Description**	**Goal of the IP**	**IP Configuration**	**Nature of IP**
Zynq UltraScale+ MPSoC	A system-on-a-chip (SoC) that integrates a quad-core ARM Cortex-A53 application processor unit (APU) with a dual-core ARM Cortex-R5 real-time processor unit (RPU), a programmable logic (PL) fabric, and a variety of peripherals.	Provides the processing power and flexibility to develop and deploy a wide range of vision AI applications.	The Zynq UltraScale+ MPSoC is the central processing unit (CPU) of the Kria KV260. It is responsible for running the operating system, executing vision AI applications, and managing the other IPs on the board.	Master
DPU	A deep learning processing unit that accelerates the execution of neural networks.	Provides high-performance and energy-efficient neural network inference for vision AI applications.	The DPU is connected to the APU via an AXI4-Stream interface. It can be programmed using the Vitis AI software development environment.	Slave
ISP	An image signal processor that pre-processes raw image data from camera sensors.	Improves the quality and performance of vision AI applications by removing noise,	The ISP is connected to the DPU via an AXI4-Stream interface. It can be programmed using the	Slave

		correcting distortion, and performing other image processing tasks.	OnSemi AP1302 ISP driver.	
PS DDR Interface	Provides a high-speed interface between the Zynq UltraScale+ MPSoC and the DDR memory	To provide the Zynq UltraScale+ MPSoC with fast access to memory	The PS DDR Interface is a slave IP. It is controlled by the Zynq UltraScale+ MPSoC.	Slave
DDR4 Memory	High-speed memory used to store data and code	To provide the system with large amounts of fast memory	The DDR4 Memory is a slave IP. It is controlled by the PS DDR Interface.	Slave
PL DDR Interface	Provides a high-speed interface between the Zynq UltraScale+ MPSoC and the DDR memory in the PL	To provide the Zynq UltraScale+ MPSoC with fast access to memory in the PL	The PL DDR Interface is a slave IP. It is controlled by the Zynq UltraScale+ MPSoC.	Slave
PL	A programmable logic fabric that can be used to implement custom hardware	To provide the system with the ability to implement custom hardware	The PL is a slave IP. It is controlled by the Zynq UltraScale+ MPSoC.	Slave
AXI Interconnect	Provides a high-speed communication bus between the Zynq UltraScale+ MPSoC and the other IPs in the system	To allow the different IPs in the system to communicate with each other	The AXI Interconnect is a master IP. It controls the other IPs in the system.	Master
MIPI CSI-2	A camera sensor interface that supports high-speed data transfer from up to eight camera sensors.	Provides the ability to connect multiple camera sensors to the	The MIPI CSI-2 receiver is connected to the ISP via an AXI4-Stream interface.	Slave

		Kria KV260 for vision AI applications.		
Ethernet	A high-speed networking interface that supports data transfer rates of up to 1 Gbps.	Provides the ability to connect the Kria KV260 to a network for data transfer and communication.	The Ethernet PHY is connected to the APU via an RMII interface.	Master
HDMI	A high-definition video output interface.	Provides the ability to display video output from the Kria KV260 on an external display.	The HDMI transmitter is connected to the APU via a TMDS interface.	Master
USB	A universal serial bus that supports data transfer and communication with a variety of devices.	Provides the ability to connect external devices to the Kria KV260, such as USB storage devices, keyboards, and mice.	The USB controller is connected to the APU via an AXI4-Stream interface.	Master

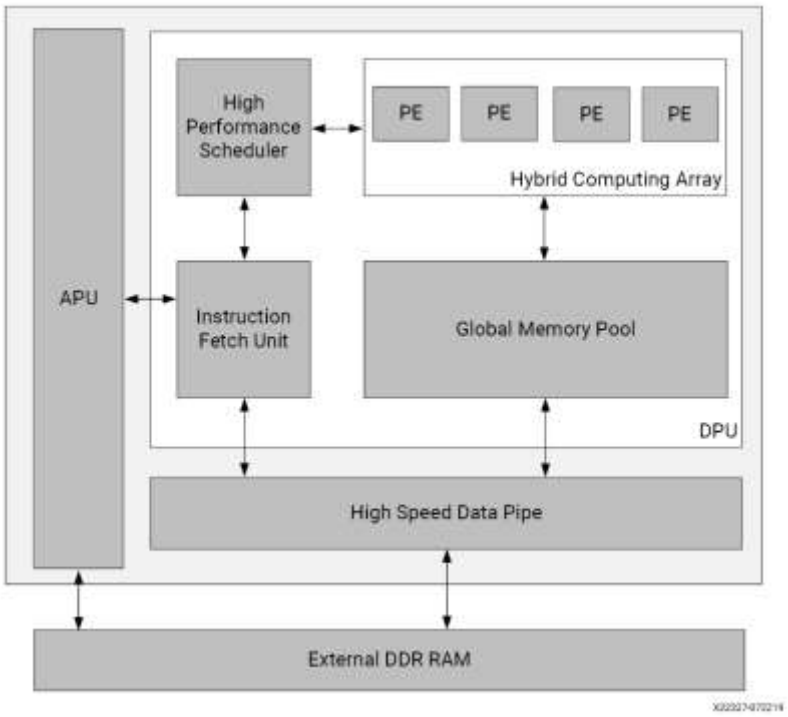

Figure 20: DPU Top-Level Block Diagram[79]

DPU is a programmable engine optimized for deep neural networks. It is a group of parameterizable IP cores pre-implemented on the hardware with no place and route required. The DPU is released with the Vitis AI specialized instruction set, allowing efficient implementation of many deep learning networks. Figure 20 provides a visual representation of the internal architecture of the DPU, which comprises of the following four primary components: the instruction fetches unit block, the scheduler module, the Processing Elements (PE), and the global memory pool module. An ARM processor known as the Application Processing Unit (APU) is the component that oversees running the application, managing data transfers to and from the DPU, and responding to any interruptions that may occur. The DPU instructions that are associated with the CNN models are read by the instruction fetch unit, and these instructions are then executed. After these instructions for the DPU have been processed, they are sent on to the high-performance scheduler to be managed. The high-performance scheduler is responsible for transferring instructions between the PEs and the memory. The DPU instructions are carried out by the PEs using the data that is provided by the Global Memory Pool, which operates as a buffer for both the input and output of data. The block RAM that makes up the on-chip memory is where the weight, bias, and intermediate feature maps are kept. Using the Vitis AI framework, users can make use of the DPU by converting CNN models created with Tensorflow or Pytorch into a format that is

compatible. Vitis AI can make model quantization, pruning, and compilation into DPU-supported instructions easier to accomplish. Within the Vivado or Vitis suite, one has the ability to generate a configuration file named arch.json. This file is then read by the Vitis AI compiler. The instructions and options necessary to configure the DPU core are included in this configuration file for your convenience. For this demonstration, the default configuration file that was supplied is being used in conjunction with the fingerprint of the model architecture, which is **0x1000020f6014406**. I located this fingerprint in the folder of the Vitis-AI software located at **/opt/vitis ai/compiler/arch/DPUCZDX8G/KV260/arch.json**. This configuration file decides how the data processing unit (DPU) is laid out and where the computations are stored. Utilizing the Vitis unified platform allows for the creation of a bespoke configuration for an architectural design. For the purposes of this investigation, the precompiled configuration was chosen.

4.2. Software Architecture

The software architecture of the proposed architecture during the execution of Tuberculosis Classification inference is categorized into three primary layers. The following are the layers, Application Layer, Middleware layer, and Operating Systems layer.

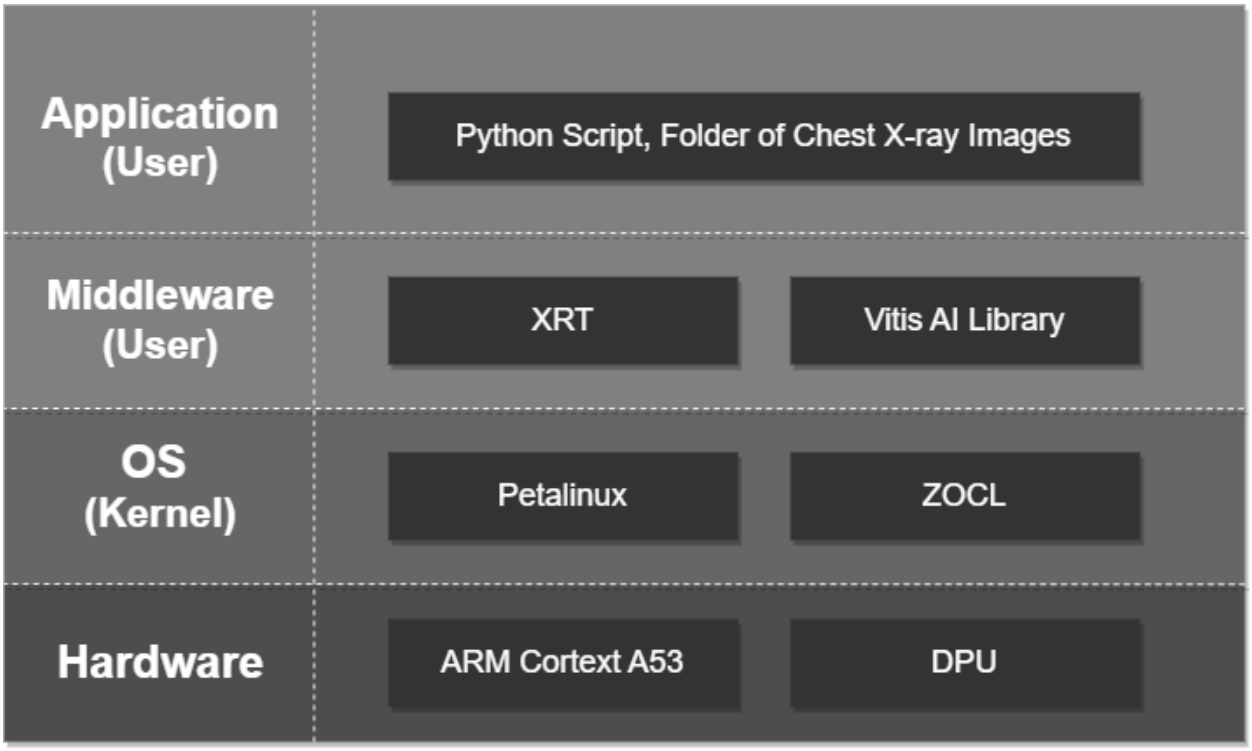

Figure 21: Software Architecture

Application layer: This layer contains the user-facing applications that run on the system. In this case, the application layer contains a Python script that is used to analyze chest x-ray images.

Middleware layer: This layer provides services to the application layer, such as communication, database access, and security. In this case, the middleware layer contains Xilinx Runtime (XRT) and Vitis AI Library, which provide services for accelerating AI processing on FPGAs.

OS layer: This layer provides a basic operating environment for the middleware and application layers. In this case, the OS layer contains Petalinux and ZOCL Driver, which are Linux distributions and C++ libraries for programming Xilinx Zynq UltraScale+ MPSoCs.

Hardware layer: This layer contains the physical hardware components of the system, such as the CPU, memory, and storage. In this case, the hardware layer contains an ARM Cortex-A53 and a DPU, which are the CPU and deep learning accelerator used in the system.

4.2.1. Software - Hardware Communication

When running an inference, the software communicates with the hardware in the following way:

The software communicates with the hardware through the GStreamer pipeline, utilizing a combination of messages and data buffers. The host application sends inference requests and input data to the pipeline, which relays the information to the hardware accelerator via streaming buffers. The hardware accelerator performs the inference and sends the results back through the pipeline, eventually reaching the host application for interpretation and presentation. Having the full picture based-on our architecture the flow goes in the following way:

1. The application layer sends a request to the middleware layer to start an inference.
2. The middleware layer prepares the input data for the inference and sends it to the hardware.
3. The hardware performs the inference and sends the results back to the middleware layer.
4. The middleware layer processes the results and sends them back to the application layer.

4.2.2. Operation Distribution When Running Inference

The operations for running inference on the Kria KV260 are distributed between the software and hardware as follows:

Software:

- Preprocessing and postprocessing of input data (e.g., image resizing, normalization)
- Model loading and initialization.
- Managing communication between software and hardware

Hardware:

- Software Execution is done by Arm Processor (In this context run by Arm Cortex A53)
- Hardware Acceleration done by FPGA fabric.
- Execution of neural network inference on the DPU (Deep Learning Processing Unit)

4.2.3. Software Tools

In this project we have used software tools developed by Xilinx for the design, simulation, and verification of FPGA-based systems as well as compiling the model for the edge device. These tools include Vivado 2022.1, Vitis 2022.1, and Vitis AI 2.5. These tools play crucial roles in the development and testing of FPGA designs and compiling the model for Kria KV260. The below explains more about the software tools used.

Vivado: In the context of this project, I have used Kria KV260, it has played a central role in designing the FPGA-based hardware architecture, including image processing units, memory, and I/O interfaces but more specifically reconfiguring the DPU IP for accelerating the inference. It created a way through for us to optimize, and verify the hardware design, ensuring that it meets the requirements of this project.

Vitis: Vitis plays an important role as far as the development of software applications is concerned that can leverage the FPGA's hardware acceleration capabilities. By combining our custom-built FPGA-based image processing units with the software we developed for TB detection using machine learning and Vitis, we can conduct real-time image analysis with hardware acceleration.

Vitis AI: AI inference on Xilinx hardware platforms, including edge devices, is sped up with the help of the Viti AI development environment. Optimized IP cores, tools, libraries, models, and sample designs are all included. To fully realize the potential of AI acceleration on Xilinx FPGAs, specifically Kria KV260, and on adaptive compute acceleration platforms (ACAPs) like Versal, it is built with high efficiency and ease of use in mind. The Vitis AI development environment hides the complexity of the underlying FPGA and ACAP from the user, making it simple for non-FPGA experts to create deep learning inference applications.

4.3. Machine Learning model development flow

The following figure is a block diagram of a machine learning flow for TB classification from chest X-ray images.

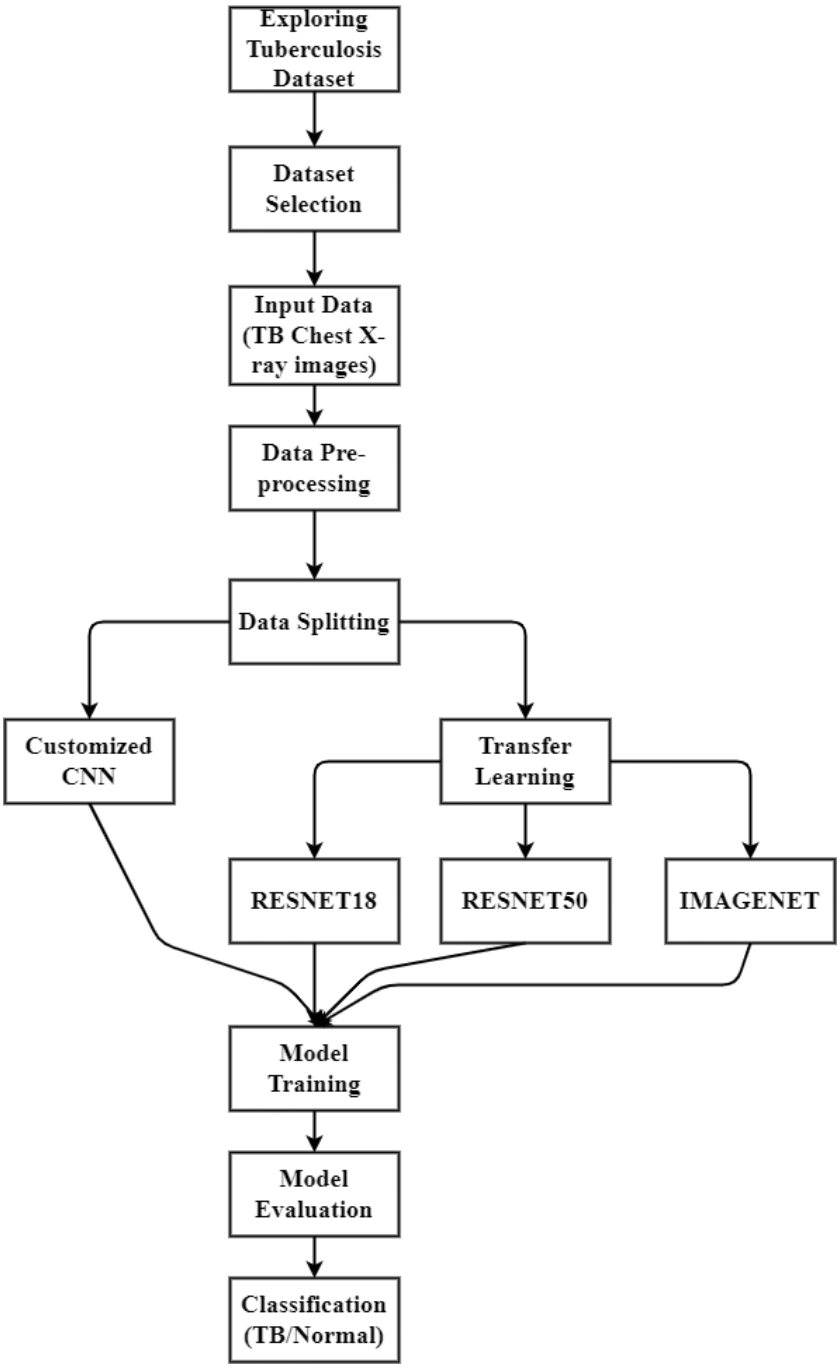

Figure 22: Overview of Chest X-ray Image Classification

4.3.1. Dataset

We explored two datasets of Tuberculosis, The TB Chest X-ray Database dataset from Kaggle which contains 4200 images, of which 3500 are normal and 700 are TB images. The XTBTorch

Dataset from GitHub contains 800 scans, of which 394 are positive cases and 406 are negative cases. We chose the Kaggle dataset for the following reasons:

- It is larger, with 4200 images compared to 800 images in the XTBTorch dataset.
- It contains a more balanced mix of normal and TB images, with 3500 normal images and 700 TB images, compared to 406 negative cases and 394 positive cases in the XTBTorch dataset.
- It is more publicly accessible, as anyone can download it from Kaggle without signing an agreement.

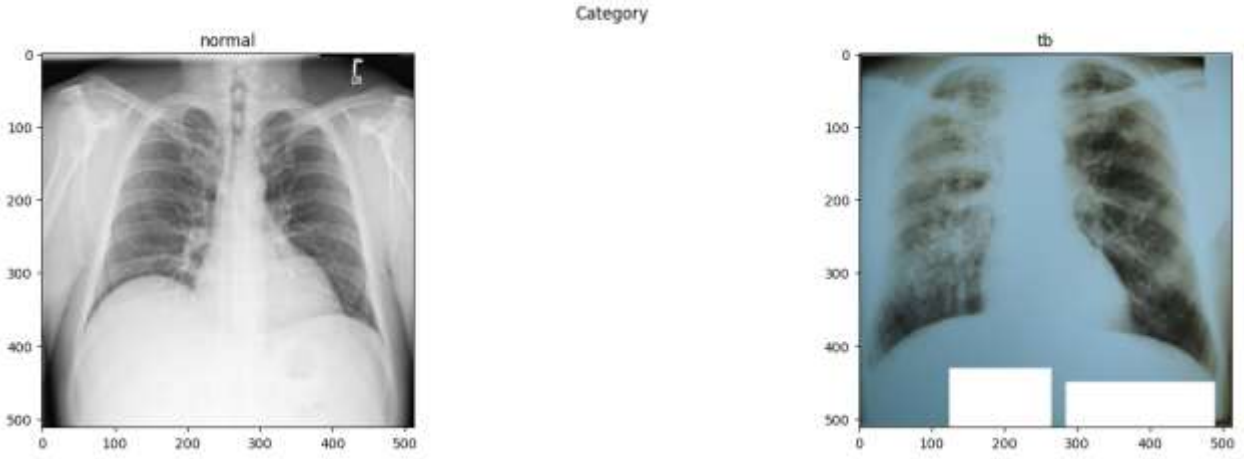

Figure 23: Chest X-ray Image for Tuberculosis and Normal

TB Chest X-ray Dataset contains a collection of 700 frontal chest X-ray images from patients with TB and 3500 normal chest X-ray images. The dataset was curated by Tawsifur Rahman, a data scientist, and includes images from different sources such as public datasets, publications, and hospitals. The dataset was created to aid in the development of deep learning models to automatically detect TB from chest X-ray images. The images were manually labeled by experienced radiologists as either TB positive or negative. The dataset also includes a Comma Separated Values (CSV) file with metadata for each image, including patient age and sex. Several deep learning models were developed using this dataset to detect TB from chest X-ray images. One of the models used was a CNN with four convolutional layers and two dense layers. The model achieved an accuracy of 91% on the validation set and 89% on the test set. Another model used was a transfer learning approach, where the ResNet-50 pre-trained model was fine-tuned on the TB dataset. This model achieved an accuracy of 94.4% on the test set.

4.3.2. Machine learning framework used.

We used two frameworks to train our models. Namely: TensorFlow, and PyTorch.

TensorFlow, a machine learning software library, was developed by Google and is available as an open-source platform [80]. The framework possesses a high degree of flexibility and robustness, rendering it suitable for a diverse range of applications, encompassing deep learning, natural language processing, and computer vision [81]. TensorFlow is widely recognized for its robust capabilities in facilitating production deployment, rendering it a popular choice among numerous companies operating within the technology sector [82].

PyTorch, an open-source machine learning framework developed by Facebook, has gained significant interest in recent years due to its user-friendly nature and dynamic computational graph capabilities [83]. This framework is particularly advantageous for academic research and experimentation, as it facilitates rapid model construction and prototyping for researchers. Additionally, PyTorch is designed for Python programming and GPU acceleration, making it suitable for a wide range of machine learning applications [76]**.**

Rationale for choosing TensorFlow and PyTorch

In the context of training a model for TB detection, there are several reasons why TensorFlow and PyTorch are good choices:

- Both frameworks are well-suited for deep learning tasks. Deep learning is a powerful approach to image recognition, which is a key task in TB detection.
- Both frameworks have strong support for transfer learning. Transfer learning is a technique that allows you to use a pre-trained model to solve a new problem. This can be a valuable tool for TB detection, as there are several pre-trained models available for image recognition tasks.
- Both frameworks have large communities of users. This means that there are many resources available to help you learn how to use the frameworks and solve problems.

Finally, we also used two platforms to train our different models. We used Google Colab and Jupyter Notebook. We opted to do so because we wanted to reduce bias among the best models which we want to choose before going on further to be used on Kria KV260.

Chapter 5

Results and Analysis

In this section we present the results of our work, and a detailed explanation of our findings.

5.1. Training for The Regular Computer

As discussed in chapter 4, where we showed the overview of the training process of the model. We trained four different models using both Frameworks TensorFlow and PyTorch and in both platforms Google Colab and Jupyter Notebook. We include several Metric measures such as Precision, Recall, Accuracy, and F1 score. The goal of training all these models was to find which model will perform better than the others. Then after selecting the best model from these training, we go on training the model for the Kria Kv260 based on the selected best model. We used two approaches to train these models, where we had to buy in the **Transfer Learning** and **Customized CNNs**.

5.1.1. Transfer Learning

It is a machine learning technique that involves using pre-trained models mostly for feature selection. We used three different pre-trained models: ResNet18, ResNet50, and ImageNet-V2. These models are all CNNs, which are a type of neural network that is particularly well-suited for image classification tasks. The results are obtained from each model which we have used.

5.1.1.1. ResNet18

We splitted the dataset into 80% of the dataset was for training and 20% of it was for testing. Then, after training the model we observed the following results. There were 50 epochs with 10 batch sizes.

Table 4: Results of ResNet18

	Precision	**Recall**	**F1-score**	**Support**
No TB	0.81	0.77	0.79	60
TB	0.78	0.82	0.80	60
Accuracy			0.79	120
Macro avg	0.79	0.79	0.79	120
Weighted avg	0.79	0.789	0.79	120

The table shows the precision, recall, F1-score, and support for two classes: No TB and TB. Precision is the fraction of predicted positives that are positive. Recall is the fraction of actual positives that are correctly predicted. F1-score is the harmonic mean of precision and recall. Support is the number of samples in each class. The **precision** for both classes is high, indicating that the model is good at predicting whether a sample has TB. The **recall** for No TB is higher than the recall for TB, which means that the model is better at predicting samples that do not have TB than samples that do have TB. This is likely because there are more samples of No TB than TB in the dataset. The **overall** accuracy of the model is 79%. This means that the model correctly predicts 79% of the samples in the dataset. The macro average precision, recall, and F1-score are all 79%. This means that the model is performing similarly on both classes.

The weighted average precision, recall, and F1-score are all 79%. This means that the model is performing slightly better on the No TB class than the TB class, because the No TB class has more samples.

Training Loss and Validation Loss on ResNet18

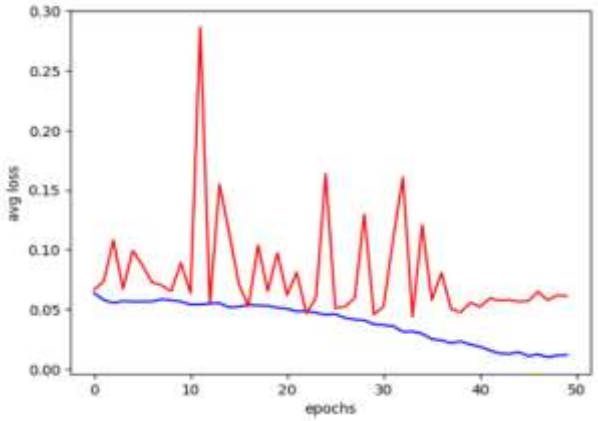

Figure 24: Training and Validation Average Loss for ResNet18

The graph shows the average loss for both training and testing the model. The red line shows the average loss of Training, while the blue line shows average loss of testing.

The graph shows the average loss over the course of 50 epochs. The average loss is a measure of how well the model is performing on the training data. The lower the average loss, the better the model is performing. The graph shows that the average loss is decreasing over time, which means that the model is getting better at predicting the training data. The model starts off with a high average loss, but it quickly decreases to a low average loss by epoch 30. After epoch 30, the average loss continues to decrease slowly, but it eventually reaches a plateau. The plateau means that the model is not able to improve its performance on the training data any further. This is often because the model has learned all the patterns in the training data that it can learn. Overall, the graph shows that the model can learn from the training data and improve its performance over time. However, the model eventually reaches a point where it is not able to improve its performance any further.

5.1.1.2. ResNet50

There were 50 epochs with 10 batch sizes.

Table 5: Results of ResNet50

	Precision	**Recall**	**F1-score**	**Support**
No TB	0.80	0.73	0.77	60
TB	0.75	0.82	0.78	60
Accuracy			0.78	120
Macro avg	0.78	0.77	0.77	120
Weighted avg	0.78	0.78	0.77	120

The **precision** for TB is 0.75, which means that 75% of the cases that the model predicted to have TB did have TB. The precision for No TB is 0.80, which means that 80% of the cases that the model predicted to not have TB did not have TB. The **recall** for TB is 0.82, which means that the model correctly identified 82% of the cases that had TB. The recall for No TB is 0.73, which means that the model correctly identified 73% of the cases that did not have TB. **F1 score** is a harmonic mean of precision and recall, which balances the two metrics. In this case, the F1 score for TB is 0.78, and the F1 score for No TB is 0.77, and there were 60 samples in each class. the overall accuracy is 0.78. **Macro avg** is the unweighted average of the precision, recall, and F1 score for each class. In this case, the macro avg for precision is 0.77, the macro avg for recall is 0.78, and the macro avg for F1 score is 0.77. **Weighted avg** is the weighted average of the precision, recall, and F1 score for each class, where the weights are the number of samples in each class. In this case, the weighted avg for precision is 0.78, the weighted avg for recall is 0.78, and the weighted avg for F1 score is 0.77.

Training Loss and Validation Loss on ResNet50

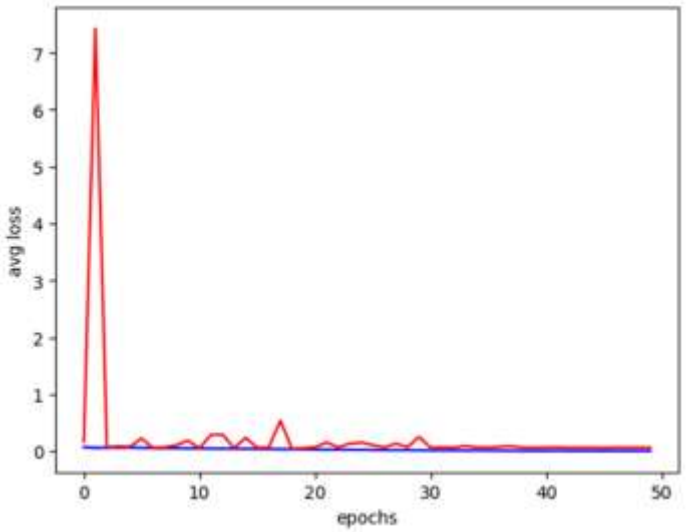

Figure 25: Training and Validation Average Loss for ResNet50

The graph shows the training and validation loss of a model trained using ResNet50. The red line represents training loss, while the blue line represents validation loss. The training loss decreases steadily throughout the training process, while the validation loss initially decreases but then starts to increase after about 20 epochs. This suggests that the model is overfitting the training data after 20 epochs. The training loss plateaus after about 30 epochs. This suggests that the model is no longer learning from the training data. The validation loss starts to increase after about 20 epochs. This suggests that the model is overfitting the training data after 20 epochs.

5.1.1.3. IMAGENET-V2

We used IMAGENET-V2 for feature extraction, there were 3 epochs. The overall Training performance of the model yielded 98.7% and Validation of 98.73%

Table 6: Results of IMAGEMET-V2

	Precision	Recall	F1-Score	Support
No TB	0.99	1.00	0.99	527
TB	1.00	0.92	0.96	103
Accuracy			0.987	630
Macro avg	0.99	0.96	0.98	630
Weighted avg	0.99	0.99	0.987	630

The results show that the model is very good at predicting both positive and negative cases. The precision for both TB and No TB is very high, at 0.99 and 1.00 respectively. This means that the model correctly identifies a high proportion of positive cases. The recall is also high for both TB and No TB, at 0.92 and 1.00 respectively. This means that the model misses very few positive cases. The overall accuracy of the model is 0.987. This means that the model correctly identifies 98.7% of all cases. The F1-score is also high, at 0.987. This is a harmonic mean of the precision and recall, and it is a good measure of the overall performance of the model. Overall, the results are very good. The model can accurately predict both positive and negative cases, and it has a high overall accuracy.

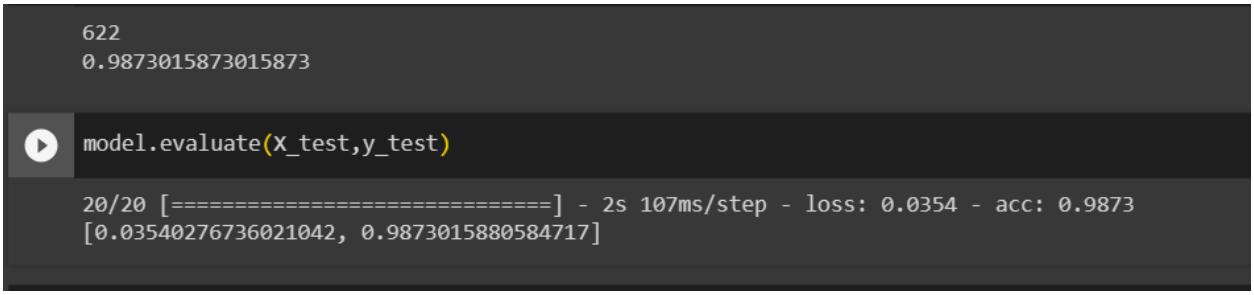

Figure 26: Training Accuracy, Validation Accuracy, and Validation loss

When we were training using IMAGENET-V2 we did include training loss and validation loss history, but we just did it at the evaluation stage. Because we learnt from the training where we used RESNET18 and RESNET50 where our model was overfitting the dataset. Then we introduced early stopping and reduced the epochs to 3. Then we came up with a Confusion Matrix to evaluate further our model.

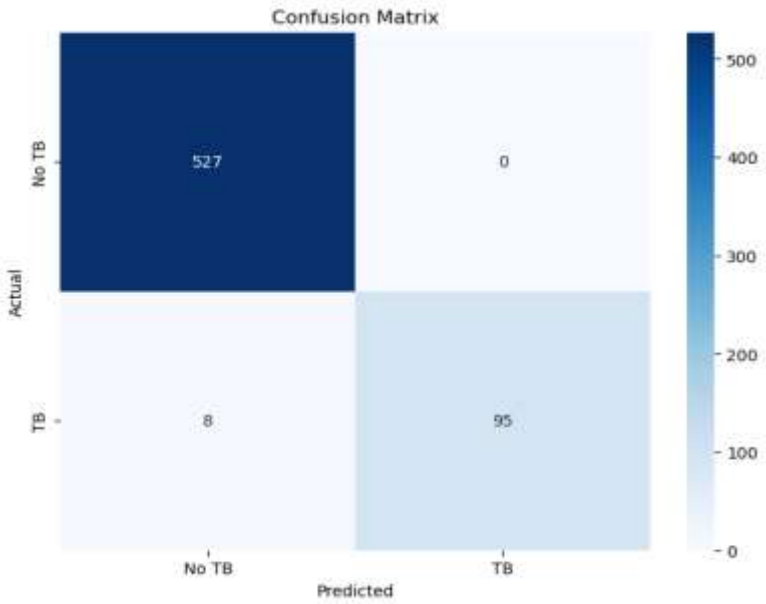

Figure 27: Confusion Matrix for IMAGENET-V2

Figure 27 is a confusion matrix that shows the performance of a classification model. The rows of the matrix represent the actual classes, and the columns represent the predicted classes. Each cell in the matrix shows the number of instances that were predicted to be in a particular class, given that they were in a particular class. The confusion matrix for the classification model in the image shows that the model predicted 95 instances correctly as TB, and 8 instances incorrectly as TB. The model also predicted 527 instances correctly as No TB, and 0 instances incorrectly as No TB.

5.1.2. Customized CNNs

Unlike with the transfer Learning where we were applying pre-trained models to extract features, but here we must customize our network and define some parameters to yield high performance of the model. The network starts with a series of **convolutional layers** which help in feature extraction. Then, there's a transition to **fully connected layers** responsible for classification.

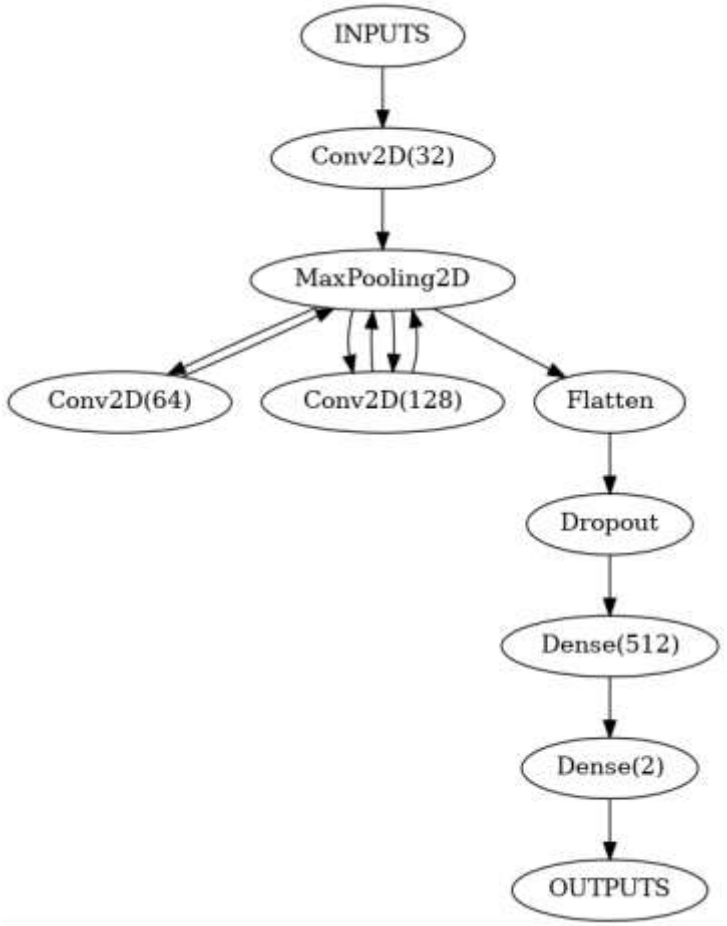

Figure 28: Overview of Custom CNN

Figure 28 represents how the custom CNN model is designed. The input layer takes an image of shape as input. The data then passes through a series of convolutional layers first convolution layer with 32 filters with ReLU activation function, then another convolution layer with 64 filters also use ReLU activation function, then two other convolution layers with 128 filters also ReLU activation function was used, each followed by a max-pooling layer. The convolutional layers extract features from the image, and the max-pooling layers reduce the dimensionality of the data. After passing through the convolutional and max-pooling layers, the data is flattened and passed through a dropout layer, which randomly drops some of the neurons to prevent overfitting. The data then passes through a fully connected layer with 512 neurons, followed by another fully connected layer with 2 neurons and a SoftMax activation function, which outputs the probability of each class.

5.1.1.1. Convolutional Layers (conv_layer)

The model consists of a sequence of convolutional layers, pooling layers, and activation functions. The convolutional layers extract features from the input image, while the pooling layers reduce the dimensionality of the feature maps. The activation functions introduce non-linearity into the model, which is important for learning complex patterns.

```
print(model)

Net(
  (conv_layer): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): ReLU(inplace=True)
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Dropout2d(p=0.05, inplace=False)
    (13): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): ReLU(inplace=True)
    (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
```

Figure 29: Convolutions Layers

The following is a detailed explanation of each layer in the model:

Conv2d: This layer performs a convolution operation on the input image. The convolution operation is a linear operation that is used to extract features from the image. The kernel size of the convolution operation specifies the size of the filter that is used to extract the features. The stride of the convolution operation specifies how much the filter is moved across the image. The padding of the convolution operation specifies how much the image is padded to ensure that the output feature map has the same dimensions as the input image.

BatchNorm2d: This layer performs batch normalization on the output feature map of the previous layer. Batch normalization helps to stabilize the training process and improve the performance of the model.

ReLU: This layer applies a Rectified Linear Unit (ReLU) activation function to the output feature map of the previous layer. The ReLu activation function is a nonlinear function that outputs the input value if it is positive, and zero otherwise.

MaxPool2d: This layer performs max pooling on the output feature map of the previous layer. Max pooling reduces the dimensionality of the feature map by taking the maximum value in a rectangular window. The kernel size of the max pooling operation specifies the size of the window that is used to perform the max pooling operation. The stride of the max pooling operation specifies how much the window is moved across the feature map.

Dropout2d: This layer applies dropout to the output feature map of the previous layer. Dropout is a regularization technique that helps to prevent overfitting. Overfitting is a situation where the model learns the training data too well and is unable to generalize to new data. Dropout works by randomly dropping out neurons from the network during training. This forces the network to learn more robust features.

5.1.1.2. Fully connected layer ((fc_layer)

The fully connected layers are responsible for taking the output of the convolutional layers and converting it into a vector of logits, where each logit represents the probability of the input image belonging to a particular class.

```
  (fc_layer): Sequential(
    (0): Dropout(p=0.1, inplace=False)
    (1): Linear(in_features=4096, out_features=1024, bias=True)
    (2): ReLU(inplace=True)
    (3): Linear(in_features=1024, out_features=512, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.1, inplace=False)
    (6): Linear(in_features=512, out_features=2, bias=True)
  )
)
```

Figure 30: Fully Connected Layer

Linear: This layer performs a linear transformation on the input feature map. The linear transformation is a linear operation that is used to combine the features from the previous layer. The input and output dimensions of the linear layer are specified by the in_features and out_features arguments, respectively.

ReLU: This layer applies a rectified linear unit (ReLU) activation function to the output feature map of the previous layer. The ReLU activation function is a nonlinear function that outputs the input value if it is positive, and zero otherwise.

Dropout: This layer applies dropout to the output feature map of the previous layer. Dropout is a regularization technique that helps to prevent overfitting. Overfitting is a situation where the model learns the training data too well and is unable to generalize to new data. Dropout works by randomly dropping out neurons from the network during training. This forces the network to learn more robust features.

The output of the last layer is a vector of logits, where each logit represents the probability of the input image belonging to a particular class. The class with the highest logit is predicted to be the class of the input image. When training the model using customized CNN, we used 5 epochs, and we got the below results.

Table 7: Results of Customized CNN

Training		**Validation**	
Train Loss	Train Accuracy	Validation Loss	Validation Accuracy
0.0086	98.91	0.0093	99.05

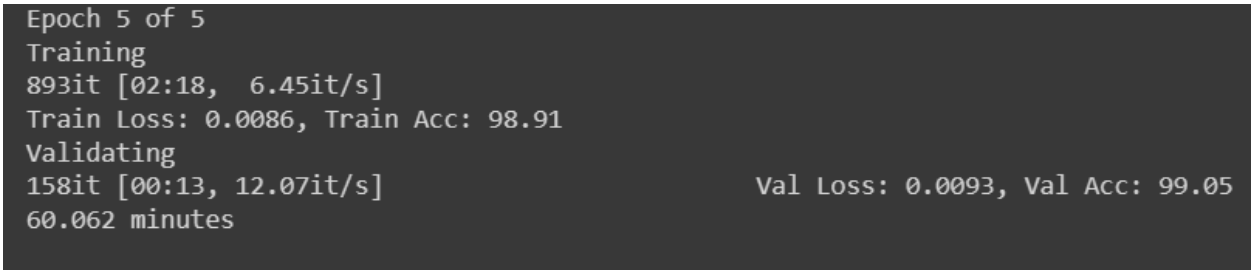

Figure 31: Results of Customized CNN model

The results in figure 31 and table 7 show that our customized CNN was able to achieve a train loss of 0.0086 and a train accuracy of 98.91% after 5 epochs. The validation loss and validation accuracy were 0.0093 and 99.05%, respectively. This suggests that our CNN can generalize well to unseen data.

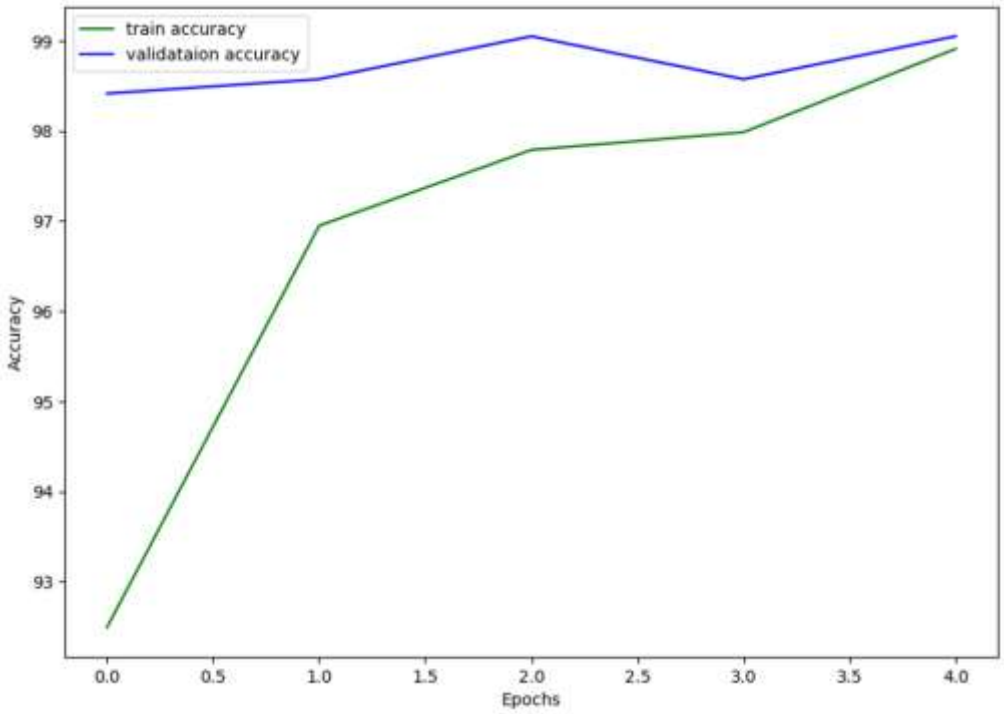

Figure 32: Training Accuracy and Validation Accuracy

The figure 32 shows the training and validation accuracy of the customized CNN model over 5 epochs. The training accuracy is higher than the validation accuracy for all epochs, which is a good sign. The training accuracy increases from 93% to close to 99% over the 5 epochs, while the validation accuracy increases from 97% to 99%. This suggests that the model is learning well.

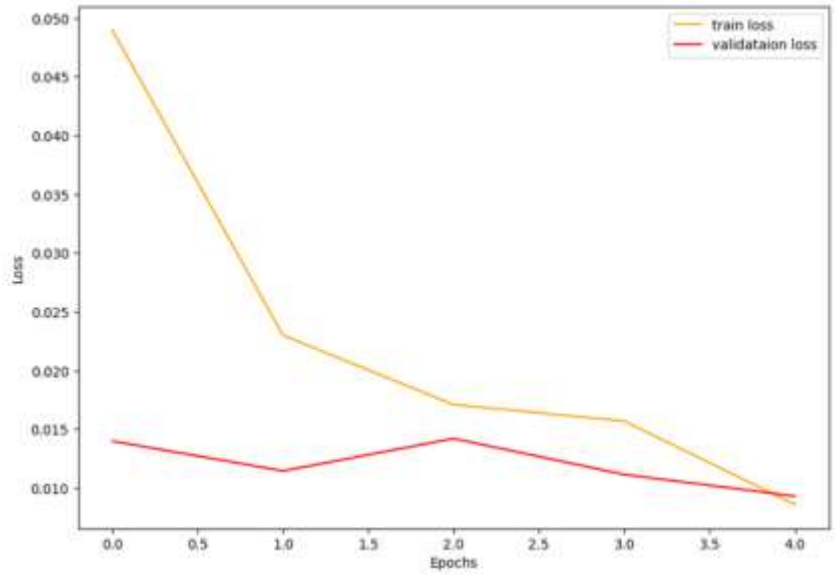

Figure 33: Training Loss and Validation Loss

The figure 33 shows that both the training loss and validation loss decrease over time, which is a good sign. However, the training loss is consistently lower than the validation loss, which suggests that the model may be overfitting the training data.

5.1.2. **Comparison of training with various ML models**

We trained the model in different ways. We divided the training process by training using **transfer learning** and **customized CNN**. We tried to use three pre-training models which consist of RESNET18, RESNET50 and IMAGENET-V2, and for **Customized CNN**. Now we want to compare our trained models so that we can choose the best model. The metrics used to evaluate their performance are Training Accuracy and Testing Accuracy.

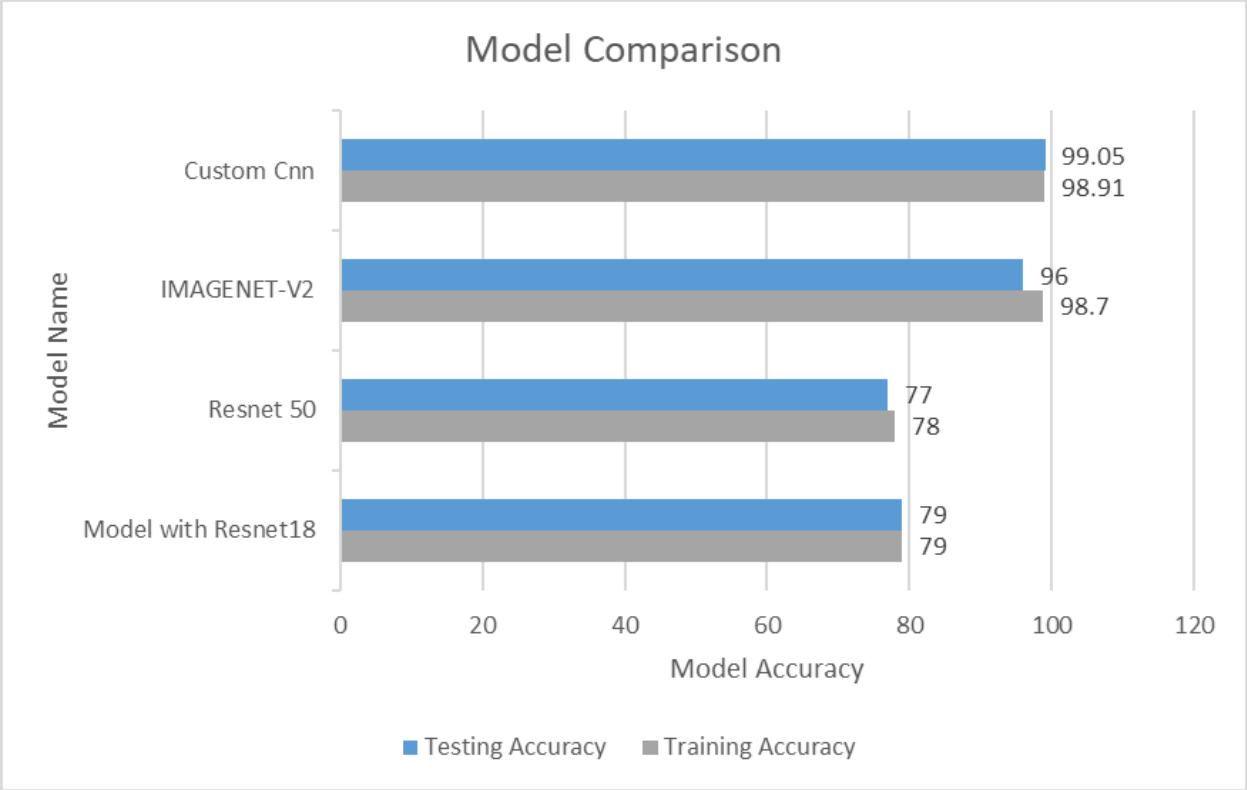

Figure 34: Model Comparison Bar graph

Based on the figure 34 above, the custom CNN has the highest testing accuracy of 99.05%, followed by the ImageNet-V2 with 98.91%. The ResNet-50 has a testing accuracy of 77%, and the model with ResNet-18 has a testing accuracy of 79%. **Custom CNN:** This model has the highest testing accuracy, suggesting that it is the most reliable model for making predictions on new data. However, it is also the most complex model, which means that it may take longer to

train which has been evidenced in figure 30 where it took 60 minutes which is the longest time compared to other models. **ImageNet-V2:** This model has a slightly lower testing accuracy than the custom CNN, but it is also less complex. This means that it may be a better choice for applications where training time and deployment resources are limited. **ResNet-50:** This model has a lower testing accuracy than the custom CNN and ImageNet-V2 models. However, it is also the least complex model, making it a good choice for applications where simplicity is important. **Model with ResNet-18:** This model has the lowest testing accuracy of all the models. This suggests that it is the least reliable model for making predictions on new data.

Overall, the **customized CNN** is the best performing model in terms of testing accuracy since it is the most important factor. Then we choose the customized CNN model to go further with it to train the model for Kria KV260.

5.2. Model Evaluation Metrics According to the Objectives

To assess the effectiveness of a machine learning model designed to detect tuberculosis from X-ray images, we used a variety of metrics that evaluate different elements of the model's performance. The assessment criteria according to the objectives of this study consist of Accuracy, **Throughput**, **power consumption**, and **latency**. These metrics have been collected in both when running an inference on a regular computer, and Kria Kv260.

Accuracy: When evaluating the overall performance of a model, one common evaluation metric that is used is the accuracy rating. It is a measure of the proportion of samples out of the total number of samples that have been correctly classified (including both true positives and true negatives). Based on this research, Accuracy refers to the model's ability to correctly classify X-ray images as either tuberculosis or normal. Accuracy is calculated as

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{TP} + \boldsymbol{TN}}{\boldsymbol{TP} + \boldsymbol{TN} + \boldsymbol{FP} + \boldsymbol{FN}}$$

where **TP** is the number of true positives (correctly predicted tuberculosis cases), **TN** is the number of true negatives (correctly predicted normal cases), **FP** is the number of false positives (normal cases incorrectly classified as tuberculosis), and **FN** is the number of false negatives (tuberculosis

cases incorrectly classified as normal). Accuracy provides an initial sense of how well the model is performing in terms of classification correctness. The model's accuracy provides an initial indication of how well it is performing in terms of the correctness of its classifications [84]. A high level of accuracy is essential in the field of medicine because errors in patient classification can have major consequences for the patient's health. It is essential to strike a balance between sensitivity (the ability to correctly identify positive cases) and specificity (the ability to correctly identify negative cases) to avoid both false positives and false negatives in diagnostic testing.

Throughput: The number of inferences that can be drawn from the model in each amount of time, usually per second, is referred to as its throughput. often expressed in predictions per second (PPS) or inferences per second (IPS) or Frame per second (FPS). In this context Throughput measures the number of X-ray images that can be processed by the model in each time, and it is measured empirically by running the model on a real or synthetic workload and recording the number of inferences it can perform within a specific time frame. It is a crucial factor, particularly in applications that require real-time processing or high throughput. For Instance, in a clinical environment, it is necessary to make certain that the model can process a high volume of X-ray images in an effective manner to facilitate a quick diagnosis [85]. It is necessary to have the capacity to process many images in a short amount of time, particularly in settings with many patients.

Power Consumption: During the process of model inference, the term "power consumption" refers to the quantity of electrical energy that is consumed by the hardware (such as GPUs, CPUs, and accelerators). The percentage of time that the CPU, GPUs, and DPUs are being used can be calculated from the power consumption. Power consumption is a very vital evaluation criterion, particularly for embedded systems or other devices with limited power resources. A lower power consumption enables a longer battery life and reduces the energy requirements of the model when it is being inferred [86]. Models that require less power to operate are preferable for use in settings with limited resources.

Latency: It is the amount of time it takes for the model to make a prediction after an inference request has been made. In medical applications, low latency is critical for real-time or near-real-time diagnosis and decision-making. Delays in patient care may result from excessive latency.

5.3. Deep Learning Processing Unit Evaluation

As it is shown on section 4.1 the Deep Learning Processing Unit (DPU) is the main component where the inference of the trained model take place. As such we must show how the resources are being used in this unit. Under this section we will show the resource usage of the DPU, the power consumed by the DPU, and finally the timing.

5.3.1. Resource Utilization of DPU

Table 8: Resource Usage by DPU IP

NAME	UTILIZATION
CLB LUTS	7851 (6.70%)
CLB Registers	8870 (3.79%)
LUT as Logic	6814 (5.82%)
LUT as Memory	1037 (1.80%)
Block RAM	0 (0.00%)
DSPs	6 (0.48%)

Table 8 shows the resource utilization of the DPU IP. **CLB LUTs** refers to the number of Configurable Logic Blocks (CLBs) used as Look-Up Tables (LUTs). LUTs are the fundamental building blocks of FPGAs and can be configured to implement various logic functions. The DPU IP uses 7851 CLB LUTs, which represents 6.70% of the total available CLB LUTs. Where **CLB Registers** refers to the number of CLBs used as registers. Registers are used to store temporary data and state within the DPU IP. The DPU IP uses 8870 registers, which represents 3.79% of the total available registers. **LUT as Logic** refers to the number of LUTs used to implement logic functions. Out of the 7851 LUTs used by the DPU IP, 6814 LUTs are used for logic functions, which represents 5.82% of the total available LUTs, Whilst **LUT as Memory** refers to the number of LUTs used to implement memory functions. Out of the 7851 LUTs used by the DPU IP, 1037 LUTs are used for memory functions, which represents 1.80% of the total available memory LUTs. **Block RAM** refers to the number of Block RAM (BRAM) resources used by the DPU IP. BRAMs are specialized memory blocks within the FPGA that offer high performance and flexibility. The DPU IP does not use any BRAMs in this case. **DSPs** refers to the number of Digital Signal

Processing (DSP) slices used by the DPU IP. DSP slices are specialized hardware blocks optimized for performing various signal processing operations. The DPU IP uses 6 DSP slices, which represents 0.48% of the total available DSP slices.

The DPU IP utilizes a relatively small percentage of the available FPGA resources, leaving ample room for other components in the design which allows us to add additional features or functionalities to the design without exceeding the FPGA's capacity.

5.3.2. **Power Utilization**

Power Utilization of the DPU IP can be affected by several factors, one which is the environment where the device is operating. In this context, the environment settings for a power analysis used 25 °C (Degrees Celsius) which is the normal room temperature. 2.417 Watts of Dynamic power representing 95% is being used by Processing Systems (PS) where Application Processing Unit (APU), Real-Time Processing Unit (RPU) and other components. Soon after powering up the device these components start using power. Whilst the DPU IP uses 0.04Watts which is broken down into several components. As shown in the table below.

Table 9: DPU IP Power Utilization

Component	**Power (W)**	**Description**
Clocks	0.014	Power consumed by the DPU's internal clocks.
Signals	0.009	Power consumed by the data signals entering and exiting the DPU.
Data	0.008	Power consumed by processing and manipulating data within the DPU.
Clock Enable	0.001	Power consumed by the logic that enables and disables the DPU's clocks.
Set/Reset	<0.001	Power consumed by the logic that sets and resets the DPU's internal state.
Logic	0.012	Power consumed by the DPU's logic circuits, which control its operation.
DSP	<0.001	Power consumed by the DPU's DSP hardware, used for specific calculations.
PS	<0.001	Power consumed by the DPU's power supply circuitry.

5.3.3. **Timing Summary**

The DPU clock is provided by the ZYNQ PS at 100 MHz (corresponding to a period of 10 nanosecond). However, the timing analysis shows a worst negative slack (WNS) of 14.287 ns, allowing the design to be clocked in the order of gigahertz.

5.4. Training a model for Kria KV260

After the model had been trained for regular computers, the model was saved in (.h5) format. The most important point is. We cannot directly use this model rather it should undergo another training process which is a bit different from the normal training model. The selected model is called Float Model because of its nature. It is of (INT32) but Kria KV260 will not be able to run such a model because of the nature of the hardware which is used in Kria KV260.

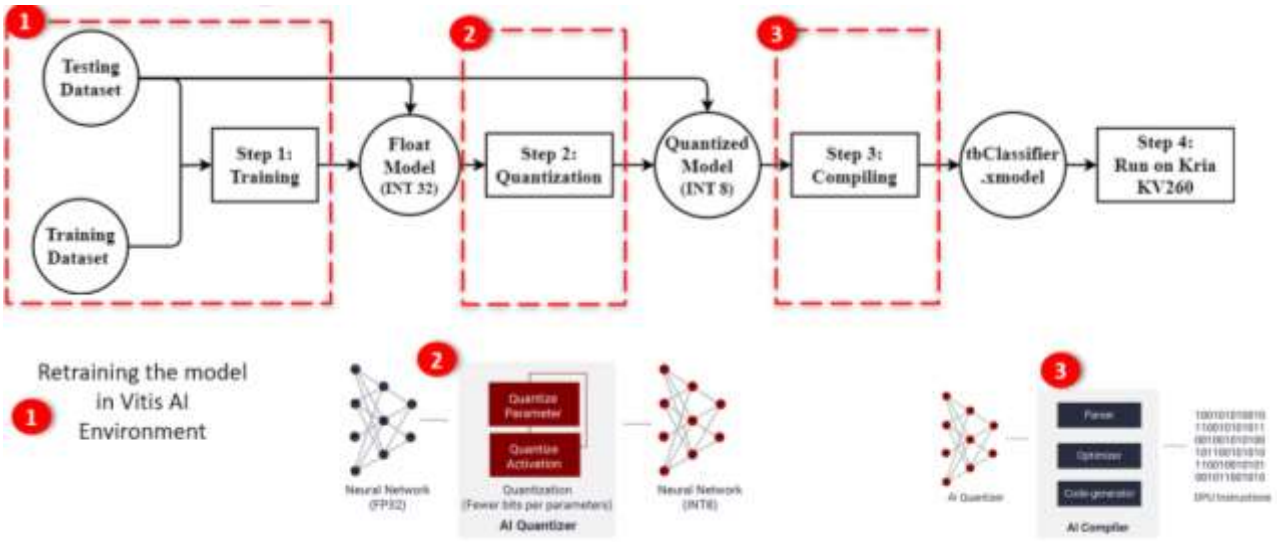

Figure 35: Training a model for Kria KV260 flow.

The flow was supposed to start from Quantization of the model using the model which has been selected. But this is different in the way because we must set up some environment to do all the processes required. There have been incompatibility issues with the model which was trained in for the regular computer due to two things:

Table 10: Customized CNN trained for Regular computers and Kria KV260

	Trained for Regular Computer	**Trained for Kria KV260**
Framework	PyTorch	TensorFlow
Size of Input	(224,224,3)	(100,100, 1)

In table 10, Summarizes some of the changes which were made on the chosen model. One being the customized model which was selected was developed based on PyTorch but transforming to Training in the New environment I had to change to Tensorflow2. I used the same Architecture to design the network but this time it was used in Tensorflow2. Another reason was the input size of the images. The one which was trained for regular computers could allow the input (224,224,3) which accept the RGB scale whilst the one which was trained for Kria KV260 could allow (100,100,1). This has pushed us to change a few parameters in our Custom CNN. The first step we started with retraining the model in Vitis AI environment. Where we involved the whole process of training the model by using the same customized CNN, after training the model now this model is in the float-point format which is 32 bits. The second step is the quantization process. Under this process is where we reduce the bits per parameters in our neural network. Basically, what is done here is the AI quantizer; the library which is available in the Vitis AI is used for the whole quantization process. AI quantizer convert the 32-bit floating-point weights and activations to fixed-point like INT8, the AI Quantizer can reduce the computing complexity without losing prediction accuracy. The fixed-point network model requires less memory bandwidth, thus providing faster speed and higher power efficiency than the floating-point model. At the end we have the quantized model which is a fixed-point model meaning the model which has 8 bits per parameter. Then it is from this step where we proceed with the compilation process where the input now is the quantized model into AI compiler library which is also available in Vitis AI. The AI Compiler maps the AI model to a highly efficient instruction set and dataflow model. It also performs sophisticated optimizations such as layer fusion, instruction scheduling, and reuses on-chip memory.

5.4.1. **Setting up the environment**

The environment was based on Ubuntu 22.04 LTS, we cloned the Vitis AI 2.5 from the Xilinx-AMD GitHub page. Then we run the docker command since the Vitis AI is a docker image. From there we activate the tensorflow2 workflow inside Docker. Once the environment is activated, that is when we start the process of training the model for Kria KV260.

5.4.2. **Training the Model**

Training the model, we used the edited customized CNN we had from the previous training as it is mentioned in table 8. But this time we will train in Vitis AI 2.5 environment, and we also used Vitis AI 2.5 CPU not GPU and we used the same dataset Tuberculosis X-ray Images where 3500 are normal and 700 are TB images.

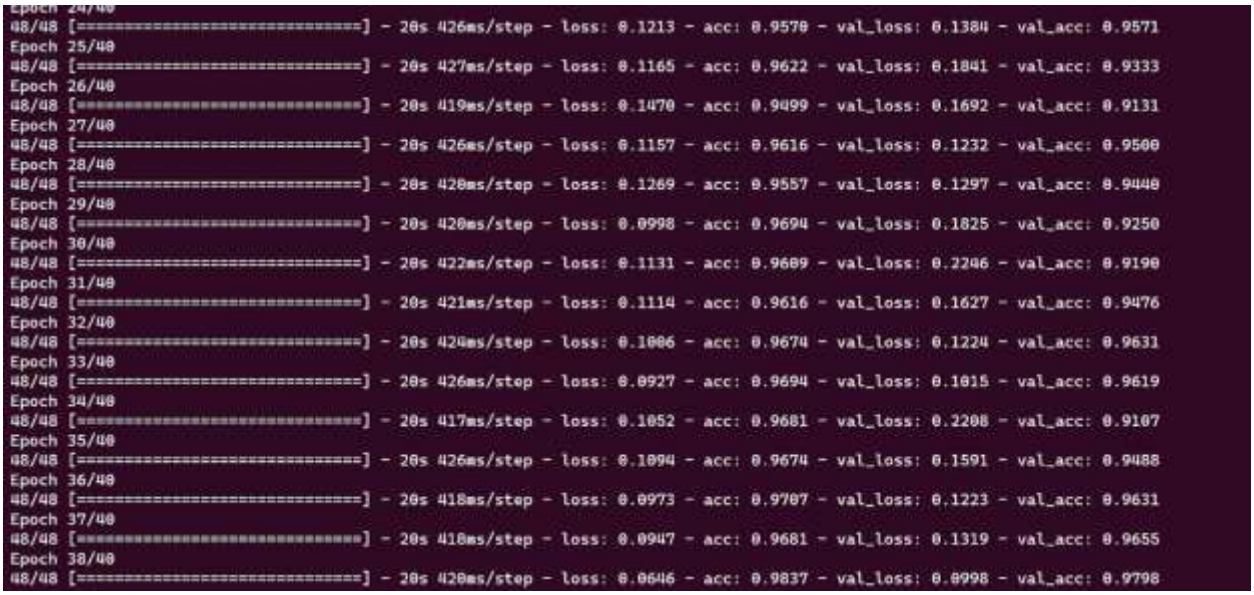

Figure 36: Training the model in Vitis AI 2.5 Environment

The results obtained show that the model trained in Vitis AI 2.5 with TensorFlow 2 has a training accuracy of 98.37% and a validation accuracy of 97.98%. This means that the model can correctly classify 98.37% of the images in the training set and 97.98% of the images in the validation set. The training loss is 0.0646 and the validation loss is 0.0998. Loss is a measure of how well the model is fitting the training data. A lower loss indicates that the model is fitting the training data better. The model which has been generated is called **TBclassifier.h5** and it is a float model.

5.4.3. **Evaluating the Float-Point Model**

After training the float model we have to evaluate it to get the results based on the evaluation mentioned in section 5.2. We considered some images which were not included in the training process or validation. To check if our model can give us intended results. We randomly choose 14 Chest X-ray images from another dataset which was left where 7 images were normal and 7 images were TB positive as you can see below.

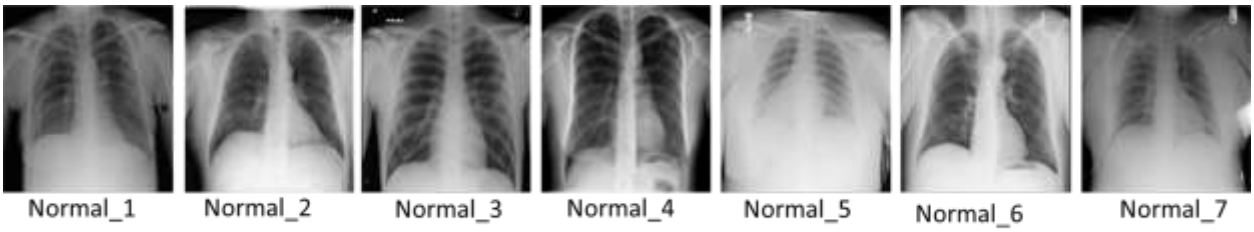

Figure 37: Chest X-ray images that are Normal.

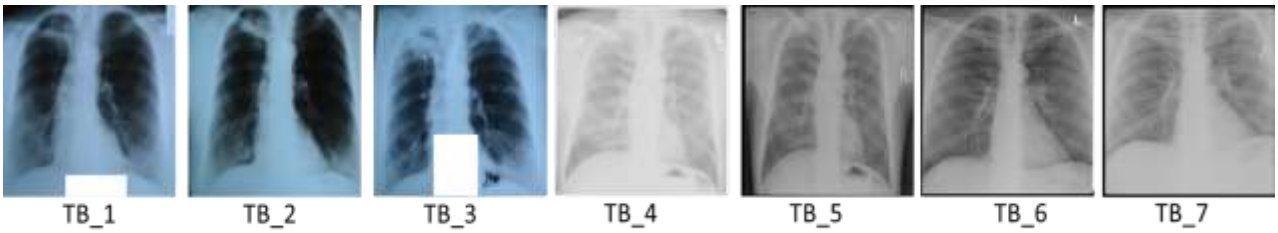

Figure 38: TB positive Chest X-ray images

We wrote the python script which was to specifically test the above images based on the evaluation metrics such as accuracy, throughput, latency, and power consumption which was possible with the help of **psutil library** which records the CPU utilization percentage during the inference.

	Image	Prediction	Accuracy (%)	Latency (s)	Throughput (pred/s)	CPU Utilization (%)
0	example_0.png	Normal	99.992406	0.193439	5.169575	12.6
1	ab.png	Normal	99.722326	0.037046	26.993152	22.2
2	ab1.png	Normal	99.999881	0.038010	26.309113	22.9
3	ab2.png	Normal	98.997164	0.036840	27.144085	22.2
4	ab3.png	Normal	99.773186	0.037050	26.990547	18.2
5	ab4.png	Normal	99.954790	0.037424	26.720758	22.9
6	ab5.png	Normal	99.941111	0.039176	25.525996	17.1
7	ab6.png	TB	99.999058	0.038612	25.898599	22.2
8	ab7.png	TB	99.996221	0.042025	23.795309	29.5
9	ab8.png	TB	96.740234	0.039366	25.402627	17.6
10	ab9.png	TB	83.705318	0.035731	27.986842	18.2
11	ab10.png	TB	99.431300	0.034147	29.285129	28.6
12	ab11.png	TB	53.350556	0.034967	28.598632	10.3
13	ab12.png	TB	99.957937	0.033580	29.779573	14.3

Total_Throughput:20.67 fps, Frames:14, Latency:0.68 seconds, CPU_Utilization:14.30

Figure 39: Evaluation of Float Model which was trained in Vitis AI 2.5

5.4.4. **Quantization of Float Model**

Quantization is the process of converting the trained, floating-point checkpoint into a fixed-point integer checkpoint. This is necessary because the Xilinx DPU family of ML accelerators run models and networks with parameters in integer format.

The input of this process is the float-point model, and the output will be the fixed-point model.

```
model input size: 100 100

Load validation dataset for quantization..
Found 4200 images belonging to 2 classes.

Run quantization..
[VAI INFO] Start CrossLayerEqualization...
10/10 [==============================] - 1s 135ms/step
[VAI INFO] CrossLayerEqualization Done.
[VAI INFO] Start Quantize Calibration...
132/132 [==============================] - 72s 525ms/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 Refninement 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.
in or evaluate the model.

Saved quantized model as ../output/tb_quantized_model.h5
(vitis-ai-tensorflow2) Vitis-AI /workspace/customized_ai_model/tbClass/code >
```

Figure 40: Quantization process

From figure 40 we use the tbclassifier.h5 model as an input which is a float-point model, and we were able to get a fixed-point model which was saved in the ./output/ directory and the model

was saved as **tb_quantized_model.h5**. But in this model, we must evaluate it. After evaluating it we got the following results. We use the AI Quantizer library for the quantization process.

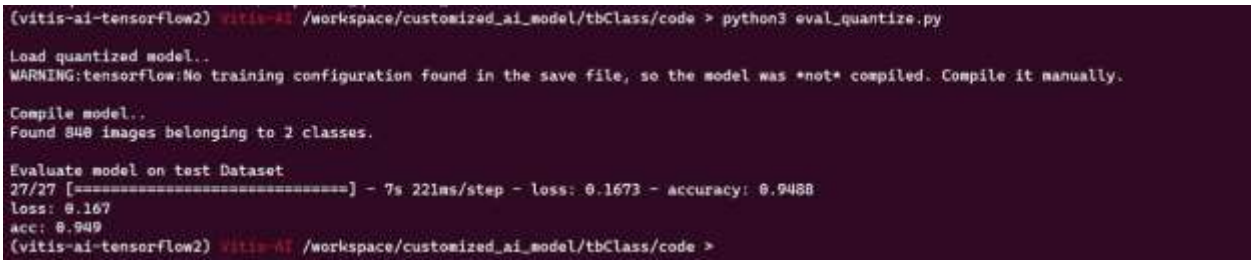

Figure 41: Evaluation of Quantized model

After we tested the quantized model, we were able to get the accuracy of 94.9 %. We used another type of images which were totally different from the training set.

5.4.5. **Compilation of Quantized model**

This step is where we convert the quantized model into the format which is recognized by the DPU. the model is converted from .h5 format into .xmodel.

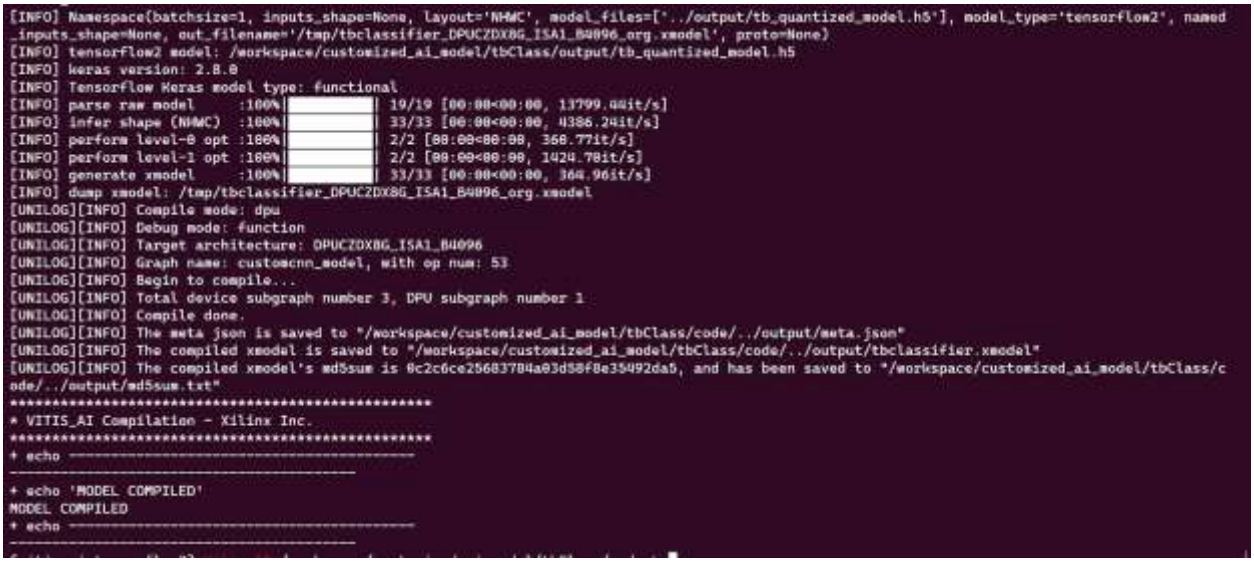

Figure 42: Compilation of Quantized model

Figure 42 shows the output after the model has been compiled, three files are being generated the meta.json which contain the fingerprint of the device. The compiled model in this case is tbclassifier.xmodel and the m5sum.txt.

5.4.6. **Running the application on Kria KV260**

This step needs to prepare some files before running into the Kria KV260. There is need to prepare the images, xmodel, and application code for copying to Kria KV260, follow these steps:

- Create a folder and inside the main folder create another folder where images will reside.
- Copy the .xmodel file into the main folder.
- Then copy the application code into the main folder also.
- Then finally copy the main folder which contains all the files mentioned above.

What is contained in the application code: Code implements a multi-threaded image classification application using the Vitis AI Runtime (VART) for hardware acceleration on the Kria KV260 platform. It takes a directory of images as input, runs them through a pre-trained CNN model, and outputs the classification results.

5.4.7. **Results after running an inference on Kria KV260**

After running the inference on Kria KV260, the following results were obtained, in which the first was taking the default 1 thread.

```
Command line options:
 --image_dir :  Pictures1
 --threads   :  1
 --model     :  tbclassifier_v3.xmodel
Pre-processing 14 images...
Starting 1 threads...
Throughput=1078.72 fps, total frames = 14, time=0.0130 seconds
Normal:14, TB:0, Accuracy:1.0000
```

Figure 43: The Kria KV260 classification Results 1 Thread

The results in the figure 43 above show that the model was able to classify all 14 chest X-ray images correctly, resulting in an accuracy of 100%. The CPU utilization was 1.09%. We also tried to add more threads than the results were as follows:

```
Command line options:
 --image_dir :  Pictures1
 --threads   :  3
 --model     :  tbclassifier_v3.xmodel
Pre-processing 14 images...
Starting 3 threads...
Throughput=1377.02 fps, total frames = 14, time=0.0102 seconds
Normal:14, TB:0, Accuracy:1.0000
```

Figure 44: The Kria KV260 classification Results with 3 Threads

The model was able to classify 14 images in 0.0102 seconds, with an accuracy of 100%means that the model was able to correctly classify all the images in the dataset. It suggests that the model is well-generalized and will be able to perform well on new data. This is a throughput of 1377.02 frames per second which is very fast. This means that the model can be used to classify images in real time. When running the inference, we have observed that the power consumed during inference is relatively low as it is indicated in the figure 45.

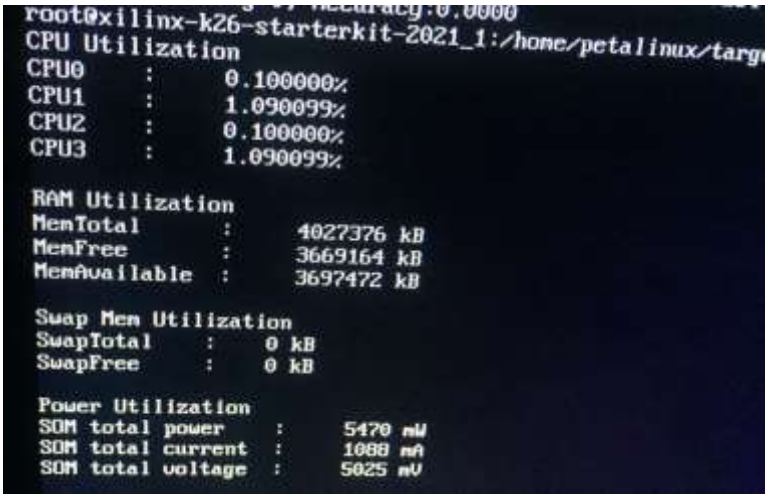

Figure 45: Resource Utilization when running Inference.

Figure 45 shows CPU utilization of a Kria KV260 board. The CPU utilization is the percentage of time that the CPU is busy running processes. The CPU utilization of the Kria KV260 board shows 1.090099%. This means that the CPU is idle for 98.909901% of the time. The SOM total power of the Kria KV260 board is 5470 mW. This means that the board is consuming 5.47 watts of power. Overall, the results from the attached image show that the Kria KV260 board is idle and consumes very little power.

5.4.8. **Discussion of Results**

The table below shows the data for comparison based on the below metrics. The results which are obtained on the CPU was calculated based on the average on that evaluation metric. The testing on the which CPU has the following features(Processor Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz, 2496 MHz, 4 Core(s), 8 Logical Processor(s)) Since in the CPU the data which was provided was tested on each image as its distinct item.

Table 11: Comparison of the results from KV260 and the regular Computer

	Number of Images	**Throughput (fps)**	**Latency (s)**	**Accuracy (%)**	**CPU Utilization (%)**	**Power Consumption (mW)**
Kria KV260 (1 Thread)	14	1078.72	0.0130	100	1.09	5470
Kria KV260 (3 Threads)	14	1377.02	0.0102	100	1.09	5470
Regular Computer (1 thread)	14	20.67	0.68	99.99	14.30	~Estimated 45000

The table 11 shows the performance metrics of two different hardware setups: the Kria KV260 and an Intel Core i5-10300H CPU. The Kria KV260 performs significantly better than the CPU, with a **throughput** of over 1377.02 frames per second compared to the CPU's 20.67 frames per second. This indicates that the Kria KV260 can process TB classification tasks at a much higher speed than the CPU.

Latency represents the time taken for a single inference or prediction. The Kria KV260 demonstrates remarkably lower latency (0.0102 seconds) compared to the CPU (0.68 seconds). Lower latency is favorable, especially in real-time applications, as it signifies quicker response times for inference tasks.

The **accuracy** for the Kria KV260 is indicated as 1, which represents 100% accuracy. On the other hand, the CPU shows an accuracy of 99.99%. However, it's essential to note that an accuracy of 1 for the Kria KV260 has been shown since it classifies the image to its category whether the image is Normal or TB.

Power consumption measures the amount of power used during inference tasks. The Kria KV260 consumes significantly less power (5.47W) compared to the CPU (~45W). Lower power consumption can be advantageous, especially in scenarios where energy efficiency is critical.

Table 12: Comparison of the results from State-of-the-art

	Our work	**Sandro et al**[87]		**Zakariah et al[49]**
Device	Kria KV260	Kria KV260	ZCU104	Intel® Arria® 10 GX 1150 FPGA
Throughput (FPS)	1,377.02	13.13	13.48	1,251
Latency (Seconds)	0.0102	-	-	0.712
Power Consumption (W)	5.47	11.28	19.57	-

Table 12 above shows the performance of different devices on a throughput, latency, and Power consumption benchmark. We used different devices but all of them have Programmable Logic where the FPGA fabric was implemented to run the model inference. Our work (Kria KV260), Sandro et al (Kria KV260, ZCU104), and Zachariah et al (Intel Arria 10 GX 1150 FPGA). Our work achieves the highest throughput of 1377.02 FPS, which is higher than the state-of-the-art results from Sandro et al and Zachariah et al. This work also achieves a very low latency of 0.0102 seconds, which is comparable to the state-of-the-art results from Zachariah et al. Additionally, this work achieves the lowest power consumption of 5.47 W, which is significantly lower than the state-of-the-art results from Sandro et al and Zachariah et al. Overall, the results in the table show that our work achieves a significant improvement in throughput, latency, and power consumption over the state-of-the-art results. This makes this work a very promising approach for deploying deep learning models on FPGAs in the field of X-ray image analysis.

Although our work performed better results, there are some factors that can be taken into consideration. Firstly, the devices were operating on clock frequency for our work we used 300MHz and for the other work they did not include the operating clock frequency. This is important because the throughput of a device is dependent on its clock frequency. Secondly, we were targeting different application as well as dataset used which could determine the size of the inputs of the model for instance Sandro et al used vineyard dataset where they used different

devices to. This is important because the performance of a device can vary depending on the target application.

5.4.8.1. Validation of Hypothesis

In previous section 1.5 we made some hypotheses for our study. The first Hypothesis based on power consumption where we claimed that FPGA-based SoC can consume lower power during the inference phase. This is supported in table 10. The Kria KV260 consumed significantly less power (5.47W) compared to the CPU (~45W). This substantial difference reinforces the advantage of FPGA-based SoC designs in offering energy-efficient solutions for deep learning inference tasks.

We also claimed that FPGA-Based SoC architecture will yield in high throughput and low latency with the FPGA-based SoC architecture. The Kria KV260 demonstrated a remarkably higher throughput (1377.02 fps) and significantly lower latency (0.0102 seconds) compared to the CPU (20.67 fps and 0.68 seconds, respectively). These results indicate that the FPGA-based SoC significantly outperforms regular computing platforms in terms of processing speed and response times for analyzing X-ray images related to TB classification.

In general, The FPGA-based SoC, in our context represented by the Kria KV260, demonstrates not only lower power consumption but also significantly higher throughput and lower latency compared to traditional CPU-based computing platforms. This suggests that FPGA-based architectures are indeed well-suited for fast and energy-efficient analysis of X-ray images in TB classification using deep neural networks.

Chapter 6

Conclusion and Future Work

In this section we conclude the research study where we will talk about the overall objective of the research and what has been done to archive it. Then we will delve deeper on future work.

6.1. Summary

While there is a need for fast and efficient TB identification. In our research we investigate the design of a SoC for X-ray image analysis and TB detection using FPGA-based architecture and machine learning. So, our methodology consisted of an approach that included several components. The hardware for the system was initially developed, with a particular focus on the integration of FPGA-based system-on-chip. The subsequent step consisted of training the deep learning model on a traditional computer, which was the first phase of the process. In the subsequent steps, the model was subjected to additional training and optimization within Vitis AI to get it ready for deployment on the Kria KV260 platform. During the research, the performance of the FPGA-based SoC was evaluated and compared to that of conventional CPU-based computing platforms. The metrics that were taken into consideration were power efficiency, throughput, accuracy, and latency. In the field of tuberculosis detection from X-ray images, the findings that were obtained from this comparative analysis yielded significant findings that highlighted the benefits of FPGA-based architectures, particularly the Kria KV260.

It is important to note that the Kria KV260 demonstrated a significantly lower power consumption during inference, which highlights the energy efficiency that is inherent in FPGA technology for deep learning work. In addition, it demonstrated a significantly higher throughput and a significantly lower latency, which is indicative of superior processing speed and rapid response times, both of which are essential for effective tuberculosis detection capabilities.

6.2. Future work direction

Advanced FPGA Architectures for AI: We would like to find out more about how state-of-the-art FPGA designs tailored for AI applications can be integrated. To further optimize and accelerate deep learning computations, we would like to investigate newer FPGA Architecture that have AI-

centric features like dedicated AI accelerators, improved memory structures, or specific instructions. Try out other architectures, such as adaptable neural networks or hardware-software co-designs focused on artificial intelligence, to see what works best for tuberculosis classification.

AI Model Optimization for FPGA: Investigate methods for enhancing AI models for use with FPGAs. Come up with ways to optimize, reduce, or quantify deep learning models so they fit inside FPGAs while keeping their accuracy. Examine methods for TB classification on X-ray images that are optimized for FPGA architectures in terms of model pruning, sparsity, or low-precision arithmetic, with the goal of maximizing performance and minimizing resource utilization.

Real-time Deployment and Edge Computing: Emphasize the use of FPGA-based edge devices for the real-time deployment of tuberculosis classification models. To facilitate on-device inference for rapid diagnosis and decision-making at the point of care, investigate the possibility of integrating FPGA-powered devices into healthcare infrastructure. Data preprocessing, model deployment, and result visualization are all aspects of integration that need to be studied to guarantee accuracy, speed, and reliability in clinical settings.

Researchers can improve the efficiency, performance, and practicality of TB classification from X-ray images by further investigating these areas and taking advantage of the combination between AI and the most recent developments in FPGA technology. Particularly in areas with limited resources, where fast and accurate tuberculosis identification is of the utmost importance, this future research may lay the groundwork for enhanced healthcare diagnostics.

Reference

[1] World Health Organization, *Global Tuberculosis Report*. 2020.

[2] T. Rahman *et al.*, "Reliable Tuberculosis Detection Using Chest X-Ray With Deep Learning, Segmentation and Visualization," *Ieee Access*, 2020.

[3] S. Suchitra, S. J. A. Ibrahim, M. Sathya, V. Sahini, and N. S. K. Chakravarthy, "Recent Advances in Analysis and Detection of Tuberculosis System In Chest X-Ray Using Artificial Intelligence (AI) Techniques: A Review," *Curr. Mater. Sci.*, 2023.

[4] O. O., "Modelling Outcome of Drug Resistant Tuberculosis and Drug Susceptible Tuberculosis Patients in Oyo State," *Int. J. Public Heal. Pharmacol.*, 2023.

[5] G. C. Ireton, R. Greenwald, H. Liang, J. Esfandiari, K. P. Lyashchenko, and S. G. Reed, "Identification Of <i>Mycobacterium Tuberculosis</I> Antigens of High Serodiagnostic Value," *Clin. Vaccine Immunol.*, 2010.

[6] P. Lakhani and B. Sundaram, "Deep Learning at Chest Radiography: Automated Classification of Pulmonary Tuberculosis by Using Convolutional Neural Networks," *Radiology*, 2017.

[7] V. Acharya *et al.*, "AI-Assisted Tuberculosis Detection and Classification From Chest X-Rays Using a Deep Learning Normalization-Free Network Model," *Comput. Intell. Neurosci.*, 2022.

[8] S. I. Nafisah and G. Muhammad, "Tuberculosis Detection in Chest Radiograph Using Convolutional Neural Network Architecture and Explainable Artificial Intelligence," *Neural Comput. Appl.*, 2022.

[9] S. Heo *et al.*, "Deep Learning Algorithms With Demographic Information Help to Detect Tuberculosis in Chest Radiographs in Annual Workers' Health Examination Data," *Int. J. Environ. Res. Public Health*, 2019.

[10] O. Uwishema, K. S. Frederiksen, I. F. S. Correia, A. Mahmoud, H. Onyeaka, and B. Dost, "The Impact of COVID-19 on Patients With Neurological Disorders and Their Access to Healthcare in Africa: A Review of the Literature," *Brain Behav.*, 2022.

[11] L. Booi and G. D. Breetzke, "The Spatial Relationship Between Tuberculosis and Alcohol Outlets in the Township of Mamelodi, South Africa," *Afr. Health Sci.*, 2022.

[12] R. Saleh *et al.*, "System-on-Chip: Reuse and Integration," *Proc. Ieee*, 2006.

[13] Q. Li, F. Wu, X. Zhang, X. Wang, M. A. Jian-jun, and X. Li, “A Research on Operator Design Method for Health Detection SoC,” *J. Phys. Conf. Ser.*, 2023.

[14] J. Guo, A. Papanikolaou, H. Zhang, and F. Catthoor, “Energy/Area/Delay Tradeoffs in the Physical Design of on-Chip Segmented Bus Architecture,” *Ieee Trans. Very Large Scale Integr. Syst.*, 2007.

[15] Y. Wang *et al.*, “Lithium and Lithium Ion Batteries for Applications in Microelectronic Devices: A Review,” *J. Power Sources*, 2015.

[16] P. Tendulkar, “Mapping and Scheduling on Multi-core Processors using SMT Solvers,” no. October, 2014.

[17] P. Brouwers, “Historical Periodization and the Age of Moore’s Law (1965-),” *Groniek*, 2019.

[18] J. M. Paul and B. H. Meyer, “Amdahl’s Law Revisited for Single Chip Systems,” *Int. J. Parallel Program.*, 2007.

[19] EyermanStijn and EeckhoutLieven, “Modeling Critical Sections in Amdahl’s Law and Its Implications for Multicore Design,” *Acm Sigarch Comput. Archit. News*, 2010.

[20] Z. Cheng, “Design and Optimization of Asynchronous FIFO Based on Verilog HDL,” 2023.

[21] S. Ellinidou, G. Sharma, J.-M. Dricot, and O. Markowitch, “A SDN Solution for System-on-Chip World,” 2018.

[22] A. Ambashanker and P. Kumar, “Modified TACIT Algorithm Based on 4h-Key Distribution for Secure Routing in NoC Architecture,” *Ieice Electron. Express*, 2014.

[23] Z. Wu, Z. Gao, Y. Cao, X. Ye, and J. Yang, “Tolerance Design and Adjustment of Complex Customized Product Based on Cloud Manufacturing,” *Procedia Cirp*, 2015.

[24] K. S. Hanane, L. Abderrazak, R. Adlene, A. Mohamed, and K. Mohamed, “Fuzzy Logic Control of Maximum Power Point Tracking Controller in an Autonomous Hybrid Power Generation System by Extended Kalman Filter for Battery State of Charge Estimation,” *Int. J. Eng.*, 2023.

[25] IEEE Trans. VLSI Syst., “IEEE Transactions on Very Large Scale Integration (VLSI) Systems,” *Ieee Trans. Very Large Scale Integr. Syst.*, 2020.

[26] V. D. Cong, “Industrial Robot Arm Controller Based on Programmable System-on-Chip Device,” *Fme Trans.*, 2021.

[27] M. Wójcik, D. Witek, T. Klej, and E. Ramotowski, “Embedding Components in Voltage Converter PCB for Size Reduction and Heat Management,” *Circuit World*, 2016.

[28] I. Kuon, R. Tessier, and J. Rose, “FPGA Architecture: Survey and Challenges,” 2007.

[29] J. M. Mbongue, “Hardware/Software Infrastructure for Transparent Multi-Tenancy in FPGA-Accelerated Clouds.” University of Florida, 2021.

[30] A. Stratikopoulos, C. Kotselidis, J. Goodacre, and M. Luján, “FastPath: Towards Wire-Speed NVMe SSDs,” 2018.

[31] L. Wang, S. Gong, C. Liu, and M. Song, “A Sparse Pyramid Pooling Strategy,” 2015.

[32] M. Zhao and G. E. Suh, “FPGA-Based Remote Power Side-Channel Attacks,” 2018.

[33] É. Monmasson, L. Idkhajine, M. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar, “FPGAs in Industrial Control Applications,” *Ieee Trans. Ind. Informatics*, 2011.

[34] D. Meng, W. Shi, X. Fang, and B. Zhang, “Design of a novel low temperature intelligent medicine box with Internet control,” *2022 4th Int. Conf. Intell. Control. Meas. Signal Process. ICMSP 2022*, pp. 595–598, 2022.

[35] F. Carrizosa-Corral *et al.*, “FPGA-SoC Implementation of an ICA-based Background Subtraction Method,” *Int. J. Circuit Theory Appl.*, 2018.

[36] F. Baranti, R. Roncella, R. Saletti, and W. Zamboni, “FPGA Implementation of the Mix Algorithm for State-of-Charge Estimation of Lithium-Ion Batteries,” 2014.

[37] J. Caba, M. A. M. Díaz, J. Barba, R. Guerra, and J. A. de la Torre, “FPGA-Based on-Board Hyperspectral Imaging Compression: Benchmarking Performance and Energy Efficiency Against GPU Implementations,” *Remote Sens.*, 2020.

[38] A. Miele, “A Software Framework for Dynamic Self-Repair in Embedded SoCs Exploiting Reconfigurable Devices,” 2010.

[39] N. G. da Silva, A. S. R. Oliveira, R. J. Santos, and L. Almeida, “The OReK Real-Time Micro Kernel for FPGA-based Systems-on-Chip,” 2008.

[40] S. Kalapothas, G. Flamis, and P. Kitsos, “Efficient Edge-Ai Application Deployment for FPGAs,” *Information*, 2022.

[41] V. Boppana, S. Ahmad, I. Ganusov, V. Kathail, V. Rajagopalan, and R. Wittig, “UltraScale+ MPSoC and FPGA Families,” 2015.

[42] F. Ben Abdelaziz, H. Kunze, D. La Torre, and B. Sinclair-Desgagné, “Guest Editorial: Artificial Intelligence and Machine Learning in Business and Management,” *J. Model.*

Manag., 2022.

[43] M. Utmal, “Machine Learning Its Applications, Challenges &Amp; Tools: A Review,” *Int. J. Comput. Sci. Mob. Comput.*, 2021.

[44] P. Belcak and R. Wattenhofer, “Periodic Extrapolative Generalisation in Neural Networks,” 2022.

[45] W. Dai *et al.*, “A Paper-Like Inorganic Thermal Interface Material Composed of Hierarchically Structured Graphene/Silicon Carbide Nanorods,” *ACS Nano*, 2019.

[46] J. Kaur, “Constructive Neural Network: A Framework,” *Int. J. Eng. Adv. Technol.*, 2019.

[47] Y. LeCun, Y. Bengio, and G. E. Hinton, “Deep Learning,” *Nature*, 2015.

[48] S.-S. Zhang, J. Liu, X. Zuo, R. Lu, and S. Lian, “Online Deep Learning Based on Auto-Encoder,” *Appl. Intell.*, 2021.

[49] N. F. F. Zakaria *et al.*, “Tuberculosis Classification Using Deep Learning and FPGA Inferencing,” *J. Adv. Res. Appl. Sci. Eng. Technol.*, 2023.

[50] U. M. D. E. C. D. E. Los, *No 主観的健康感を中心とした在宅高齢者における 健康関連指標に関する共分散構造分析Title*. .

[51] Y. Wang, “Study on Mechanical Automation With X-Ray Power Conveyor Belt Nondestructive Detection System Design,” *Adv. Mater. Res.*, 2013.

[52] C. Miao, B. Shi, P. Wan, and J. Li, “Study on Nondestructive Detection System Based on X-Ray for Wire Ropes Conveyer Belt,” 2007.

[53] G.-D. Kim *et al.*, “A Single FPGA-based Portable Ultrasound Imaging System for Point-of-Care Applications,” *Ieee Trans. Ultrason. Ferroelectr. Freq. Control*, 2012.

[54] C. Youdong, X. Chunxiang, T. Yong, and S. Kai, “FPGA Implementation of Real-Time Ethernet for Motion Control,” *Adv. Mech. Eng.*, 2013.

[55] A. Nosrat and Y. S. Kavian, “Hardware Description of Multi-Directional Fast Sobel Edge Detection Processor by VHDL for Implementing on FPGA,” *Int. J. Comput. Appl.*, 2012.

[56] M. V. G. Rao, P. R. Kumar, and T. Balaji, “A High Performance Dual Stage Face Detection Algorithm Implementation Using FPGA Chip and DSP Processor,” *J. Inf. Syst. Telecommun.*, 2022.

[57] R. Raut, R. H. Jhaveri, R. K. Dhanaraj, and B. Balusamy, “FPGA-Based Deep Learning Models for Analysing Corona Using Chest X-Ray Images,” *Mob. Inf. Syst.*, 2022.

[58] D. S. J. Ting *et al.*, “Artificial Intelligence and Deep Learning in Ophthalmology,” *Br. J.*

Ophthalmol., 2018.

[59] J. Lee, J. He, and K. Wang, “FPGA-based Neural Network Accelerators for Millimeter-Wave Radio-Over-Fiber Systems,” *Opt. Express,* 2020.

[60] Z. Que *et al.*, “Efficient Weight Reuse for Large LSTMs,” 2019.

[61] J. W. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, and K. Vissers, “A Low-Latency Library in FPGA Hardware for High-Frequency Trading (HFT),” 2012.

[62] B. Zhang, H. Zeng, and V. K. Prasanna, “Low-Latency Mini-Batch GNN Inference on CPU-FPGA Heterogeneous Platform,” 2022.

[63] AVNET, “AMD Xilinx Kria KV260 Vision AI Starter Kit,” 2022. [Online]. Available: https://www.avnet.com/wps/portal/us/products/new-product-introductions/npi/xilinx-kria-kv260-vision-ai-starter-kit/. [Accessed: 08-Dec-2023].

[64] V. T. Q. Huy and C. Lin, “An Improved Densenet Deep Neural Network Model for Tuberculosis Detection Using Chest X-Ray Images,” *Ieee Access,* 2023.

[65] K. Manohar *et al.*, “Identifying Drug-Resistant Tuberculosis in Chest Radiographs: Evaluation of CNN Architectures and Training Strategies,” 2021.

[66] I. E. Livieris, A. Kanavos, V. Tampakas, and P. E. Pintelas, “An Ensemble SSL Algorithm for Efficient Chest X-Ray Image Classification,” *J. Imaging,* 2018.

[67] T. Xu and Z. Yuan, “Convolution Neural Network With Coordinate Attention for the Automatic Detection of Pulmonary Tuberculosis Images on Chest X-Rays,” *Ieee Access,* 2022.

[68] S. S. Guia, A. Laouid, M. Kara, and M. Hammoudeh, “Tuberculosis Detection Using Chest X-Ray Image Classification by Deep Learning,” 2023.

[69] L. An *et al.*, “E-TBNet: Light Deep Neural Network for Automatic Detection of Tuberculosis With X-Ray DR Imaging,” *Sensors,* 2022.

[70] M. N. Akbari and A. Azizi, “Building a Convolutional Neural Network Model for Tuberculosis Detection Using Chest X-Ray Images,” *Ghalib Q. J.,* 2023.

[71] Y. Xie *et al.*, “Computer-Aided System for the Detection of Multicategory Pulmonary Tuberculosis in Radiographs,” *J. Healthc. Eng.,* 2020.

[72] M. Liebenlito, Y. Irene, and A.-K. Hamid, “Classification of Tuberculosis and Pneumonia in Human Lung Based on Chest X-Ray Image Using Convolutional Neural Network,” *Inpr. Indones. J. Pure Appl. Math.,* 2020.

[73] Sundaram, “An Adaptive Region Growing Algorithm With Support Vector Machine Classifier for Tuberculosis Cavity Identification,” *Am. J. Appl. Sci.*, 2013.

[74] B. Hooper, “ScholarWorks @ CWU Image Features for Tuberculosis Classification in Digital Chest Radiographs,” 2020.

[75] R. Serrano-Gotarredona *et al.*, “On Real-Time AER 2-D Convolutions Hardware for Neuromorphic Spike-Based Cortical Processing,” 2008.

[76] H. Zhang, Z. Gu, M. Xia, Z. Tang, and G. Hu, “A Reconfigurable System-on-Chip Architecture for Medical Imaging: Preliminary Results,” 2005.

[77] B. Neji, C. Hamrouni, A. M. Alimi, and K. Schilling, “Design and Prototype of an Image Capturing and Processing System for ERPSat-1 Pico Satellite,” 2009.

[78] S. Jung and S.-S. Kim, “Hardware Implementation of a Real-Time Neural Network Controller With a DSP and an FPGA for Nonlinear Systems,” 2007.

[79] AMD, “DPUCZDX8G Architecture,” 2020. [Online]. Available: https://docs.xilinx.com/r/1.2-English/ug1414-vitis-ai/Features%0A. [Accessed: 08-Dec-2023].

[80] S. M. Anwar, M. Majid, A. Qayyum, M. Awais, M. R. Alnowami, and M. K. Khan, “Medical Image Analysis Using Convolutional Neural Networks: A Review,” *J. Med. Syst.*, 2018.

[81] G. Orlando, D. Raimondi, R. Duran-Romaña, Y. Moreau, J. Schymkowitz, and F. Rousseau, “PyUUL Provides an Interface Between Biological Structures and Deep Learning Algorithms,” *Nat. Commun.*, 2022.

[82] X. Fan, D. Wu, W. Cao, W. Luk, and L. Wang, “Stream Processing Dual-Track CGRA for Object Inference,” *Ieee Trans. Very Large Scale Integr. Syst.*, 2018.

[83] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” 2019.

[84] J. Maertzdorf *et al.*, “Concise Gene Signature for Point-of-care Classification of Tuberculosis,” *Embo Mol. Med.*, 2015.

[85] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, and V. Sze, “FastDepth: Fast Monocular Depth Estimation on Embedded Systems,” 2019.

[86] M. Alaei and F. Yazdanpanah, “A High-performance FPGA-based Multicrossbar Prioritized Network-on-chip,” *Concurr. Comput. Pract. Exp.*, 2020.

[87] S. C. Magalhães, F. N. dos Santos, P. Machado, A. P. Moreira, and J. Dias, "Benchmarking edge computing devices for grape bunches and trunks detection using accelerated object detection single shot multibox deep learning models," *Eng. Appl. Artif. Intell.*, vol. 117, no. November 2022, p. 105604, 2023.

[88] W. C. Shiang, A. A. Izzatdin, N. S. Haron, J. Jaafar, N. N. Ismail, and M. Mehat, "The high performance linpack (HPL) benchmark evaluation on UTP high performance cluster computing," *J. Teknol.*, vol. 78, no. 9–3, pp. 21–30, 2016.