UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Science
School of Electronics and Computer Science

# A Suitability Score to Optimize CNNs on an FPGA Accelerator

*by*

**Serkan Saglam**
ORCiD: 0000-0002-7308-8184

***Supervisors***

Prof Mark Zwolinski
Prof Sarvapali Ramchurn
Prof Timothy Underwood

*A thesis for the degree of*
*Doctor of Philosophy*

October 2025

University of Southampton

Abstract

Faculty of Engineering and Physical Science
School of Electronics and Computer Science

Doctor of Philosophy

**A Suitability Score to Optimize CNNs on an FPGA Accelerator**

by Serkan Saglam

This thesis presents a structured optimisation methodology for deploying convolutional neural networks (CNNs) on field-programmable gate arrays (FPGAs), targeting high-throughput operation under constraints of computational resources and latency. The proposed approach integrates model-level restructuring, hardware-aware scheduling, and hardware–software co-design and deployment on FPGAs to deliver high-throughput performance while preserving CNN model accuracy. Oesophageal cancer detection is used as a representative case study, providing a computationally intensive and accuracy-critical scenario for evaluating the proposed methods.

The proposed methodology introduces the Suitability Score, a metric identifying which convolutional layers benefit most from hardware-aware optimisation. This analysis enables selective adjustments that reduce computational cost without sacrificing model accuracy. Based on these insights, a layer-specific pipelining strategy improves the hardware resource efficiency and inference latency of the deployed CNN accelerator. The optimised model is deployed on an FPGA using a co-design framework, demonstrating high throughput and competitive accuracy while consuming fewer hardware resources than FPGA-based CNN accelerators reported in the literature.

The proposed accelerator is deployed on an AMD Kintex UltraScale+ FPGA and evaluated against graphics processing units (GPU)-based inference and existing FPGA implementations. Compared to a GPU baseline, the accelerator achieves at least 47.6% higher throughput and more than twice the energy efficiency. In FPGA-based comparisons, it processes up to 7.8$\times$ more images per second while using fewer hardware resources. Moreover, the results demonstrate that the proposed accelerator achieves a throughput of 76.19 images/s with 97.45% accuracy, while maintaining low resource and power consumption. These results demonstrate that the proposed FPGA-based approach supports real-time CNN inference with high accuracy, high throughput, and efficient hardware usage, making it suitable for broader use in embedded, latency-sensitive image analysis applications.

# Contents

# List of Figures

# List of Tables

# Listings

# Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as:

Signed:..... Date:.....

# Acknowledgements

I sincerely thank my supervisor, Professor Mark Zwolinski, for his guidance, support, and thoughtful feedback throughout my doctoral studies. His expertise and consistent encouragement were valuable in shaping the direction and quality of this research.

Additionally, I would like to gratefully acknowledge the financial support provided by the Turkish Ministry of National Education, Republic of Turkey, which made it possible for me to pursue my doctoral studies. This support was instrumental in allowing me to focus on my research and benefit from a rich academic environment throughout my studies.

I am especially grateful to my dear friends, Jing He and Mahdi Shaban, for their constant support, insightful conversations, and countless moments of shared motivation during this journey. Their friendship made this process not only more manageable but also more meaningful.

I would like to express my heartfelt gratitude to my family, especially my parents, Hulya and Faruk Saglam, for their unwavering support during my doctoral studies and throughout my entire life. Their patience, encouragement, and strength have helped me navigate many challenges, and their belief in me has always been a source of motivation. I am also deeply thankful to my brother, Gokhan Saglam, for always looking out for me and being there whenever I needed support.

Most importantly, I am deeply grateful to my wife, Deniz, whose constant support, patience, and love have sustained me throughout this long and challenging journey. Her steady belief in me has been a continuous source of strength. This thesis came to life thanks to her encouragement and emotional support.

# Definitions and Abbreviations

| | |
|---|---|
| FPGA | Field-Programmable Gate Array |
| WHO | World Health Organization |
| RTL | Register Transfer Level |
| GPU | Graphics Processing Units |
| CPU | Central Process Unit |
| CNN | Convolutional Neural Network |
| HLS | High-Level Synthesis |
| MAC | Multiplication Accumulation |
| MRI | Magnetic Resonance Imaging |
| VLSI | Very Large Scale Integration |
| DMA | Direct Access Memory |
| PYNQ | Python productivity for Zynq |
| PL | Programmable Logic |
| PS | Processing System |
| MM2S | Memory-Mapped to Stream |
| RGB | Red-Green-Blue |
| ASICs | Application-specific integrated circuits |
| LUT | Look-up-Table |
| FF | Flip Flop |
| BRAMs | block RAMs |
| DSP | Digital signal processing |
| CRT | Chemoradiotherapy |
| PE | Processing Element |

# Chapter 1

# Introduction

Recent advancements in machine learning and image processing have profoundly impacted a wide range of fields, notably healthcare [1–4]. In particular, medical imaging has become a cornerstone for the early detection and diagnosis of diseases such as cancer [5, 6]. Early diagnosis is crucial for improving treatment outcomes and survival rates, as evidenced by numerous studies highlighting the importance of timely intervention for various cancer types [7–9]. Consequently, the development of efficient and accurate image processing techniques holds the potential to reduce cancer-related mortality[7, 9].

Convolutional Neural Networks (CNNs) have emerged as an approach in image analysis due to their ability to automatically extract hierarchical feature representations from raw data [10–12]. However, the computational demands associated with CNN inference are substantial, necessitating the use of high-performance computing platforms such as central processing units (CPUs), graphics processing units (GPUs), application-specific integrated circuits (ASICs), and field-programmable gate arrays (FPGAs) [13–15]. While CPUs often struggle with the high degree of parallelism required by deep neural networks [16], GPUs, despite offering excellent parallel throughput, are typically constrained by high power consumption [17]. ASICs provide tailored efficiency but lack flexibility [18]. In contrast, FPGAs offer a compelling balance of reconfigurability, short development cycles, energy efficiency, and high computational throughput, making them particularly attractive for real-time, resource-constrained image processing applications [19–21].

FPGAs consist of a reconfigurable fabric composed of look-up tables (LUTs), digital signal processing (DSP) elements, and on-chip memory blocks such as block RAMs (BRAMs) [22]. These components can be customized to implement highly parallel computational pipelines [23]. This fine-grained parallelism enables the concurrent execution of multiple operations and allows the hardware to be efficiently tailored to

application-specific workloads, which results in superior performance per watt compared to traditional processors [24–26]. The architectural flexibility and parallelism of FPGAs align well with the computational characteristics of CNNs, which rely on extensive multiply–accumulate operations and structured reuse of input data and filter weights [27]. As a result, FPGAs have become a well-suited platform for accelerating machine learning applications, particularly in scenarios where low latency, high throughput, and energy efficiency are key [28, 29].

The demand for efficient CNN acceleration is particularly critical in edge and embedded scenarios, where devices must operate under strict power, memory, and latency constraints [30, 31]. This need is evident in autonomous drones for real-time navigation, portable medical devices delivering rapid diagnostics, and surveillance systems processing continuous high-resolution video streams without cloud support [32, 33]. Deploying CNNs on FPGAs remains challenging due to the need to coordinate highly parallel computations while efficiently mapping them onto heterogeneous hardware resources such as LUTs, DSP blocks, and on-chip memory [34–36]. These constraints necessitate careful hardware–software co-design to fully exploit the FPGA's potential while avoiding excessive consumption of its limited resources [37, 38]. This underscores the critical need for FPGA-based CNN acceleration strategies to bridge the gap between state-of-the-art deep learning models and their deployment in real-world, resource-constrained applications.

Addressing the computational challenges posed by modern CNN-based image processing [39–41], this thesis proposes an FPGA-based deployment strategy, using oesophageal cancer detection as a representative case study. Although the case study centres around cancer detection, the primary objective is to develop a real-time FPGA-based CNN acceleration approach for RGB-domain image processing tasks. Oesophageal cancer is selected due to its clinical importance as a highly lethal disease where early diagnosis improves patient outcomes [42]. The availability of full-resolution RGB endoscopic datasets and the widespread adoption of CNNs in medical imaging further motivated this choice. Nevertheless, the methods and architectural strategies proposed herein are adaptable to a broader class of image processing applications requiring efficient, low-latency inference beyond healthcare.

In summary, this thesis proposes a systematic approach for optimising CNN architectures and their deployment on FPGA hardware, aiming to enable scalable, accurate, and resource-efficient real-time image processing systems. By combining structured model optimisations with hardware-aware deployment strategies, the proposed approach contributes towards bridging the gap between algorithmic advancements in deep learning and practical, power-efficient real-world applications.

## 1.1 Aim and Objectives

The aim of this thesis is to develop an FPGA-based acceleration methodology for CNNs that enables real-time, resource-efficient inference while maintaining model accuracy. To achieve this aim, the thesis pursues several objectives, each of which addresses challenges in deploying CNNs on FPGA hardware.

- Chapter 3 focuses on optimising CNN structures to reduce computational cost and improve FPGA deployment efficiency without sacrificing accuracy. To achieve this, it introduces the *Suitability Score*, a metric that evaluates each convolutional layer's compatibility with Winograd transformations. Based on this score, selective modifications are applied to kernel size and output channel count, enabling targeted reductions in complexity while preserving inference accuracy. This hardware-conscious optimisation forms the foundation for the subsequent acceleration strategies developed in later chapters.

- Chapter 4 introduces an adaptive pipelining strategy to improve hardware-aware parallelism in FPGA-based CNN accelerators. It dynamically assigns pipeline initiation intervals according to each layer's computational load and structural complexity. The objective is to reduce idle cycles and balance FPGA resource utilization while sustaining throughput. In this way, computationally intensive layers achieve lower latency through deeper pipelining, whereas less demanding layers conserve hardware by avoiding unnecessary parallel computation. This chapter demonstrates a measurable balance between latency reduction and resource efficiency, forming the basis for the system-level deployment in Chapter 5.

- Chapter 5 aims to demonstrate the practical effectiveness of deploying the optimized CNN accelerator on FPGA hardware, thereby validating the architectural and optimization strategies developed in the preceding chapters. To this end, oesophageal cancer detection is adopted as a representative use case due to its stringent accuracy, latency, and resource requirements. The chapter presents a complete FPGA implementation that enables high-throughput, low-latency inference on RGB endoscopic images. The design follows a hardware–software co-design methodology, integrating key components such as PCIe-based host–device communication, external DDR memory management, and fixed-point quantization to balance computational efficiency with predictive accuracy. Furthermore, the chapter provides a comparative evaluation against GPU-based inference and prior FPGA implementations, focusing on throughput, latency, resource utilization, and deployment suitability in demanding real-world image analysis scenarios.

## 1.2 Academic Contributions

- Saglam, S., Zwolinski, M., (2025, March). Using a Suitability Score to Optimize CNN Implementation on FPGAs. In 2025 IEEE Symposium Series on Computational Intelligence (SSCI) (poster).
- A research article based on this thesis has been submitted and is currently under review for publication in the *IEEE Access*.

# Chapter 2

# Background

## 2.1 Introduction

As deep learning continues to revolutionise computer vision and image analysis, CNNs have emerged as indispensable tools across a wide range of domains [43–45]. The increasing complexity and depth of modern CNN architectures, driven by the pursuit of higher accuracy, have led to a significant surge in their computational demands [46]. This escalating computational burden poses a considerable challenge for real-time deployment, particularly in resource-constrained environments such as embedded systems and edge devices [29]. The intensive multiply-accumulate (MAC) operations inherent in CNNs necessitate substantial processing power, often exceeding the capabilities of CPUs to meet the latency requirements of many applications [29, 46].

Hardware acceleration has emerged as a critical approach to addressing the computational bottlenecks associated with CNNs and achieving the desired performance and efficiency [25, 47, 48]. Various hardware platforms, including GPUs, ASICs, and FPGAs, have been explored for their potential to accelerate CNN workloads [44, 49]. While GPUs offer high throughput for parallel computations, their power consumption can be prohibitive for edge deployments. ASICs can provide the highest performance and energy efficiency but lack the flexibility to adapt to evolving CNN architectures [21].

FPGAs have attracted considerable attention as a promising hardware platform for accelerating CNNs, primarily due to their distinctive combination of high parallelism, reconfigurability, and energy efficiency [43, 50]. FPGAs enable the implementation of custom hardware architectures that can be precisely tailored to the specific computational demands of CNNs, facilitating fine-grained parallelism and deep pipelining [16]. Their inherent reconfigurability provides the crucial flexibility to adapt to the diverse and rapidly changing CNN models and optimisation techniques, rendering them highly suitable for the dynamic field of deep learning [51, 52]. Furthermore, FPGAs

often demonstrate a superior performance-per-watt ratio compared to GPUs for many CNN inference tasks, making them particularly appealing for applications with stringent power constraints [50, 53]. Since convolution operations are the most computationally intensive part of CNNs, optimising these operations on FPGAs is crucial for maximising performance and efficiency on this platform [25].

Convolution operations account for the majority of computational workload in CNNs, and consequently, optimising their execution is paramount for achieving high overall performance [54]. The Winograd algorithm stands out as a fast convolution technique that effectively reduces the number of multiplications required for computing convolutions, especially for the small kernel sizes that are prevalent in contemporary CNN architectures [46]. By transforming the convolution operation into a sequence of matrix multiplications within a transformed domain, the Winograd algorithm achieves a reduction in arithmetic complexity, specifically in the number of multiplications, albeit at the expense of an increased number of additions [46]. This decrease in multiplications can lead to substantial performance gains and a reduction in resource utilisation, particularly in FPGA implementations where Digital Signal Processors (DSPs) are a finite resource.

Although the methods discussed in this thesis are intended to support CNN applications, cancer detection is referenced as a motivating scenario due to its computational demands. These include the need for real-time inference, high-resolution image processing, and hardware-efficient deployment [55]. Additionally, the availability of publicly accessible and well-annotated datasets makes it a practical context for evaluation [56].

Despite extensive research on optimising CNN inference for FPGAs, challenges persist in systematically aligning model structures with hardware-specific acceleration techniques such as Winograd convolution. Moreover, achieving a balanced trade-off between computational efficiency, resource utilisation, and inference accuracy remains an open research problem. To address these limitations, this thesis presents a cohesive optimisation framework that systematically unifies model-level restructuring and hardware-aware deployment techniques, with the objective of enabling efficient, scalable, and high-accuracy CNN inference on FPGA platforms.

## 2.2 FPGA-based Acceleration of Convolutional Neural Networks

This section provides a foundational overview of employing FPGAs for the acceleration of CNN inference. It begins by delineating the inherent advantages FPGAs offer for these computationally intensive workloads, followed by examining the significant challenges encountered during implementation. Subsequently, prominent architectural

paradigms developed to address these challenges and maximise performance are discussed. Finally, the key metrics used to evaluate and compare the efficacy of FPGA-based CNN accelerators are defined and contextualised.

### 2.2.1 Advantages of FPGAs for CNN Acceleration

FPGAs present a compelling alternative to traditional processing platforms like CPUs and GPUs for accelerating CNNs, particularly for inference tasks. Their unique architectural characteristics, including inherent parallelism, reconfigurability, and potential for high energy efficiency, offer several distinct advantages for mapping the computationally intensive operations found in CNNs [25, 44]. These advantages make FPGAs particularly well-suited for accelerating the inference phase of CNNs, which involves deploying trained models for tasks like image classification or object detection.

#### 2.2.1.1 Parallelism and Pipelining Capabilities

A principal advantage of FPGAs lies in their capacity for massive parallelism and deep pipelining, which can be customised to match the computational structure of CNNs. The underlying fabric of configurable logic blocks (CLBs), interconnects, and dedicated resources like DSP blocks allows designers to instantiate numerous custom processing units (PUs) or processing elements (PEs) that execute concurrently [23]. This hardware-level parallelism can be tailored to exploit the data-level parallelism inherent in CNNs, such as performing convolutions across different input or output channels or spatial dimensions of feature maps.

Additionally, FPGAs support fine-grained pipelining, allowing complex CNN operations to be decomposed into smaller stages that operate in a streaming fashion. Data flows directly between these pipeline stages using dedicated on-chip routing resources, reducing the need for shared memory and lowering control overhead. This dataflow-oriented execution aligns well with CNN layer sequencing, where the output of one layer serves as input to the next. Systolic arrays, a regular grid of locally connected PEs, are particularly effective for CNN convolution and matrix multiplication, providing high throughput and efficient hardware utilisation [57].

The synergy between FPGA architecture and CNN workloads is most evident in implementing convolutional layers, which involve repetitive MAC operations over large feature maps. FPGAs can instantiate parallel MAC units using resource-efficient DSP blocks arranged in spatial or systolic configurations [21]. Their reconfigurability further allows designers to tune the hardware to the specific parallelism profile of each network—whether across channels, filters, or spatial dimensions. Combined with pipelined

execution across CNN layers or sub-operations, this flexibility enables high-throughput, low-latency inference, particularly in power- and resource-constrained scenarios [58].

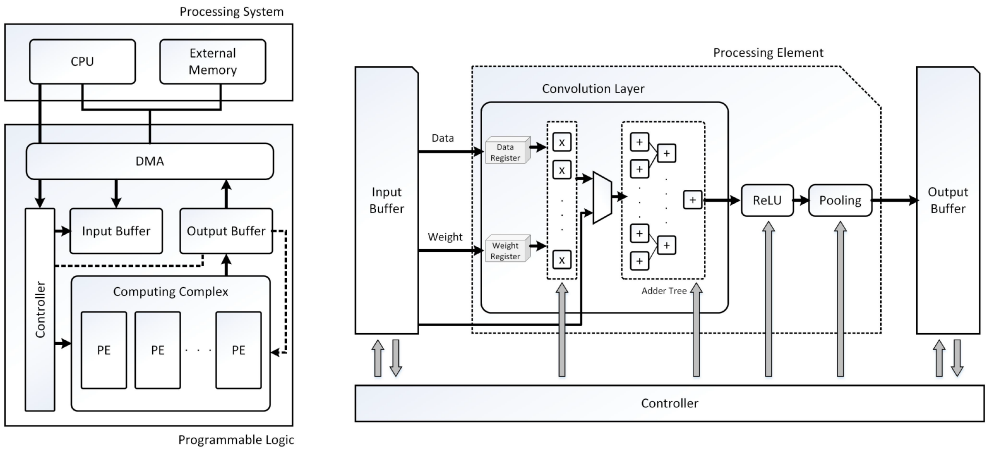



FIGURE 2.1: FPGA-based CNN accelerator architecture [59]

Figure 2.1 illustrates the architecture of a typical FPGA-based CNN accelerator, highlighting both parallelism and pipelined execution. On the left, the system-level design showcases a parallel array of PEs instantiated within the programmable logic. These PEs execute concurrently, enabling convolution operations across multiple channels or feature maps.

Data is transferred via direct memory access (DMA) into input buffers, processed by the PE array, and results are stored in output buffers. On the right, a zoomed-in view of a single PE reveals pipelined stages, including data and weight registers, MAC computation via an adder tree, activation (ReLU), and optional pooling. This architecture demonstrates how spatial parallelism and sequential pipelining effectively combine to accelerate CNN inference on FPGAs [60]. This foundational parallelism capability sets the stage for customising FPGA architectures, which is further explored in the next section on reconfigurability and architectural flexibility [60].

#### 2.2.1.2 Reconfigurability and Flexibility for Custom Architectures

A key advantage of FPGAs in CNN acceleration is their reconfigurability, which enables hardware designs to be updated or modified after deployment. Unlike fixed architectures such as CPUs, GPUs, or ASICs, the logic fabric and interconnects in FPGAs can be reprogrammed to support evolving neural network models, algorithmic updates, or new optimisation techniques—without requiring hardware redesign or fabrication [61]. This flexibility is particularly beneficial in deep learning, where models and methods change frequently.

Reconfigurability also allows designers to build custom accelerator architectures tailored to the computational structure of specific CNNs or even individual layers. This

includes optimising PEs, data paths, and memory hierarchies—capabilities not available in fixed-hardware platforms [23]. Beyond flexibility, another hallmark of FPGA-based CNN acceleration is their ability to deliver low-latency responses and high sustained throughput, which is especially critical for real-time applications.

#### 2.2.1.3 Potential for Low Latency and High Throughput

FPGAs offer unique architectural advantages—such as massive parallelism, deep pipelining, and efficient on-chip data movement—that enable both low inference latency and high throughput in CNN accelerators [62]. This dual capability makes FPGAs particularly suitable for real-time applications requiring rapid and continuous processing of input samples, such as autonomous vehicles, robotic control, and live video analytics.

Unlike GPUs, which typically optimise for high throughput via large batch sizes, FPGAs can be tailored for single-sample inference, minimising the time required to process each input. In pipelined hardware architectures, latency is often modelled as:

$$\text{Latency} = L + (N - 1) \times II \tag{2.1}$$

where $L$ is the pipeline depth, $N$ is the batch size, and $II$ is the initiation interval. In many real-time scenarios where $N$ is small (even equal to 1), the latency is dominated by $L$, and reducing $II$ directly improves throughput.

Indeed, throughput is inversely proportional to the initiation interval, i.e., Throughput $\propto \frac{1}{II}$, where $II$ denotes the initiation interval. In the context of CNNs, throughput is also quantified in terms of MACs, expressed as Giga Operations Per Second (GOPS). This can be computed as:

$$\text{Throughput (GOPS)} = \frac{\text{MACs per input} \times \text{FPS}}{10^9} \tag{2.2}$$

where FPS is the number of inferences per second. Maximising MAC utilisation and reducing $II$ enables FPGAs to achieve high throughput even at low batch sizes.

#### 2.2.1.4 Energy Efficiency Compared to GPUs and CPUs

FPGAs are often preferred over GPUs and CPUs for CNN inference in edge and power-constrained scenarios due to their superior energy efficiency. This is typically measured in operations per watt, such as GOPS/W or TOPS/W. While GPUs can achieve high peak throughput, FPGAs can offer competitive performance with lower power consumption by eliminating general-purpose processing overhead [63].

This efficiency is enabled by several architectural features. First, FPGAs allow custom logic tailored to the network, activating only necessary components. Second, their abundant on-chip memory (e.g., BRAMs) enables efficient data reuse and reduces costly off-chip DRAM accesses, which dominate energy use in CNN accelerators. Third, FPGAs support low-precision arithmetic—such as INT8, INT4, or binary formats—which drastically reduce both computation and data transfer energy.

For example, using INT8 instead of FP32 can yield up to 4× energy savings in MAC operations alone [25]. Some studies report FPGA-based accelerators achieving up to 10–40× better energy efficiency than GPUs, depending on the network size and optimisation level.

$$\text{Energy Efficiency} = \frac{\text{Throughput (OPS)}}{\text{Power (W)}} \tag{2.3}$$

Compared to ASICs, FPGAs offer slightly lower energy efficiency, but gain the advantage of reconfigurability and lower development cost. However, their energy efficiency is highly context-dependent, influenced by model type, hardware generation, and optimisation effort. In some cases, the faster development workflows of GPU ecosystems may outweigh FPGA efficiency gains [21]. Although FPGAs offer many advantages for accelerating CNN inference, their practical implementation still involves several challenges related to limited hardware resources and hardware design complexity.

### 2.2.2 Comparative Analysis of CNN Deployments on CPU, GPU, and FPGA Platforms

Table 2.1 provides a comparative evaluation of CNN model deployments across CPU, GPU, and FPGA platforms, focusing on key metrics such as latency, throughput, power consumption, and energy efficiency. Several critical observations emerge from the analysis:

- **Throughput:** FPGAs consistently outperform CPUs in throughput across multiple models. For instance, the Xilinx VC709 FPGA achieves 424.7 GOPS on LeNet-5, offering approximately **14.8**× higher throughput compared to the CPU baseline. Although GPUs provide high throughput, FPGAs often achieve comparable or superior throughput with substantially lower power consumption.
- **Latency:** For deeper networks such as ResNet-18, FPGAs demonstrate lower latency. Specifically, ResNet-18 achieves a latency of **1.5 ms** on FPGA versus **4.0 ms** on GPU, a substantial reduction critical for real-time applications.

- **Power Consumption:** FPGAs exhibit markedly lower power consumption relative to GPUs. For MobileNet-v2, the ZCU102 FPGA consumes **10.7 W** compared to **11.1 W** for the NVIDIA Jetson TX2 GPU, despite achieving higher throughput.
- **Energy Efficiency:** FPGAs consistently demonstrate superior energy efficiency. In the case of MobileNet-v2, the FPGA achieves **83.4 FPS/W**, compared to **27.6 FPS/W** for the GPU, indicating that the FPGA is approximately **three times more energy-efficient**.
- **Efficiency Gains Relative to Baseline:** The *Comparison* column shows that FPGAs deliver considerable improvements relative to both CPU and GPU baselines, particularly in terms of energy-delay product (EDP), thereby achieving a balanced trade-off between speed and energy usage.

Based on these observations, FPGAs offer several decisive advantages over CPUs and GPUs:

- Substantially higher throughput compared to CPUs.
- Lower latency compared to GPUs, essential for real-time inference.
- Reduced power consumption.
- Better throughput-per-watt performance.
- Reconfigurability and adaptability to evolving CNN architectures.

Therefore, for applications requiring efficient, real-time, and low-power CNN inference—such as autonomous driving, medical imaging, and edge computing—FPGAs represent a highly suitable deployment platform.

TABLE 2.1: Comparative Performance of CNN Models on FPGA, CPU, and GPU Platforms

| Model | Platform | Hardware | Prec. | Lat. | Throughput | Power (W) | Eff. | Comparison† | Study |
|---|---|---|---|---|---|---|---|---|---|
| LeNet-5 | CPU | Intel Core (unspec.) | FP32 | – | 28.61 GOPS | – | – | Baseline (for throughput) | [64] |
| | FPGA | Xilinx VC709 | FP32 | – | 424.7 GOPS | – | – | 14.8× throughput vs CPU | |
| | CPU | Intel Core i5-4590 | Float | – | 3.831 GFLOPS | 32.15 | 0.119 GFLOPS/W | Baseline (for efficiency) | [65] |
| | GPU | NVIDIA GTX 1080ti | Float | – | 27.14 GFLOPS | 52 | 0.522 GFLOPS/W | 4.38× vs CPU | |
| | FPGA | GENESYS2 (Artix-7) | Fixed-pt | – | 19.61 GFLOPS | 4.15 | 4.72 GFLOPS/W | 39.66× vs CPU, 9.06× vs GPU | |
| AlexNet | CPU | Intel Core (unspec.) | FP32 | – | 64.05 GOPS | – | – | Baseline (for throughput) | [64] |
| | FPGA | Xilinx VC709 | FP32 | – | 445.6 GOPS | – | – | 6.96× vs CPU | |
| VGG-S | CPU | Intel Core (unspec.) | FP32 | – | 98.85 GOPS | – | – | Baseline (for throughput) | [64] |
| | FPGA | Xilinx VC709 | FP32 | – | 473.4 GOPS | – | – | 4.79× throughput vs CPU | |
| ResNet-18 | GPU | NVIDIA Jetson TX2 | FP16 | 4.0 ms | 253 FPS | 12.2 W | 20.7 FPS/W | Baseline (for efficiency) | [66] |
| | FPGA | Xilinx ZCU102 | INT8 | 1.5 ms | 674 FPS | 11.1 W | 60.7 FPS/W | 2.6× faster, ↓1.4× FPGA vs GPU | |
| MobileNet-v2 | GPU | NVIDIA Jetson TX2 | FP16 | 3.3 ms | 306 FPS | 11.1 W | 27.6 FPS/W | Baseline (for efficiency) | [66] |
| | FPGA | Xilinx ZCU102 | INT8 | 1.1 ms | 892 FPS | 10.7 W | 83.4 FPS/W | 2.9× faster, ↓2.4× EDP vs GPU | |
| SqueezeNet | GPU | NVIDIA Jetson TX2 | FP16 | 3.1 ms | 323 FPS | 11.0 W | 29.4 FPS/W | Baseline (for efficiency) | [66] |
| | FPGA | Xilinx ZCU102 | INT8 | 1.2 ms | 806 FPS | 10.7 W | 75.3 FPS/W | 2.5× faster, ↓1.7× EDP vs GPU | |
| Inception-v2 | GPU | NVIDIA Jetson TX2 | FP16 | 6.1 ms | 164 FPS | 11.4 W | 14.4 FPS/W | Baseline (for efficiency) | [66] |
| | FPGA | Xilinx ZCU102 | INT8 | 2.4 ms | 413 FPS | 10.9 W | 37.9 FPS/W | 2.5× faster, ↓1.5× EDP vs GPU | |
| ResNet-50 | GPU | NVIDIA Jetson TX2 | FP16 | 9.1 ms | 110 FPS | 11.4 W | 9.6 FPS/W | Baseline (for efficiency) | [66] |
| | FPGA | Xilinx ZCU102 | INT8 | 4.3 ms | 235 FPS | 11.3 W | 20.8 FPS/W | 2.1× faster, ↓1.1× EDP vs GPU | |
| ResNet-50 (Sparse) | CPU | Intel SkyLake Gold 6130 | FP32 | – | – | ∼230 W | – | Baseline (for eff. + spd.) | [23] |
| | FPGA | Intel Stratix 10 MX | FP16 | – | – | ∼70 W | – | 3.3× more efficient vs CPU; 2× vs V100 | |
| | GPU | NVIDIA V100 | FP32 | – | – | ∼140 W | – | 1.3× faster vs FPGA | |
| MobileNet (Sparse) | CPU | Intel SkyLake Gold 6130 | FP32 | – | – | ∼245 W | – | Baseline | [23] |
| | FPGA | Intel Stratix 10 MX | FP16 | – | – | ∼70 W | – | 3.5× more efficient vs CPU; 1.7× vs V100 | |
| | GPU | NVIDIA V100 | FP32 | – | – | ∼120 W | – | 1.2× throughput vs FPGA | |

**Note†**: Lat. = latency, Thrpt. = throughput, Pwr. = power, Eff. = energy efficiency, imp. = implied, pt = point, unspec. = unspecified

### 2.2.3 Challenges in Implementing CNNs on FPGAs

Despite the compelling advantages, the practical implementation of high-performance CNN accelerators on FPGAs presents considerable challenges. These stem primarily from the finite nature of FPGA resources, the complexities of hardware design, and the demanding computational and memory requirements of modern deep learning models.

#### 2.2.3.1 Limited On-Chip Memory Resources

A key limitation of FPGAs in CNN acceleration is the limited capacity of on-chip memory, typically implemented using Block RAMs (BRAMs) or UltraRAMs (URAMs) [67]. Although modern FPGAs provide tens of megabits of on-chip storage, this is often insufficient for the full set of CNN weights, biases, and intermediate activations, especially in large models. For instance, AlexNet requires around 250MB just to store its parameters—far exceeding typical on-chip capacities. Deeper networks like VGG or ResNet further exacerbate this challenge [25].

As a result, designers must often rely on off-chip memory (e.g., DRAM) to store parts of the model and intermediate results. However, accessing off-chip memory introduces additional latency and power overheads, potentially negating the efficiency advantages of FPGA acceleration [68]. This disparity is commonly referred to as the memory wall.

Efficient usage of on-chip BRAMs is itself a non-trivial task. These memories are typically fixed in size and port configuration, which complicates the mapping of irregularly shaped CNN buffers. Fragmentation and underutilisation are common, meaning the effective usable capacity may be lower than the raw specification.

To address these constraints, various optimisation techniques are used. These include quantisation (using fewer bits per value), pruning (removing redundant parameters), and advanced buffering/caching strategies (e.g., ping-pong buffers or line buffers) [67]. Dataflow-based architectures, especially fused-layer approaches, are also employed to reduce intermediate storage demands and off-chip memory accesses.

#### 2.2.3.2 Design Complexity and Optimisation Efforts in FPGA-based CNN Acceleration

Developing efficient CNN accelerators on FPGAs inherently involves complexity and optimisation challenges compared to CPU or GPU implementations. FPGA design traditionally employs hardware description languages (HDLs) like Verilog or VHDL, demanding detailed register-transfer level (RTL) coding, manual resource optimisation,

and rigorous verification. Although HDL provides unparalleled control and performance potential, it increases development complexity and requires extensive hardware expertise.

Alternative methods, particularly framework-based tools, have gained popularity due to their ability to rapidly convert CNN models developed in popular software libraries (e.g., TensorFlow, PyTorch, Keras) into FPGA-optimised implementations [69]. These frameworks leverage pre-built libraries and abstraction layers, simplifying the FPGA deployment process. However, while facilitating ease of use and rapid deployment, they inherently introduce constraints on customisation and optimisation capabilities. Prominent examples include:

- **Vitis AI (DPU-based):** Utilises predefined Deep Processing Units, providing rapid deployment for standard CNN architectures. However, it substantially restricts layer-level customisation, limiting detailed optimisations needed for highly specialised or optimised CNN architectures [70].
- **hls4ml:** Designed primarily for small-scale, latency-critical applications and integrates seamlessly with high-level frameworks like Keras. Nonetheless, its support for complex CNN architectures and detailed layer-level optimisations is limited, restricting its effectiveness for larger models [71].
- **FINN:** Specialises in Binarised Neural Networks (BNNs), offering high resource efficiency and low latency through aggressive quantisation and streamlined dataflow architectures. Its applicability, however, is limited strictly to binarised or heavily quantised models, reducing its suitability for general-purpose CNN implementations [72].

In contrast, High-Level Synthesis (HLS) tools such as Xilinx Vitis HLS address these complexities by enabling hardware design through higher-level languages like C/C++ or OpenCL. HLS shortens development cycles, lowers the barrier to entry for hardware design, and supports rapid prototyping and iterative exploration of design spaces [73]. Importantly, HLS tools allow designers to exercise granular control through pragmas, facilitating optimised hardware implementation via loop unrolling, pipelining, and dataflow techniques [69]. This balance of abstraction and detailed customisation enables efficient performance tuning without the extensive manual coding efforts required by HDL, offering greater flexibility than framework-specific solutions.

Compared to the aforementioned frameworks, Vitis HLS provides an effective combination of flexibility and performance efficiency. Unlike the predefined architectures offered by DPU-based solutions, Vitis HLS supports extensive layer-level optimisations, including fine-grained control over memory access patterns, parallelism, and resource allocation. Unlike specialised tools such as hls4ml or FINN, HLS is not constrained by

specific quantisation or architectural requirements, thus supporting a wider array of CNN architectures and optimisation strategies.

However, this flexibility comes at the cost of increased design effort and longer development cycles. When models are initially trained using high-level Python-based frameworks such as TensorFlow or PyTorch, translating these models into C/C++ for use with HLS requires additional effort, including interface adaptation and data preprocessing. This conversion step can be particularly time-consuming and may introduce discrepancies if not carefully managed [71]. As such, while Vitis HLS offers significant potential for customisation and performance, it also necessitates a deeper involvement in low-level design decisions compared to automated, framework-based alternatives.

In conclusion, while HDL provides maximal optimisation capability, its inherent complexity renders it less practical for rapid development and iterative refinement. Framework-based tools simplify the implementation process but severely restrict customisation opportunities. Conversely, HLS tools like Vitis HLS offer an optimal balance, facilitating detailed layer-level optimisations alongside faster, more accessible development processes, thereby presenting a compelling solution for efficient FPGA-based CNN acceleration. Apart from the challenges posed by hardware and development tools, the need to balance limited resources with performance goals makes the design of CNN accelerators even more complex. Apart from the challenges posed by hardware and development tools, the need to balance limited resources with performance goals makes the design of CNN accelerators even more complex.

#### 2.2.3.3 Balancing Resource Utilisation and Performance

FPGA-based CNN accelerator design requires carefully balancing computational performance with the limited hardware resources available on the device. These resources include logic elements (LUTs, flip-flops), digital signal processing (DSP) blocks—primarily used for MAC operations—and on-chip memory (BRAMs or URAMs). Maximising throughput often entails replicating processing elements (PEs) and unrolling loops, increasing resource usage and power consumption [74]. Similarly, larger on-chip buffers that reduce off-chip DRAM access demand more memory blocks.

However, excessive parallelism can lead to routing congestion, increased power usage, and timing closure challenges, potentially limiting achievable clock frequency. Moreover, memory bandwidth can become a bottleneck, constraining the benefits of added compute parallelism. These limitations and trade-offs are evident across various CNN implementations on different hardware platforms, as summarised in Table 2.1

Another key trade-off involves accuracy versus efficiency. Aggressive quantisation and pruning can substantially reduce hardware demands and power usage but may also

degrade model accuracy. Maintaining an acceptable balance requires iterative tuning and verification. Since FPGAs typically operate at lower clock frequencies (e.g., 100–300 MHz), high throughput must be achieved primarily through spatial parallelism [63]. Therefore, careful resource allocation becomes essential to achieve optimal performance within platform constraints.

Designing an efficient CNN accelerator on FPGA ultimately involves navigating a complex multi-objective optimisation space. Dataflow strategy, quantisation level, memory hierarchy, and the specific CNN architecture must be co-optimised to meet application-specific performance goals. This highlights the need for systematic design space exploration to effectively balance key metrics such as throughput, latency, power, and area within the constraints of available FPGA resources.

### 2.2.4 Comparative Perspective and Platform Suitability

Section 2.2.1 and 2.2.3 highlight that while FPGAS offer compelling advantages for accelerating CNN inference, such as high energy efficiency, low latency, and architectural customisability, they also introduce substantial design and implementation complexity. Key challenges include limited on-chip memory capacity, increased development time, and a more demanding development process than CPUs and GPUs.

To contextualise these trade-offs, Table 2.2 compares FPGA-based acceleration with CPU and GPU implementations across key criteria, including throughput, latency, power consumption, energy efficiency, and development complexity.

TABLE 2.2: Qualitative Comparison of CNN Deployment Platforms [75]

| **Metric** | **CPU** | **GPU** | **FPGA** |
|---|---|---|---|
| Throughput | Lowest | Highest | High |
| Latency | Highest | Medium | Lowest |
| Power | Medium | Highest | Lowest |
| Energy Efficiency | Worst | Medium | Best |
| Device Size | Small | Large | Small |
| Development Effort | Easiest | Easy | Hard |
| Library Support | Sufficient | Sufficient | Limited |
| Flexibility | Limited | Limited | Flexible |

The table shows that GPUs typically offer the highest raw performance in throughput, making them suitable for high-throughput data centre applications. While it is easy to program and deploy, CPUs perform poorly in both latency and energy efficiency, making them unsuitable for real-time or power-sensitive tasks.

FPGAs, by contrast, strike a compelling balance—they provide better energy efficiency and latency than GPUs while offering customizability for domain-specific workloads. However, this comes at the cost of increased design complexity and longer development cycles.

Therefore, the choice of processing platform should be guided by application-specific constraints. For instance, real-time edge applications that demand low power and latency may favour FPGAs. In contrast, data-intensive training or inference tasks with high throughput requirements and less stringent power budgets may benefit more from GPUs. CPUs are useful for general-purpose control and integration tasks but less suitable as standalone CNN accelerators.

In summary, FPGAs represent a powerful yet complex alternative for deploying CNNs, particularly in environments where energy efficiency, low latency, and hardware-level flexibility are paramount. Their suitability is maximised when energy efficiency, latency, and architectural control outweigh the need for rapid development or broad framework support. Understanding these trade-offs is critical for making informed platform selection decisions in modern deep-learning pipelines.

## 2.3 Algorithmic Strategies for Convolutional Layer Optimisation on FPGAs

Convolutional layers are the most computationally demanding components of CNNs, typically dominating processing time and hardware resource usage during inference. Their optimisation is critical for efficient CNN deployment on hardware platforms, particularly FPGAs [54]. This section investigates algorithmic strategies for accelerating convolution operations, focusing on FPGA implementation. It provides a comparative overview of existing approaches—including direct convolution, im2col with matrix multiplication, FFT-based methods, and the Winograd algorithm—highlighting their computational characteristics and hardware implications. Particular attention is given to the Winograd minimal filtering algorithm, which offers significant arithmetic reduction, especially in MACs. The review explores Winograd's potential and discusses its practical limitations and implementation challenges specific to FPGA architectures.

### 2.3.1 Computational Landscape of Convolutional Neural Networks

A thorough understanding of the computational demands of CNNs is essential for designing efficient hardware accelerators. CNNs comprise multiple layer types, contributing differently to the overall inference workload regarding computational intensity and memory access patterns. Below is an overview of the primary layer types

commonly found in CNNs, highlighting their respective roles and relative computational burdens:

- **Convolutional (CONV) Layers:** These foundational layers are responsible for feature extraction. They operate by sliding learnable filters (kernels) across the input volume. Each position computes a dot product between filter weights and the receptive field, involving a large number of MAC operations [41].
- **Activation Layers (e.g., ReLU):** Introduce non-linearity after convolutional or fully connected layers, enabling the network to model complex relationships. ReLU is widely used due to its simplicity and effectiveness in avoiding vanishing gradients. These are computationally less intensive than convolutional layers [25].
- **Pooling Layers (e.g., Max, Average):** Perform down-sampling by reducing the spatial dimensions. Pooling helps maintain spatial invariance and reduces the computational load of subsequent layers. These operations are also less demanding than convolutions [25].
- **Fully Connected (FC) Layers:** Connect every neuron in one layer to every neuron in the next, typically for classification. Despite their high parameter count, FC layers contribute less to the overall computational load than convolutional layers in modern CNNs [25].

### 2.3.2 Workload Analysis of CNN Layers

Convolutional layers are key parts of CNN models that extract essential features from input data through structured computations. Since convolutional layers demand numerous arithmetic operations [40], analysing convolutional layers in detail is necessary when designing efficient CNN architectures. The significance of a convolutional layer can differ based on the number of output channels and its role in feature extraction. Consequently, high-importance layers should be carefully assessed, as they considerably affect the model's accuracy and the computational resources needed for training and inference. This section aims to demonstrate why convolutional layers dominate the computational workload in CNNs and to motivate the need for their targeted optimisation, particularly on hardware platforms such as FPGAs.

MAC operations measure the computational complexity of convolutional layers in CNNs [39]. These operations measure the total number of multiplications and additions needed to produce the output feature map. Analysing and optimising MACs is crucial to improving the computational efficiency of CNNs. The parameter definitions required for MAC computation are in Table 2.3.

| Symbol | Definition |
|---|---|
| $C_{\text{in}}$ | Number of input channels |
| $C_{\text{out}}$ | Number of output channels |
| $K_H$ | Height of the convolutional filter |
| $K_W$ | Width of the convolutional filter |
| $I_H$ | Height of the input matrix |
| $I_W$ | Width of the input matrix |
| $P$ | Amount of padding applied to the input matrix |
| Stride | Step size of the convolution operation |

TABLE 2.3: Parameter definitions for $MAC_{\text{conv}}$ calculation [76]

$$O_H = \left\lfloor \frac{I_H - K_H + 2P}{Stride} \right\rfloor + 1 \tag{2.4}$$

$$O_W = \left\lfloor \frac{I_W - K_W + 2P}{Stride} \right\rfloor + 1 \tag{2.5}$$

$$MAC_{\text{conv}} = C_{\text{in}} \times C_{\text{out}} \times K_H \times K_W \times O_H \times O_W \tag{2.6}$$

For a standard convolution operation, the number of MACs can be calculated by considering the dimensions of the input feature map, the convolutional filter, and the output feature map. Given an input feature map of size $I_H \times I_W$ with $C_{\text{in}}$ input channels, and a convolutional filter of size $K_H \times K_W$ with $C_{\text{out}}$ output channels, the dimensions of the output feature map $O_H \times O_W$ are determined by the padding ($P$) and stride parameters as in (2.4) and (2.5). These equations calculate the spatial dimensions of the output feature map, which is essential for accurately estimating the MACs.

$$MAC_{FC} = N_{in} \times N_{out} \tag{2.7}$$

In contrast to convolutional layers, the computational complexity of FC layers can be described by the following Equation, where each output neuron connects to all input neurons, as shown in Equation (2.7). Here, $N_{\text{in}}$ represents the number of input neurons (from the previous layer), and $N_{\text{out}}$ denotes the number of output neurons (in the current layer). Although FC layers may involve a large number of parameters, especially in earlier CNN architectures, their contribution to the total MAC count is often considerably smaller than that of convolutional layers in modern deep networks, where most of the computational burden is concentrated in convolution operations.

In addition to MAC operations, the number of parameters is a fundamental metric that reflects the memory footprint and learning capacity of a CNN model. Parameters primarily consist of learnable weights and biases in both convolutional and fully connected layers.

Unlike MACs, which represent the computational workload during inference, parameters determine the memory required to store the model. They directly impact training duration, risk of overfitting, and deployment feasibility—particularly on resource-constrained hardware platforms such as FPGAs. For convolutional and fully connected layers, the total number of parameters can be calculated as follows:

$$\text{Params}_{\text{conv}} = C_{\text{out}} \times (C_{\text{in}} \times K_H \times K_W + 1) \tag{2.8}$$

$$\text{Params}_{\text{fc}} = N_{\text{in}} \times N_{\text{out}} + N_{\text{out}} \tag{2.9}$$

Although FC layers can contain a large number of parameters, particularly in earlier CNN architectures, their contribution to the total MAC count—and hence inference time—is typically much lower than that of convolutional layers.

(2.6) directly gives the total number of MAC operations and parameters performed by the convolutional layer, providing a comprehensive measure of the computational load. The data in the Table 2.4 reveals the relationship between the CNN computational load and convolution layers of convolutional neural networks in detail. In general, increasing the number of convolution layers also causes the computational cost to increase. However, different architectural approaches directly affect this increase. For example, AlexNet has only five convolution layers and requires 666M MACs computation, while GoogleNet uses 57 convolution layers and 1.58B MACs. Although VGG16 and VGG19 models have 13 and 16 convolution layers, the MAC value of VGG19 shows a significant jump of 19.5B. This example shows that the VGG architecture requires larger filters and intensive computations. However, the accuracy rate of VGG19 (72.7%) shows a minimal increase compared to VGG16 (71.9%), suggesting that the contribution of the computational load to the accuracy may be limited.

Table 2.4 shows that higher computational complexity does not always lead to better accuracy. For instance, VGG19's significant increase in MACs leads to only a slight increase in accuracy over VGG16. This observation reinforces the importance of balancing computational efficiency and model performance. The main factors affecting the MAC values among networks with similar layers are filter size, number of filters, stride, padding, layer types and use of fully connected layers. Evaluating the computational load requires more than just counting layers; it must also consider architectural design choices, filter dimensions, and convolution strategies. However, adjusting these

factors is not the only way to reduce computational load. Various optimisation algorithms have been developed to improve the efficiency of convolution layers. These techniques are examined in terms of computational complexity reduction, pipeline efficiency, and memory savings. The following section will discuss various algorithmic approaches to these optimisation processes.

While fully connected layers can dominate the parameter count in some CNN architectures, particularly in earlier models like AlexNet or VGG, their computational contribution—measured in MACs—is often minor compared to convolutional layers in modern networks. This is especially relevant for inference efficiency, where convolutional operations typically constitute the majority of hardware workload. Moreover, optimising convolutional stages can indirectly reduce the parameter size of fully connected layers, as the dimensionality of FC inputs depends on the feature map size produced by prior layers. Given this disproportionate computational burden, the subsequent sections will focus on algorithmic strategies aimed explicitly at optimising convolutional layers, which offer the most promising avenue for improving performance and efficiency on hardware-constrained platforms such as FPGAs.

TABLE 2.4: Popular CNN models with their computational workload. Accuracy measured on single-crops of ImageNet test-set.Image size is 224x224 [76]

| **CNN Models** | **AlexNet [77]** | **GoogleNet [78]** | **VGG16 [79]** | **VGG19 [79]** | **ResNet50 [80]** | **ResNet101 [80]** | **ResNet152 [80]** |
|---|---|---|---|---|---|---|---|
| Top-1 Accuracy | 57.10% | 68.70% | 71.90% | 72.70% | 75.30% | 76.40% | 77.30% |
| Conv Layers | 5 | 57 | 13 | 16 | 53 | 104 | 155 |
| Conv Workload (MACs) | 666 M | 1.58 B | 1.53 B | 19.5 B | 3.86 B | 7.57 B | 11.3 B |
| Conv Parameters | 2.33 M | 5.97 M | 14.7 M | 20 M | 23.5 M | 42.4 M | 58 M |
| Activation Layers | ReLU | ReLU | ReLU | ReLU | ReLU | ReLU | ReLU |
| Pool Layers | 3 | 14 | 5 | 5 | 2 | 2 | 2 |
| FC Layers | 3 | 1 | 3 | 3 | 1 | 1 | 1 |
| FC Parametrs | 58.6 M | 1.02 M | 124 M | 124 M | 2.05 M | 2.05 M | 2.05 M |
| Total Workload (MACs) | **724 M** | 1.58 B | 15.5 B | 19.6 B | 3.86 B | 7.57 B | 11.3 B |
| Total Parameters | 61 M | **6.99 M** | 138 M | 144 M | **25.5** M | 44.4 M | 60 M |

### 2.3.3 Algorithms for Accelerating Convolution Operations

Convolutional operations constitute the most computationally intensive part of CNNs, and optimising these operations is essential for efficient deployment on hardware-constrained platforms such as FPGAs. Due to the large number of MACs, the high arithmetic complexity often makes convolution layers the main bottleneck in terms of latency, power consumption, and resource utilisation. As CNN models grow deeper and more complex, the need for algorithmic strategies that reduce this computational burden becomes increasingly important [81, 82].

Several algorithmic strategies have been proposed to accelerate convolution operations by reducing computational load and improving memory efficiency [39, 83, 84]. Prominent approaches include direct convolution, the im2col transformation combined with General Matrix Multiplication (GEMM), frequency-domain methods based on the Fast Fourier Transform (FFT), and the Winograd minimal filtering algorithm. Each technique introduces distinct trade-offs in terms of computational efficiency, implementation complexity, and hardware compatibility. This section compares these methods, focusing on their strengths, limitations, and practical applicability within FPGA-based CNN accelerators.

#### 2.3.3.1 Direct Convolution

Direct convolution implements the standard mathematical definition of 2D convolution. For an input feature map $I$ with dimensions $(H_{in} \times W_{in} \times C_{in})$ and a set of filters $W$ with dimensions $(K_h \times K_w \times C_{in} \times C_{out})$, the convolution operation produces an output feature map $O$ with dimensions $(H_{out} \times W_{out} \times C_{out})$. The computation for each output pixel at position $(y, x)$ for output channel $k$ is given by [85]:

$$O(y, x, k) = B(k) + \sum_{c=0}^{C_{in}-1} \sum_{j=0}^{K_h-1} \sum_{i=0}^{K_w-1} I(y \cdot S + j, x \cdot S + i, c) \cdot W(k, c, j, i) \tag{2.10}$$

where $S$ denotes the stride and $B(k)$ is the bias term for the $k$-th output channel. The spatial dimensions of the output are adjusted based on padding strategy.

**Computational Complexity:** The total number of multiplications (and similarly additions) for direct convolution is:

$$\mathcal{O}(H_{out} \cdot W_{out} \cdot C_{out} \cdot C_{in} \cdot K_h \cdot K_w) \tag{2.11}$$

This complexity grows linearly with the number of channels and quadratically with the kernel size. For typical CNN layers with large input sizes and multiple filters, this quickly becomes computationally intensive [85].

**Suitability for FPGA:** Direct convolution maps naturally to hardware design frameworks such as HDL or HLS. Its structure aligns well with nested loops and pipelined execution models, which are commonly used to implement high-throughput architectures. However, to achieve optimal performance, careful scheduling is needed to ensure high DSP and LUT utilization. Line buffers and loop unrolling are often applied to exploit data reuse and reduce memory access latency.

#### 2.3.3.2 im2col Transformation with GEMM

The *im2col* technique restructures the convolution operation into a matrix multiplication problem, allowing efficient use of optimized General Matrix Multiply (GEMM) routines such as BLAS or systolic array implementations. The process unfolds in two main stages [83]:

- **im2col (Image to Column):** Patches from the input feature map are flattened and stored as columns in an intermediate matrix $M_{im2col}$. If the receptive field size is $K_h \times K_w \times C_{in}$ and the output feature map has size $H_{out} \times W_{out}$, then $M_{im2col}$ has dimensions $(K_h \cdot K_w \cdot C_{in}) \times (H_{out} \cdot W_{out})$.
- **GEMM:** The filter weights are reshaped into a matrix $M_{weights}$ of size $C_{out} \times (K_h \cdot K_w \cdot C_{in})$. The matrix multiplication $M_{output} = M_{weights} \cdot M_{im2col}$ yields the final output feature map as a flattened matrix of dimensions $C_{out} \times (H_{out} \cdot W_{out})$.

**Computational Complexity:** The arithmetic complexity remains the same as direct convolution:

$$\mathcal{O}(H_{out} \cdot W_{out} \cdot C_{out} \cdot C_{in} \cdot K_h \cdot K_w) \tag{2.12}$$

However, practical performance depends heavily on the efficiency of the GEMM backend and memory bandwidth.

**Memory and FPGA Considerations:** The main drawback of the im2col method is the substantial memory overhead introduced by duplicating overlapping receptive fields

in the expanded matrix $M_{\text{im2col}}$. This increases the memory footprint, posing a challenge for FPGAs with limited on-chip memory. While GEMM operations can be efficiently mapped to systolic arrays or DSP blocks, the im2col step often creates resource bottlenecks, requiring additional strategies such as loop tiling or on-the-fly buffering [86].

#### 2.3.3.3 FFT-Based Convolution

An alternative to spatial-domain convolution is to perform the operation in the frequency domain using the Fast Fourier Transform (FFT). This method leverages the Convolution Theorem, which states that convolution in the spatial domain corresponds to element-wise multiplication in the frequency domain [84]. Mathematically, the operation can be expressed as:

$$O = \mathcal{F}^{-1}\left(\mathcal{F}(I) \odot \mathcal{F}(K)\right) \tag{2.13}$$

Here, $\mathcal{F}$ and $\mathcal{F}^{-1}$ represent the forward and inverse FFT operations, $I$ denotes the input feature map, $K$ is the convolution kernel, and $\odot$ indicates element-wise (Hadamard) multiplication in the frequency domain.

**Computational Complexity:** The computational cost of FFT-based convolution is primarily dominated by the FFT and inverse FFT operations. Using efficient implementations such as the Cooley–Tukey algorithm, the complexity becomes:

$$\mathcal{O}(N^2 \log N) \tag{2.14}$$

where $N$ is the spatial dimension of the input feature map (i.e., the input size is $N \times N$). This complexity is independent of the kernel size $K$, making FFT particularly advantageous for convolutions involving large filters.

**Advantages:**

- **Efficient for Large Kernels:** FFT reduces the number of arithmetic operations when $K$ is large relative to the input, offering computational benefits for wide kernels.
- **Parallelisable Structure:** FFT operations can be parallelised across input tiles and channels, which allows high-throughput implementations on parallel hardware such as FPGAs.

**FPGA Implementation Challenges:**

- **Pre/Post-Processing Overhead:** For small kernel sizes, the overhead of performing forward and inverse FFT operations may outweigh the benefits of reduced multiplication cost.
- **Resource Usage:** FFT/IFFT modules can require significant logic, DSP, and memory resources, making them less attractive for resource-constrained FPGA deployments.
- **Latency Sensitivity:** The pipeline depth introduced by FFT stages can lead to increased latency, which may conflict with the low-latency requirements of real-time applications.

### 2.3.4 Winograd Minimal Filtering Algorithm

The Winograd minimal filtering algorithm reduces the number of multiplications required for small convolutions by transforming the computation into a so-called Winograd domain. Based on polynomial interpolation and the Chinese Remainder Theorem, the algorithm transforms input tiles and kernels, performs element-wise multiplications, and transforms the result back to the spatial domain.

The 2D Winograd convolution is typically denoted as $F(m, r)$, where $m$ is the size of the output tile and $r$ is the kernel size. For example, $F(2, 3)$ computes a $2 \times 2$ output tile using a $3 \times 3$ kernel. The output $Y$ is computed using the following equation [39]:

$$Y = A^T \left[ (GgG^T) \odot (B^T dB) \right] A \tag{2.15}$$

Here, $d$ is the input tile, $g$ is the filter kernel, and $A$, $B$, and $G$ are the Winograd transformation matrices defined for each $F(m, r)$ configuration. The operator $\odot$ denotes element-wise multiplication.

**Transformation Matrices:** The matrices $A$, $B$, and $G$ are predefined for common configurations and consist of small integers or simple fractions (e.g., 0, 1, $\pm 1$, $\frac{1}{2}$), allowing the transformations to be implemented using only additions, subtractions, and bit-shifts. This eliminates the need for general-purpose multipliers and reduces hardware complexity.

For instance, in the $F(2 \times 2, 3 \times 3)$ case, the number of multiplications is reduced from 36 (direct convolution) to 16 using Winograd, yielding a 2.25× reduction [39] A similar reduction is observed for $F(4 \times 4, 3 \times 3)$, where the number of multiplications drops

from 144 to 36. However, this reduction comes at the cost of increased additions and transformation overhead.

**Complexity Analysis:** The number of multiplications in Winograd is reduced from $m^2 \cdot r^2$ (in direct convolution) to approximately $(m + r - 1)^2$. This makes Winograd particularly beneficial for small kernels, where the multiplication cost dominates. However, the increase in additions and memory usage due to transformations must be taken into account.

**FPGA Implications:** Winograd's reduction in multiplications is well-suited for FPGAs where DSP blocks are limited. However, the increased control logic complexity, transformation overhead, and potential numerical instability may require careful balancing when implementing Winograd on resource-constrained platforms [46].

Each convolution acceleration method reflects a unique trade-off among arithmetic complexity, memory usage, and hardware suitability. While direct convolution is straightforward and FPGA-friendly, it lacks computational efficiency. im2col enables high reuse of optimised matrix multipliers but suffers from memory duplication. FFT methods are advantageous for large kernels but introduce transform overhead and precision concerns. Winograd offers a significant reduction in multiplications for small kernels but complicates control logic and memory usage. Selecting the optimal method depends on kernel size, network architecture, and the specific resource constraints of the FPGA platform.

### 2.3.5 Discussion on FPGA Suitability of Convolution Techniques

Table 2.5 presents a qualitative comparison of four widely-used convolution acceleration techniques in terms of their suitability for FPGA implementation. This section expands upon that comparison by evaluating each technique based on several key criteria, highlighting the rationale behind the selection of the Winograd algorithm for this study.

**FPGA Resource Fit:** Direct convolution relies heavily on DSP blocks for performing MAC operations. While this mapping is straightforward and efficient for small networks, DSPs are typically limited in number on FPGAs, making scalability difficult. The im2col+GEMM approach performs matrix multiplication efficiently using systolic arrays or DSPs, but its excessive reliance on large BRAM buffers restricts its scalability due to memory bandwidth bottlenecks. FFT-based approaches, although promising in theory, require complex arithmetic and consume both DSPs and LUTs, putting stress on both resources. In contrast, the Winograd algorithm strategically reduces the number

TABLE 2.5: Qualitative Comparison of Convolution Algorithms for FPGA Suitability

| Metric | Direct Conv.[87] | im2col+GEMM [88] | FFT-Based [89] | Winograd [46] |
|---|---|---|---|---|
| FPGA Resource Fit | Medium (DSP-heavy) | Low (BRAM-limited) | Low (Complex Arith.) | High (LUT-efficient) |
| Memory Requirement | Low | Very High | Moderate/High | Moderate |
| Numerical Stability | High | High | Precision Sensitive | Challenging (Low Prec.) |
| Best Suited Kernels | Small | All (if memory allows) | Large kernels | Small (3x3, 5x5) |
| Implementation Cost | Low | High (transforms+buffer) | High (FFT logic) | Medium (transform logic) |
| Overall FPGA Suitability | Medium | Low | Conditional | High |

of multiplications and instead employs additions and shifts—operations well-suited to LUTs, which are abundant on FPGAs. This makes Winograd the most resource-efficient option.

**Memory Requirement:** The memory footprint of im2col+GEMM is substantially high due to the need to store flattened and duplicated input patches. FFT-based convolution also suffers from moderate to high memory demand, primarily for storing frequency-domain data and precomputed twiddle factors. Direct convolution requires only modest memory resources for line buffering. Winograd presents a moderate memory footprint by using small buffers for intermediate tiles and transformation matrices, balancing efficiency and feasibility for FPGA-based deployment.

**Numerical Stability:** Direct convolution and im2col+GEMM are numerically stable due to the use of real-valued arithmetic. FFT-based methods are highly sensitive to quantization and numerical precision, often requiring floating-point arithmetic to avoid instability. Winograd also presents stability challenges, particularly for larger tiles and low-precision applications. However, its use with small tile sizes (e.g., $F(2 \times 2, 3 \times 3)$) mitigates these concerns and enables reliable deployment using fixed-point representations.

**Best Suited Kernels:** Direct convolution and im2col+GEMM can support arbitrary kernel sizes. FFT-based methods are primarily beneficial for large kernels, which are uncommon in contemporary CNNs. Winograd is highly optimized for small kernels, particularly $3 \times 3$ and $5 \times 5$, which are prevalent in many modern architectures such as VGG, ResNet, and MobileNet. This alignment with common kernel dimensions strengthens its applicability.

**Implementation Cost:** Direct convolution has a low implementation cost due to its straightforward nested-loop structure, though its performance is limited by DSP availability. im2col+GEMM and FFT-based methods incur significant implementation complexity due to memory management and transformation logic. Winograd has a moderate cost associated with implementing transformation matrices but compensates for this with reduced arithmetic complexity and better mapping to parallel hardware structures.

**Overall FPGA Suitability:** When all factors are considered—resource fit, memory usage, numerical behavior, kernel alignment, and implementation overhead—Winograd emerges as the most suitable algorithm for FPGA-based CNN acceleration. It offers high throughput potential, efficient utilization of FPGA resources, and is well-aligned with the architectural characteristics and kernel structures found in modern convolutional networks.

While each convolution technique exhibits its own merits, the Winograd algorithm offers the most favourable trade-off between computational efficiency and FPGA resource utilization. Its key strength lies in reducing the number of costly multiplications—an operation bound to the limited DSP blocks—by leveraging additions and shifts that are efficiently handled by abundant LUTs. This alignment with FPGA architecture not only alleviates pressure on scarce computational units but also enables better scalability and energy efficiency. Furthermore, the algorithm's suitability for small kernel sizes, which dominate contemporary CNN models, makes it practically aligned with real-world deployment scenarios. Although Winograd introduces transformation overhead and poses numerical precision challenges for certain configurations, these drawbacks are manageable with appropriate tiling strategies and fixed-point quantisation. Overall, Winograd's balance of low arithmetic cost, hardware compatibility, and practical relevance positions it as a compelling and strategic choice for high-performance CNN acceleration on resource-constrained FPGA platforms.

Given its potential to reduce arithmetic complexity, the Winograd minimal filtering algorithm offers an opportunity to accelerate CNN inference on FPGAs. However, its practical realisation is fraught with critical implementation challenges that must be carefully addressed. Translating this theoretical advantage into practical and efficient hardware implementations presents several substantial challenges.

Numerical instability is a critical concern, particularly in low-precision environments commonly used in FPGA-based inference engines (e.g., INT8). The structure of Winograd transformations can amplify quantization and rounding errors, especially when larger tile configurations such as $F(4 \times 3, 3 \times 3)$ are employed. Addressing these issues requires co-design strategies, including careful tile size selection, tuning of transformation polynomials, and the use of Winograd-aware quantization techniques.

Hardware resource constraints also represent a significant challenge. While the algorithm reduces reliance on DSP slices for multiplication, it increases demand on the logic fabric, particularly Look-Up Tables (LUTs) and flip-flops (FFs), due to the complexity of the transformation stages and associated control logic. This resource shift can limit achievable clock frequencies and overall throughput, requiring careful balancing of computational workload and resource allocation tailored to the characteristics of the target FPGA architecture.

Furthermore, Winograd's multi-stage computation pipeline imposes substantial data buffering requirements, exacerbating pressure on the memory subsystem. Limited off-chip bandwidth and constrained on-chip BRAM capacity can become bottlenecks. Effective implementations necessitate advanced memory management techniques, including line buffering, optimised tiling flows such as double buffering, and BRAM partitioning to maximise data reuse and hide latency.

Although the transformation logic is computationally lightweight, it still introduces latency and control complexity that must be offset by the reduction in arithmetic operations during the element-wise multiplication phase. Additionally, designing a flexible and reusable Winograd hardware accelerator capable of supporting various CNN parameters—such as kernel sizes, stride lengths, and tile configurations—poses considerable challenges. Approaches such as kernel decomposition can extend the applicability of Winograd to non-unit strides and larger kernels, but these often result in significant control overhead and data orchestration complexity at the system level.

### 2.3.6 FPGA-Based Acceleration of CNNs

FPGAs are programmable hardware platforms consisting of configurable logic blocks, known as look-up tables (LUTs), complemented by specialised resources such as digital signal processing (DSP) elements and block RAMs (BRAMs) [22]. Their highly parallel architecture enables concurrent execution of multiple operations, thus delivering superior performance-per-watt compared to traditional processors [25]. As a result, FPGAs have gained considerable attention for accelerating machine learning workloads, particularly in applications where low latency, high throughput, and energy efficiency are paramount [28].

Numerous studies have explored deploying CNNs on FPGAs to meet the increasing demand for efficient deep learning inference. These implementations vary widely regarding numerical precision, model architecture, and parallelisation strategy. For instance, Liang et al. [46] conducted a comprehensive evaluation of fast convolution algorithms on FPGAs by deploying multiple CNN models, including AlexNet, VGG, ResNet, and YOLO, using both standard and accelerated methods. They compared direct convolution, Winograd minimal filtering, FFT-based methods, and im2col + GEMM

in terms of throughput, energy efficiency, and hardware utilisation. On a ZCU102 FPGA with 16-bit precision, their Winograd-based implementation of AlexNet achieved 854.6 GOP/s and 36.2 GOP/s/W, outperforming the conventional version (202.8 GOP/s, 21.4 GOP/s/W). Similar improvements were observed for VGG, which reached 2601.3 GOP/s and 105.4 GOP/s/W. However, the study employed a uniform optimisation strategy across all layers and did not explore the differential suitability of Winograd for layers with varying computational profiles, limiting insights into its effectiveness in deeper and more heterogeneous network architectures.

Zhang et al. [90] proposed an 8-bit quantized CNN accelerator tailored to low-power embedded FPGA platforms. The parameterizable design integrates convolutional and fully connected layers, and uses loop unrolling and pipelining to enhance parallelism. Implemented on an Xilinx ZCU104 board, the architecture demonstrated notable energy efficiency and preserved high classification accuracy on LeNet-5, with only a 0.75% drop compared to the original 32-bit model (99.06% vs 98.31%). However, its performance on more complex architectures such as AlexNet was limited, as INT8 quantization achieved only 55.99% accuracy, which highlights challenges in generalization and scalability across deeper models.

Li et al. [91] proposed a real-time, ultra-low-power CNN accelerator targeting embedded vision applications. Their design employs a multi-level pipeline architecture combining a line buffer-based convolution engine with a global memory access optimization strategy to minimize bandwidth usage. Implemented on an Intel Arria 10 FPGA, their system achieved 27.2 W power consumption while delivering competitive throughput for models such as AlexNet. Although the accelerator demonstrated strong efficiency, its optimization was primarily focused on memory reuse and pipelining within a fixed kernel architecture, without exploring dynamic or layer-specific strategies. Furthermore, the study does not provide explicit accuracy figures, leaving the model's predictive performance and generalisation ability unassessed.

Vestias et al. [92] proposed a configurable CNN accelerator architecture specifically tailored for low-density FPGAs, focusing on efficient resource utilisation in constrained environments. Their design partitions CNN layers into convolutional and non-convolutional groups, allowing different execution paths and optimisations based on the layer type. The implementation targets a Xilinx Zynq-7020 device, using fixed 8-bit precision to reduce power and resource usage. The accelerator achieves a throughput of 229 images per second while maintaining minimal hardware footprint, consuming only 212 DSPs and 46,914 LUTs. However, the accelerator achieves a relatively low classification accuracy of 54.7%, indicating that the design prioritises hardware efficiency over predictive performance, and highlighting a potential trade-off that may limit its applicability in accuracy-critical tasks.

Wang et al. [93] proposed a design flow for deploying extremely low bit-width neural networks (ELB-NNs) on embedded FPGAs. Their method enabled layer-specific quantisation using binary, ternary, and low-precision representations to optimise the balance between resource utilisation and model performance. The implementation delivered high throughput and energy efficiency across multiple benchmarks, demonstrating the hardware benefits of aggressive quantisation strategies. However, these improvements were accompanied by a noticeable reduction in classification accuracy. For instance, Top-1 accuracy in AlexNet decreased from 55.9% to 49.3%, raising concerns about the suitability of such designs in applications where predictive reliability is essential. This underscores the need to jointly consider efficiency and accuracy when designing hardware-aware neural networks.

Neelam and Prince [94] introduced VConv, a CNN accelerator designed for FPGA deployment with an emphasis on energy efficiency and parallelism. Their architecture was implemented using a Xilinx Artix-7 FPGA (5×XC7A200T), employing 32-bit floating-point precision and optimising convolution layers through dedicated hardware modules. The accelerator achieved a throughput of 15 images per second and reported a peak performance of 56 GOPS. However, the study did not provide comprehensive details regarding classification accuracy, leaving uncertainties about the balance between speed and predictive performance. This omission is particularly notable for applications where inference accuracy is critical, indicating that while VConv shows efficiency in hardware utilisation, its effectiveness in real-world scenarios remains partially unverified.

Existing FPGA-based CNN accelerators demonstrate impressive improvements in energy efficiency and inference throughput. However, many approaches apply uniform scheduling or aggressive quantisation without adequately considering the computational variability across layers. As a result, they often overlook the impact of these choices on model generalisation and predictive accuracy. This highlights the need for adaptive, layer-aware design strategies that balance efficiency with reliable performance across diverse CNN architectures.

## 2.4 Motivation and Computational Requirements in Cancer Detection

While the proposed techniques in this thesis are not limited to any specific medical condition, cancer detection is selected as a representative application domain due to its particularly demanding computational characteristics. The clinical requirements for high accuracy, low latency, real-time inference, and privacy-aware processing make cancer detection an ideal use case for evaluating performance-critical hardware-software co-design strategies.

### 2.4.1 High Accuracy Expectations and Low Error Tolerance

Early-stage cancer detection is a high-stakes task that demands high accuracy [95]. For instance, early-stage diagnosis of oesophageal cancer is associated with improved five-year survival rates and increased eligibility for curative interventions such as surgery [96, 97]. Therefore, even small diagnostic errors may have serious clinical consequences.

False negatives can delay treatment and allow disease progression, while false positives, though less severe, may lead to unnecessary procedures, increased healthcare costs, and patient anxiety [98, 99]. This highlights the need for high sensitivity and specificity in AI-assisted diagnostics.

Even modest reductions in model performance can undermine clinical confidence and impede real-world adoption. This concern is amplified by studies reporting that AI systems, while accurate in controlled settings, often exhibit degraded performance when evaluated in real-world clinical environments [100]. Consequently, optimisation techniques must be applied with caution, ensuring that diagnostic accuracy is preserved throughout deployment.

### 2.4.2 Real-Time Inference Requirements: Latency and Throughput Constraints

Real-time cancer detection systems are subject to stringent computational constraints involving both inference latency and throughput [101]. Latency refers to the time required to generate a prediction for a single input, while throughput denotes the number of inputs processed per unit time. These performance metrics become especially critical in applications involving continuous or high-frequency data acquisition—such as video-based medical imaging—where system responsiveness directly influences the timeliness and reliability of diagnostic support [102].

For example, endoscopic cancer screening provides a representative scenario in which strict latency and throughput requirements are essential. Clinical endoscopy systems typically operate at 25–30 frames per second (fps), requiring inference times below 33–40 milliseconds per frame to keep pace with incoming video [103]. Studies have shown that failure to meet these constraints can delay diagnostic feedback, with real-time systems often defined by a minimum of 24 fps [104]. Systems unable to meet this threshold—for example, those operating at only 4 fps due to 240 ms frame processing—are considered inadequate for real-time use [103].

### 2.4.3 Privacy and On-Device Inference Considerations

In the context of cancer detection and other sensitive medical imaging applications, on-device inference—where data is processed locally on embedded hardware such as

FPGAs or edge AI devices—offers substantial advantages in terms of privacy, latency, and regulatory compliance. By keeping all patient data within the local device or hospital network, on-device processing minimises the risk of data breaches associated with network transmission and aligns with legal frameworks such as GDPR and HIPAA [105, 106]. Several studies highlight that edge-based AI systems eliminate the need to transfer protected health information (PHI) to external servers, thereby reducing exposure and simplifying data governance [105, 107].

### 2.4.4 Case Study Selection: Oesophageal Cancer

Optimisation techniques are intended to be generally applicable to CNN-based image analysis tasks, oesophageal cancer detection is presented as a representative case study because it embodies both clinically significant challenges and demanding computational characteristics. According to World Health Organization (WHO) data in 2020, there were approximately 604,000 new cases of oesophageal cancer and around 544,000 related deaths worldwide [108]. Morgan et al. project that by 2040, the global burden will increase to 957,000 new cases and 806,000 deaths [108]. Given this high mortality rate and the need for early and accurate diagnosis, oesophageal cancer offers a meaningful and data-supported context for examining the computational demands of hardware-accelerated diagnostic pipelines [42].

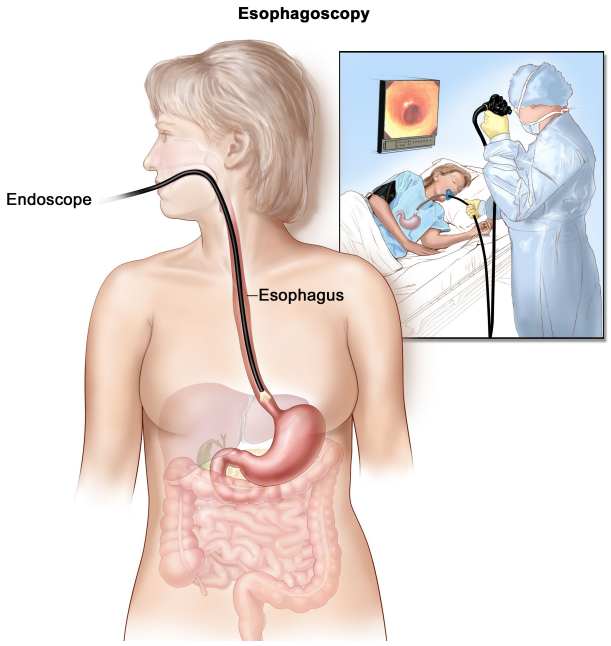



FIGURE 2.2: Oesophageal endoscopy [109]

From a computational perspective, oesophageal cancer screening is typically performed using high-resolution RGB video captured during endoscopic procedures. Unlike volumetric imaging modalities such as computed tomography (CT) or magnetic resonance imaging (MRI), endoscopic imaging involves 2D, frame-based input, which can

be efficiently processed by convolutional neural networks optimised for spatial inference. This makes the domain particularly suitable for FPGA-based acceleration, where frame-wise latency and throughput can be tightly controlled. Furthermore, publicly available, annotated endoscopic datasets allow for reproducible experiments and quantitative evaluation, strengthening the viability of this application as a benchmark scenario. Figure 2.2 shows the cancer detection method with endoscopy.

Accordingly, oesophageal cancer serves as a clinically grounded and computationally illustrative use case, helping to justify the broader focus of this thesis on real-time, accuracy-critical image processing scenarios.

## 2.5 Summary

Deploying CNNs on FPGAs presents a promising yet challenging avenue for achieving high-performance inference in power—and latency-constrained environments. This chapter has established that FPGAS' inherent reconfigurability, parallelism, and energy efficiency make them highly suitable for accelerating convolution-heavy workloads, particularly when conventional platforms like CPUs and GPUs fall short in adaptability and efficiency.

A key insight from the chapter is the disproportionate computational burden imposed by convolutional layers in modern CNNs. As such, optimising these layers is critical for unlocking the full performance potential of FPGA-based accelerators. Among the algorithmic strategies explored, the Winograd minimal filtering algorithm stands out due to its ability to drastically reduce multiplication operations—offering a distinct advantage in DSP-limited FPGA architectures.

To support this selective deployment strategy, the chapter introduced a structured comparative analysis framework that systematically evaluated the FPGA suitability of various convolution algorithms. By examining factors such as arithmetic complexity, resource utilisation, and memory requirements, this framework contextualised Winograd's strengths while illuminating its limitations in comparison to alternatives like direct convolution, im2col+GEMM, and FFT-based methods.

The computational challenges discussed in this chapter are highly relevant to application domains such as cancer detection, which often involve real-time inference, high-resolution image processing, and strict performance constraints. In such scenarios, convolution operations must be carefully optimised to balance latency, accuracy, and resource usage. These demands make cancer detection a representative and technically challenging use case for assessing the trade-offs and potential benefits of advanced convolution acceleration techniques on FPGA platforms.

Ultimately, this chapter laid the theoretical and practical foundation necessary for guiding Winograd's selective application in hardware-accelerated CNN inference. It set the stage for the next chapter, introducing a structured optimisation methodology tailored to maximise performance while preserving accuracy and respecting resource constraints.

# Chapter 3

# Layer-Level CNN Optimization for FPGA Implementation

## 3.1 Introduction

CNNs have revolutionised image classification and object detection tasks by automatically learning hierarchical feature representations from raw input data [110]. Since early architectures like LeNet [111], the field has advanced towards deeper and more complex models such as AlexNet [77], VGG [79], and ResNet [80]. However, this growth in architectural complexity has substantially elevated computational demands, making efficient deployment on resource-constrained platforms such as FPGAs increasingly challenging.

CNNs are composed of multiple types of layers—including convolutional, activation, pooling, and fully connected layers—that collectively contribute to their computational burden. Among these, convolutional layers dominate the computation, primarily due to the large MAC operations they require. This computational bottleneck necessitates the development of optimisation strategies to reduce the complexity of CNNs while maintaining predictive performance, particularly for real-time and embedded applications.

Several algorithmic approaches have been explored to alleviate this burden, including Winograd convolution [39], FFT-based methods [112], and GEMM-based optimisations [113]. Among these, the Winograd algorithm stands out due to its ability to reduce the number of MAC operations, especially for small convolutional kernels such as $3 \times 3$. Its suitability for FPGAs stems from its lower arithmetic complexity and ability to exploit fine-grained parallelism, making it highly effective for resource-limited environments [114].

Building on this motivation, Chapter 3 presents a systematic framework for optimising CNN architectures through layer-wise analysis and Winograd-based restructuring. A Suitability Score is introduced to quantitatively assess the compatibility of each convolutional layer with Winograd transformations, enabling targeted structural-level optimisations. Unlike traditional global model compression techniques, the proposed approach adapts the network architecture layer by layer, selectively modifying kernel sizes and output channel counts to enhance Winograd compatibility without sacrificing overall model accuracy.

By applying this methodology to different CNN architectures, including AlexNet and VGG16, the chapter demonstrates that substantial reductions in computational complexity and training time can be achieved while preserving high inference accuracy. These findings establish the foundation for hardware-level optimisations explored in later chapters, ultimately enabling scalable, efficient, and accurate CNN deployment on FPGA platforms.

## 3.2 Proposed Suitability Score For Convolution Layer Optimization using Winograd Algorithm

The Winograd algorithm reduces the computational load of convolutional layers by reducing the number of MAC operations. However, this algorithm's benefits may not be the same across all layers in a neural network. To address this, a *Suitability Score ($S_{score}$)* is introduced to identify the convolutional layers most compatible with the Winograd algorithm. The primary purpose of this approach is to maximise the benefits of the Winograd Method by optimising the CNN structure in a way that is compatible with the Winograd method.

$$S_{\text{score}} = \alpha \cdot \left(\frac{K_{\text{ref}}}{K}\right) + \beta \cdot \left(\frac{\text{MACs}}{\text{MACs}_{\text{ref}}}\right) + \gamma \cdot \left(\frac{C_{\text{out}}}{C_{\text{out}_{\text{ref}}}}\right) \tag{3.1}$$

$S_{\text{score}}$ is designed to quantify how well a convolutional layer in a neural network aligns with the strengths of the Winograd algorithm. This score is computed by evaluating three key factors: kernel size, computational load, and feature complexity, as in (3.1). Where $K$ represents the kernel size of the convolutional layer under consideration; MACs denotes the total number of multiply-accumulate operations required by the layer, which reflects its computational load; and $C_{\text{out}}$ is the number of output channels, indicating the feature complexity of the layer. Determining reference values for kernel size, computational complexity, and output channel count is essential to effectively applying the suitability score. $K_{\text{ref}}, \text{MACs}_{\text{ref}}$, and $C_{\text{out}_{\text{ref}}}$ represent the reference values for $S_{\text{score}}$. These values may vary during the optimization phase of the CNN model. Therefore, these values must be determined dynamically.

### 3.2.1 Selection Criteria for Suitability Score Parameters

In constructing the Suitability Score, three key architectural factors are considered: kernel size, computational load in terms of MAC operations, and the number of output channels. Each parameter captures a distinct aspect of a layer's suitability for Winograd-based acceleration. The kernel size term reflects the algorithmic compatibility of a layer, as smaller kernels (e.g., $3 \times 3$) are more amenable to Winograd transformations [115]. The MAC term quantifies the computational complexity associated with each layer, which is critical for identifying potential bottlenecks in FPGA deployments [21, 116]. Since each MAC operation corresponds to a multiplication and accumulation, the total MAC count directly reflects the underlying DSP workload and thus the intensity of the required computations. The output channel count term represents the memory and DSP resource demands, as layers with a large number of filters consume proportionally more LUTs, DSP slices, and BRAM, which in turn constrains scalability [117]. Taken together, these factors provide a comprehensive view of both the algorithmic and hardware-related characteristics of each layer.

To integrate these aspects into a single interpretable metric, the three factors are linearly combined, as shown in Equation (3.1). A linear formulation strikes a practical balance between expressiveness and computational simplicity. It enables flexible weighting of the individual components through the coefficients $\alpha$, $\beta$, and $\gamma$, which sum to one. This structure allows for rapid evaluation of CNN layers during the optimisation process without introducing the complexity of non-linear functions or additional hyperparameters. Moreover, the linear combination reflects the additive nature of these factors: a layer with a favourable kernel size but a large number of filters, for example, would receive a moderate score, as both algorithmic compatibility and resource demands are proportionally represented. This ensures that the score remains both theoretically grounded and practically efficient for iterative optimisation strategies.

### 3.2.2 Determining the Reference Values of $S_{\text{score}}$

When determining the $K_{\text{ref}}$ value, it is necessary first to examine the relationship between the working principle of the Winograd algorithm and the kernel size. Unlike the large-sized kernel size of the Winograd method, it is more effective in small sizes. Therefore, the $K_{\text{ref}}$ value should be selected as 3, the size where Winograd is most effective. However, depending on design purposes, this value can be selected over other kernel values. In order to establish the ideal kernel size for $K_{\text{ref}}$, the generalized formula can be used as in (3.2).

$$K_{ref} = \begin{cases} \arg\min_{K}(f(K)), & \text{if the goal is to minimize } f(K) \\ \arg\max_{K}(f(K)), & \text{if the goal is to maximize } f(K) \end{cases} \tag{3.2}$$

$f(K)$ represents a chosen optimization metric, which could vary depending on the specific design objectives, such as minimizing MAC operations for computational efficiency or maximizing DSP efficiency for optimal hardware resource utilization. This general formulation provides flexibility to optimize performance metrics based on the specific FPGA implementation goals.

$$\text{MACs}ref = \max_{l \in L}(\text{MACs}(l)) \tag{3.3}$$

Furthermore, when selecting a value for $MACs_{\text{ref}}$ from among different convolutional layers in a CNN, the formulation can be expressed as in (3.3). $MACs_{\text{ref}}$ is chosen dynamically by evaluating all convolutional layers ($L$) within the CNN model, where MACs($l$) indicates the MAC operations required for each convolutional layer $l \in L$. This formulation indicates that the reference MAC count is selected based on the layer with the maximum computational demand within the neural network layers.

$$C_{out_{ref}} = \max_{l \in L}(C_{out}(l)) \tag{3.4}$$

Although the Winograd algorithm reduces the number of MACs, this algorithm initially performs some transformation operations on the input data and the kernel. The cost of these transformation operations is fixed and independent of the number of output channels. In other words, these transformations are performed once under all conditions, and their results can be used jointly for all output channels. In this case, as the number of $C_{out}$ increases, this fixed cost is shared by more channels, and the computational load per channel decreases. Thus, in layers with high output channels, the total processing cost is reduced due to the transformation cost sharing and the computational cost reduction per channel. However, increasing the number of output channels is not a reason to directly apply the Winograd algorithm. High output channel numbers increase the computational load of standard convolution operations, making the Winograd algorithm's advantage more obvious. Therefore, layers with high output channel numbers can benefit more from the advantage provided by the Winograd algorithm. Accordingly, $C_{out_{ref}}$ is selected dynamically by choosing the maximum number of output channels among all convolutional layers within the CNN model as in (3.4).

### 3.2.3 Experimental Evaluation of Coefficient Configurations

#### 3.2.3.1 Coefficient Definition and Role

(3.1) has three coefficients, $\alpha$, $\beta$, and $\gamma$, which play a crucial role in determining the relative importance of each factor in assessing the suitability of a convolutional layer for Winograd optimisation. These coefficients represent filter compatibility, computational efficiency, and feature complexity. The coefficients enable the customisation of the Suitability Score to align with specific neural network architectures and application needs. One key condition in this adjustment process is that the sum of the coefficients must always equal 1, as shown in (3.5), which enables a balanced trade-off among filter compatibility, computational efficiency, and feature complexity.

$$\alpha + \beta + \gamma = 1 \tag{3.5}$$

*Filter Compatibility Factor ($\alpha$)* reflects the significance of the kernel size ($K$) in determining the layer's compatibility with the Winograd algorithm. Smaller kernel sizes, such as 3x3, are more compatible with Winograd, making this factor highly influential in the score.

*Computational Efficiency Factor ($\beta$)* represents the computational load of the layer and is usually related to the total MAC value. Since layers with higher MAC values require more computation, they benefit more from Winograd optimization. Therefore, it is appropriate to increase the value of $\beta$ in cases where the computational load is prioritized.

*Feature Complexity Factor ($\gamma$)* represents the complexity of the features extracted by the layer, indicated by the number of output channels. The fixed conversion cost of the Winograd algorithm is shared among more channels as the number of output channels increases, allowing the algorithm to operate more efficiently at large $C_{\text{out}}$. However, considering that the effect of $C_{\text{out}}$ is generally not as critical as $\alpha$ and $\beta$, it is recommended to keep the coefficient $\gamma$ lower.

A set of experiments is conducted to apply and evaluate the effect of different coefficient configurations on the Suitability Score. Each configuration targets a specific optimisation priority, such as kernel size, computational load, or output channel complexity, enabling comparative analysis of how these priorities influence layer selection.

#### 3.2.3.2 Experimental Setup

A series of experiments using CNN models is conducted to apply and evaluate the effect of different coefficient configurations on the Suitability Score. All experiments are implemented in Python using the Keras library with a TensorFlow backend.

The analysis begins with AlexNet, which offers structural diversity with varying kernel sizes and a relatively shallow architecture. This makes it suitable for clearly observing the influence of different coefficient configurations on the Suitability Score.

VGG16 is also included in the analysis to improve the generalisability and robustness of the findings. With its deeper structure and consistent 3x3 kernels, VGG16 provides a contrasting case that helps determine whether the observed effects are consistent across networks with different architectural properties.

| CNN Model | Conv. Layer | Kernel Size | Filters | Stride | MACs ($10^8$) |
|---|---|---|---|---|---|
| AlexNet | 1 | 11x11 | 96 | 4x4 | 1.02 |
| | 2 | 5x5 | 256 | 1x1 | 4.48 |
| | 3 | 3x3 | 384 | 1x1 | 1.49 |
| | 4 | 3x3 | 384 | 1x1 | 2.24 |
| | 5 | 3x3 | 256 | 1x1 | 1.49 |
| VGG16 | 1 | 3x3 | 64 | 1x1 | 0.89 |
| | 2 | 3x3 | 64 | 1x1 | 18.5 |
| | 3 | 3x3 | 128 | 1x1 | 9.26 |
| | 4 | 3x3 | 128 | 1x1 | 18.5 |
| | 5 | 3x3 | 256 | 1x1 | 9.26 |
| | 6 | 3x3 | 256 | 1x1 | 18.5 |
| | 7 | 3x3 | 256 | 1x1 | 18.5 |
| | 8 | 3x3 | 512 | 1x1 | 9.25 |
| | 9 | 3x3 | 512 | 1x1 | 18.5 |
| | 10 | 3x3 | 512 | 1x1 | 18.5 |
| | 11 | 3x3 | 512 | 1x1 | 4.63 |
| | 12 | 3x3 | 512 | 1x1 | 4.63 |
| | 13 | 3x3 | 512 | 1x1 | 4.63 |

TABLE 3.1: Architectural details of AlexNet and VGG16 models, including kernel size, number of filters, stride, and MACs per convolutional layer

Table 3.1 presents the architectural features of the AlexNet and VGG16 models used in the experiments, including kernel sizes, number of filters, stride values, and MAC operations, highlighting the structural differences between the two models.

#### 3.2.3.3 Results and Observations

Table 3.2 presents the coefficient configurations used to define different optimisation scenarios, which are designed to analyse their impact on the Suitability Score. According to the experimental results, it is observed that the $S_{\text{score}}$ value of layers with small kernel sizes increases in the **Kernel-focused scenario**, which prioritises only kernel size. This result shows that the Winograd algorithm is more effective in small kernel sizes. Similarly, in the **MACs-focused scenario**, which focuses only on computational load, it is determined that the $S_{\text{score}}$ increases in layers with high MAC values, and such layers benefit more from the Winograd algorithm.

| Scenario | $\alpha$ | $\beta$ | $\gamma$ | Layer | $S_{\text{score}}$ | Priority |
|---|---|---|---|---|---|---|
| Kernel-focused | 1 | 0 | 0 | 1 | 0.27 | Emphasizes small kernel sizes |
| | | | | 2 | 0.6 | |
| | | | | 3 | **1.00** | |
| | | | | 4 | **1.00** | |
| | | | | 5 | **1.00** | |
| MACs-focused | 0 | 1 | 0 | 1 | 0.22 | Prioritizes layers with high computational load |
| | | | | 2 | **1.00** | |
| | | | | 3 | 0.33 | |
| | | | | 4 | 0.50 | |
| | | | | 5 | 0.33 | |
| Output Channel focused | 0 | 0 | 1 | 1 | 0.25 | Emphasizes large output channel counts |
| | | | | 2 | 0.66 | |
| | | | | 3 | **1.00** | |
| | | | | 4 | **1.00** | |
| | | | | 5 | 0.66 | |
| Balanced Combination | 0.4 | 0.4 | 0.2 | 1 | 0.24 | Balanced approach prioritizing both kernel size and MACs |
| | | | | 2 | 0.77 | |
| | | | | 3 | 0.73 | |
| | | | | 4 | **0.80** | |
| | | | | 5 | 0.66 | |
| Kernel-weighted Combination | 0.6 | 0.2 | 0.2 | 1 | 0.26 | Strong emphasis on small kernel sizes with some MACs and Cout effect |
| | | | | 2 | 0.69 | |
| | | | | 3 | 0.86 | |
| | | | | 4 | **0.90** | |
| | | | | 5 | 0.80 | |
| MACs-weighted Combination | 0.2 | 0.6 | 0.2 | 1 | 0.25 | Prioritizes high computational load while considering other factors |
| | | | | 2 | **0.85** | |
| | | | | 3 | 0.60 | |
| | | | | 4 | 0.70 | |
| | | | | 5 | 0.53 | |
| Cout-weighted Combination | 0.2 | 0.2 | 0.6 | 1 | 0.25 | Focuses on layers with large output channel counts |
| | | | | 2 | 0.72 | |
| | | | | 3 | 0.86 | |
| | | | | 4 | **0.90** | |
| | | | | 5 | 0.67 | |

TABLE 3.2: Suitability Score Coefficient Configurations for Different Optimization Scenarios

In the **output channel-focused scenario**, which focuses only on the output channel, it is observed that the $S_{\text{score}}$ of layers with a high number of output channels is higher. However, it is found that this approach provides advantages only in certain types of layers and is generally not as decisive as kernel size and computational load.

When more complex scenarios are examined, the **balanced combination scenario** gives equal weight to kernel size and MAC factors and provides more stable results for large-scale models. This combination provides reasonable $S_{\text{score}}$ for layers with small kernel sizes and layers with high computational load.

Conversely, in the **kernel-weighted combination scenario**, which prioritises small kernel sizes but also partially considers the number of MACs and output channels, it is seen that the $S_{\text{score}}$ of layers with small kernel sizes increases, but other factors contribute to a limited extent.

In the **MACs-weighted Combination scenario**, which prioritises layers with more computational load, it was found that the suitability score increased in layers that required intensive computation. This structure reveals the advantage of the Winograd algorithm in that it is more effective in layers with high computational load.

Finally, in the **Cout-weighted Combination scenario**, it was seen that the suitability score increased in layers with a high number of output channels. However, the general evaluation found that the effect of combinations focusing only on the number of output channels on the suitability score was not as decisive as other factors.

The analysis results generally show that small kernel sizes and layers with high MACs benefit more from the Winograd algorithm. It is observed that mixed coefficient combinations, such as balanced combinations, provide more stable and balanced results in large-scale models. However, depending on the design goals, combinations that prioritise only one factor may be more efficient for specific layers. These findings emphasise the importance of carefully selecting coefficient combinations to increase the applicability of the Winograd algorithm.

#### 3.2.3.4 Impact of Filter Compatibility Factor ($\alpha$) on Suitability Score

The impact of $\alpha$ on $S_{\text{score}}$ is evaluated for both the AlexNet and VGG16 architectures, as depicted in Figures 3.1 and 3.2.

**AlexNet Analysis** For AlexNet, it is observed that increasing the $\alpha$ coefficient results in a notable rise in the Suitability Score, particularly from the second convolutional layer (Conv2) onwards. The initial layer (Conv1) exhibits only a modest increase in $S_{\text{score}}$ as $\alpha$ increases, indicating that kernel size is not the dominant factor at the input

stage, where the kernel size is relatively large (11×11). A more pronounced improvement is evident in Conv2 to Conv4, where kernel sizes reduce to 5×5 and 3×3. This behaviour confirms that smaller kernel sizes are more compatible with the strengths of the Winograd algorithm and that $\alpha$ plays a critical role in quantifying this alignment. As $\alpha$ approaches higher values (e.g., 0.7–0.8), the Suitability Scores for Conv3 and Conv4 approach 0.9, suggesting highly favourable conditions for Winograd-based optimisation.

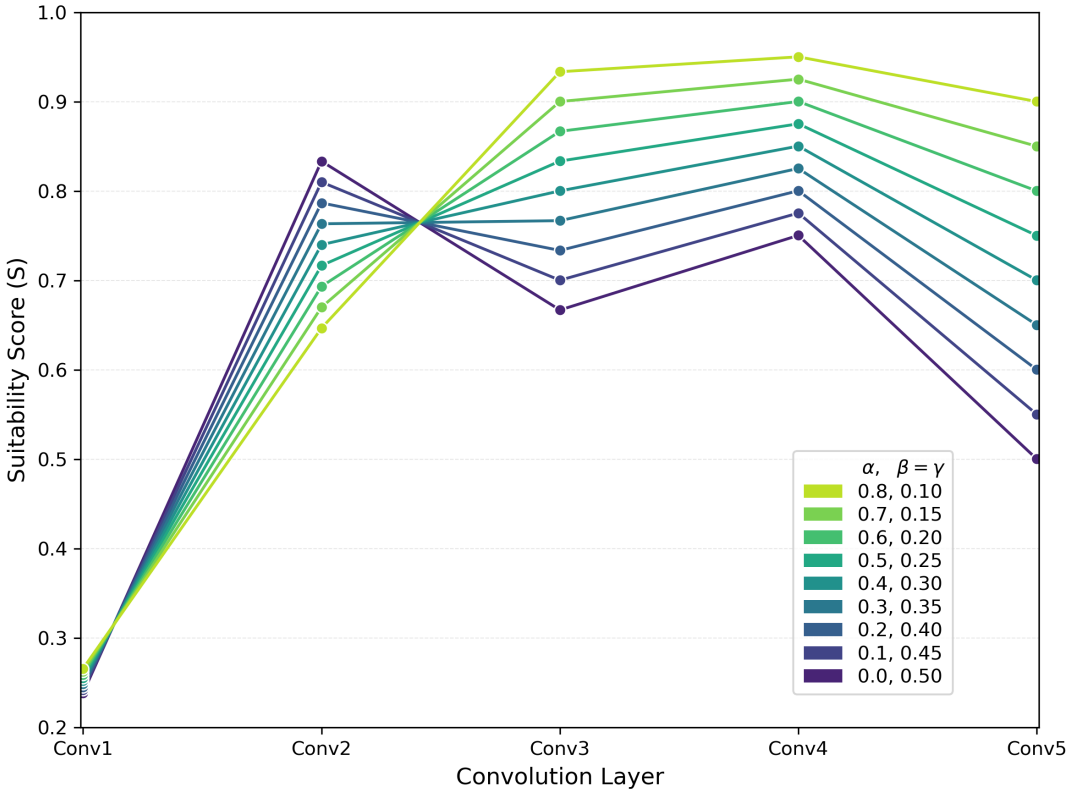



FIGURE 3.1: Impact of the filter compatibility factor ($\alpha$) on Suitability Score ($S_{\text{score}}$) across the convolutional layers of AlexNet.

**VGG16 Analysis** In the case of VGG16, the effect of $\alpha$ is even more significant and consistent across all convolutional layers. Due to the uniform use of $3\times3$ kernels throughout the architecture, the Suitability Score steadily increases with rising $\alpha$ values. When $\alpha$ is set to 1.0, nearly all layers achieve Suitability Scores close to 1.0, indicating optimal compatibility with the Winograd algorithm. The sharp increase in $S_{\text{score}}$, particularly between $\alpha$ values of 0.5 and 1.0, highlights the dominant influence of kernel size compatibility in architectures where kernel dimensions are homogeneous and small. Compared to AlexNet, VGG16 exhibits a more uniform and stable Suitability Score progression, indicating that its architectural design is inherently more favourable for Winograd-based optimisation when $\alpha$ is prioritised.

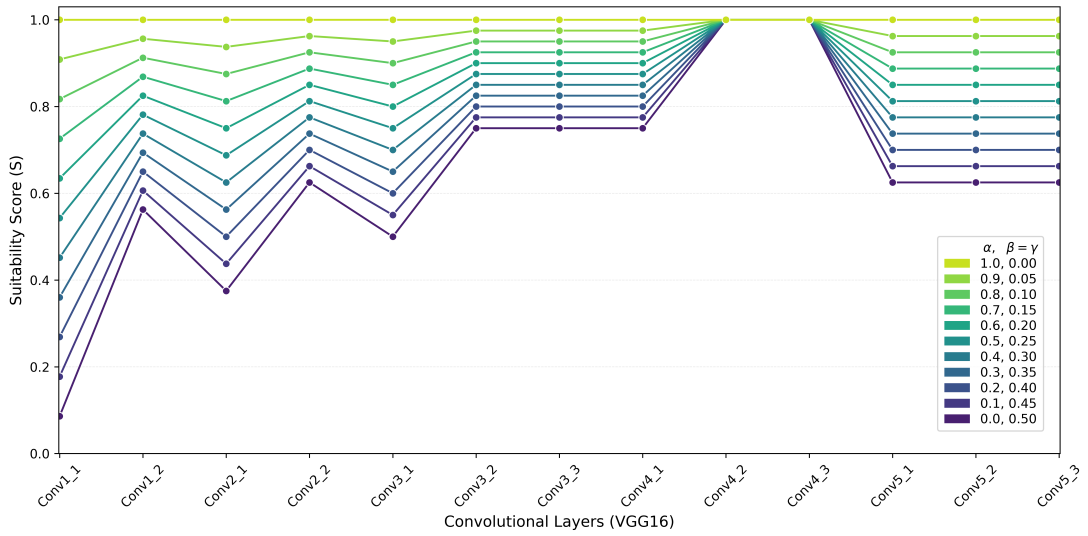



FIGURE 3.2: Impact of the filter compatibility factor ($\alpha$) on Suitability Score ($S_{\text{score}}$) across the convolutional layers of VGG16.

**Comparative Insights** Although both AlexNet and VGG16 demonstrate a positive correlation between $\alpha$ and the Suitability Score, the effect is notably stronger and more consistent in VGG16 due to its uniform small kernel sizes. In AlexNet, the variability in kernel sizes across different layers results in a more heterogeneous increase in $S_{\text{score}}$ as $\alpha$ increases. These findings suggest that networks featuring consistent and smaller kernel configurations can benefit more predictably from Winograd-based optimisation strategies when filter compatibility is emphasised.

#### 3.2.3.5 Impact of Computational Efficiency Factor ($\beta$) on Suitability Score

The impact of $\beta$ on $S_{\text{score}}$ is evaluated for both the AlexNet and VGG16 architectures, as shown in Figures 3.3 and 3.4.

**AlexNet Analysis** In the case of AlexNet, increasing the $\beta$ coefficient leads to a significant rise in the Suitability Score for layers with a higher computational load. This effect is particularly evident at Conv2, where a large number of MAC operations is required. As $\beta$ increases towards 1.0, the Suitability Score for Conv2 reaches values close to 1.0, indicating a strong alignment with the benefits offered by the Winograd algorithm. In contrast, layers with lower computational loads, such as Conv1 and Conv5, exhibit more modest improvements in their Suitability Scores. This trend suggests that prioritising computational efficiency heavily influences layers with higher MAC values, enhancing their compatibility with the Winograd transformation.

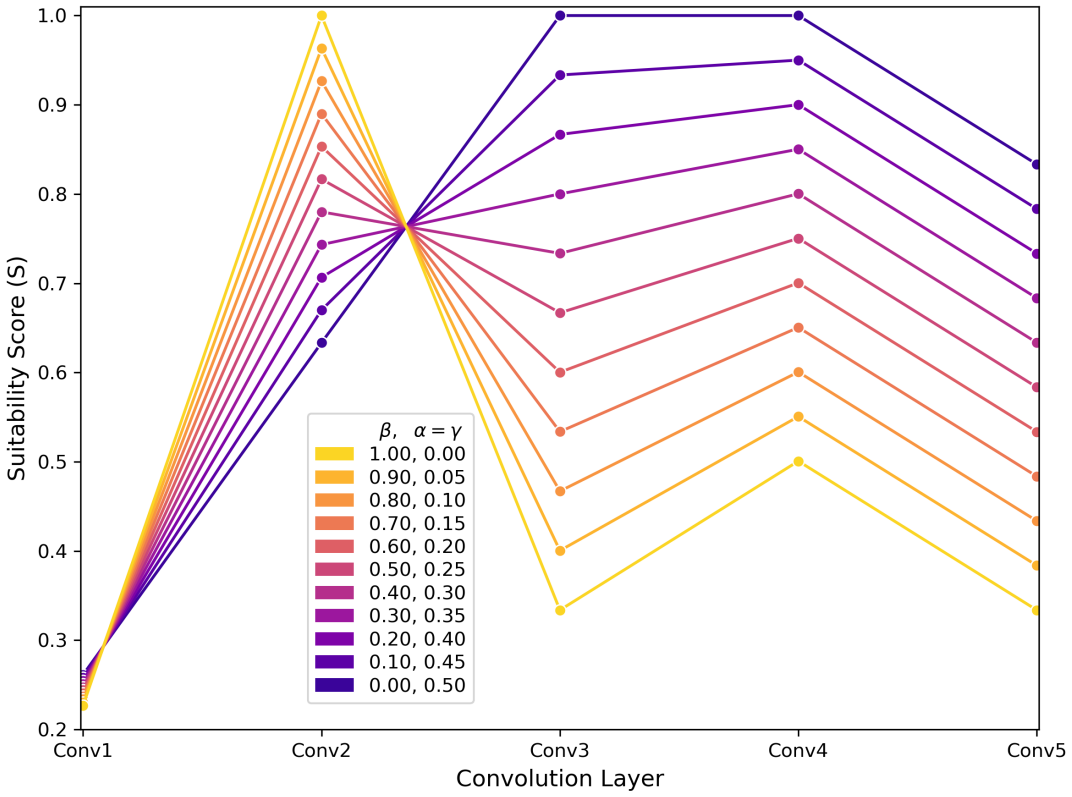



FIGURE 3.3: Impact of the computational efficiency factor ($\beta$) on Suitability Score ($S_{\text{score}}$) across the convolutional layers of AlexNet.

**VGG16 Analysis** For VGG16, the effect of $\beta$ is even more pronounced across deeper layers. Layers such as Conv2_1, Conv3_1, and Conv4_1, which require higher computational effort, show substantial increases in Suitability Score as $\beta$ rises. When $\beta$ approaches 1.0, these layers achieve Suitability Scores nearing 1.0, whereas layers with relatively lower computational demands maintain moderate scores. Compared to AlexNet, VGG16 exhibits a more dynamic response to $\beta$ variations due to its deeper and more complex architecture. The analysis highlights that layers demanding greater computational resources are more amenable to optimisation via the Winograd method when $\beta$ is emphasised.

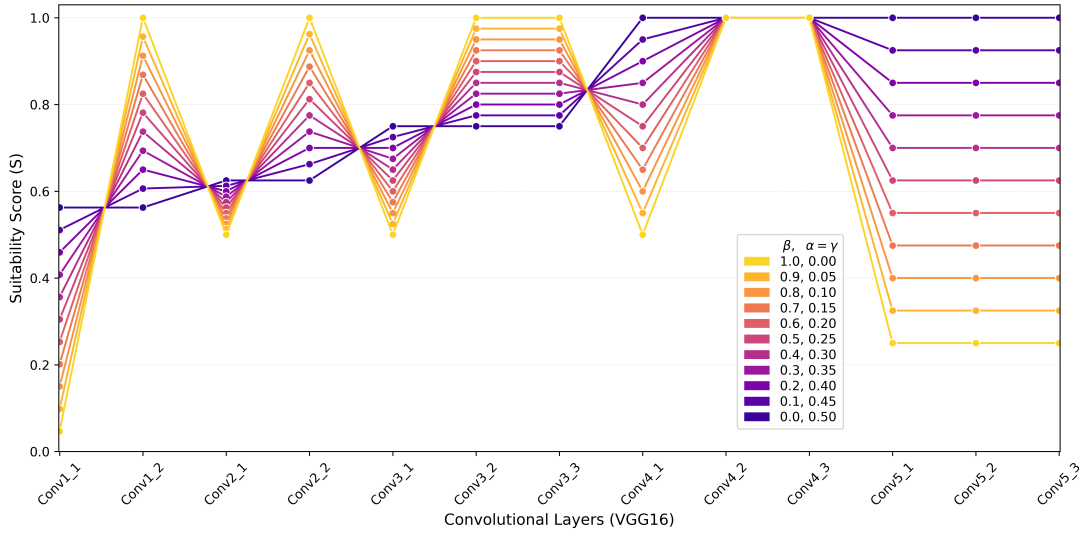



FIGURE 3.4: Impact of the computational efficiency factor ($\beta$) on Suitability Score ($S_{\text{score}}$) across the convolutional layers of VGG16.

**Comparative Insights** Both AlexNet and VGG16 confirm the critical role of $\beta$ in optimising layers with high computational loads. However, the response in VGG16 is more differentiated across layers due to its deeper structure. AlexNet exhibits a dominant peak around Conv2, whereas VGG16 displays multiple peaks across its deeper stages, reflecting the broader distribution of computational intensity.

#### 3.2.3.6 Impact of Feature Complexity Factor ($\gamma$) on Suitability Score

The impact of $\gamma$ on $S_{\text{score}}$ is analysed for both the AlexNet and VGG16 architectures, as presented in Figures 3.5 and 3.6.

**AlexNet Analysis** For AlexNet, increasing the $\gamma$ coefficient results in a slight but consistent rise in the Suitability Score for layers with a higher number of output channels. However, the overall impact of $\gamma$ is less pronounced compared to the effects observed with $\alpha$ and $\beta$. The changes in $S_{\text{score}}$ are more evident in Conv3 and Conv4, where the output channel counts are relatively higher. Nonetheless, the variation in $S_{\text{score}}$ across the layers remains limited, suggesting that feature complexity, although a relevant factor, plays a secondary role in determining Winograd applicability in this architecture.

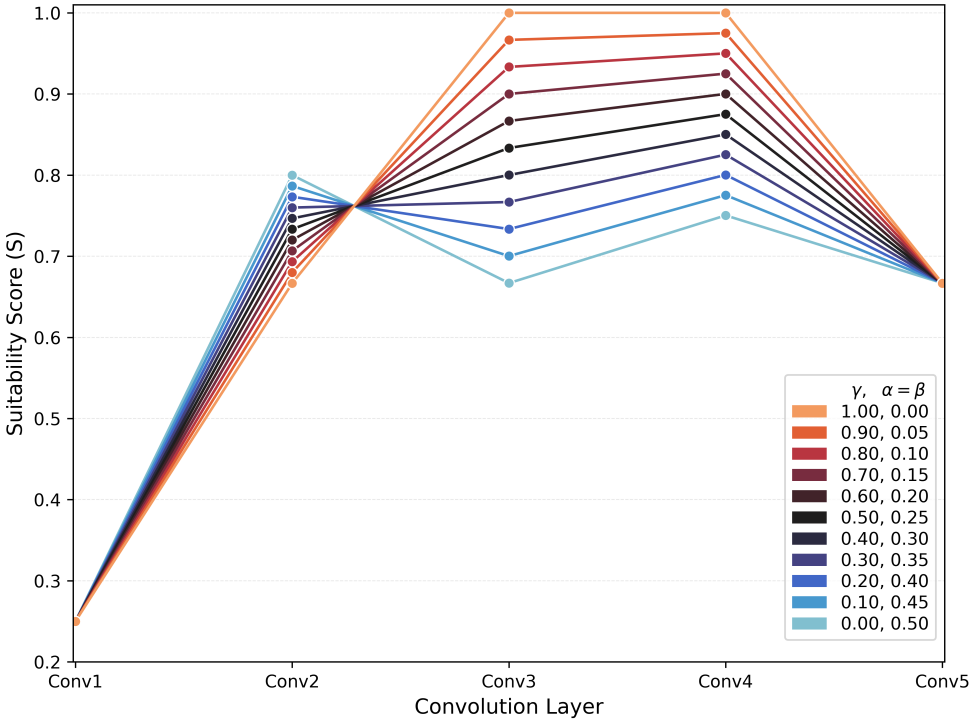



FIGURE 3.5: Impact of the feature complexity factor ($\gamma$) on Suitability Score ($S_{\text{score}}$) across the convolutional layers of AlexNet.

**VGG16 Analysis** In the VGG16 model, a similar trend is observed. As $\gamma$ increases, layers with higher output channel counts show a moderate increase in Suitability Scores. Layers such as Conv2_1, Conv3_1, and Conv4_1, which have higher numbers of output channels, display the most noticeable improvements. However, compared to the influences of $\alpha$ and $\beta$, the effect of $\gamma$ remains relatively modest. The Suitability Scores increase gradually but do not reach the high levels observed when $\alpha$ or $\beta$ are prioritised.

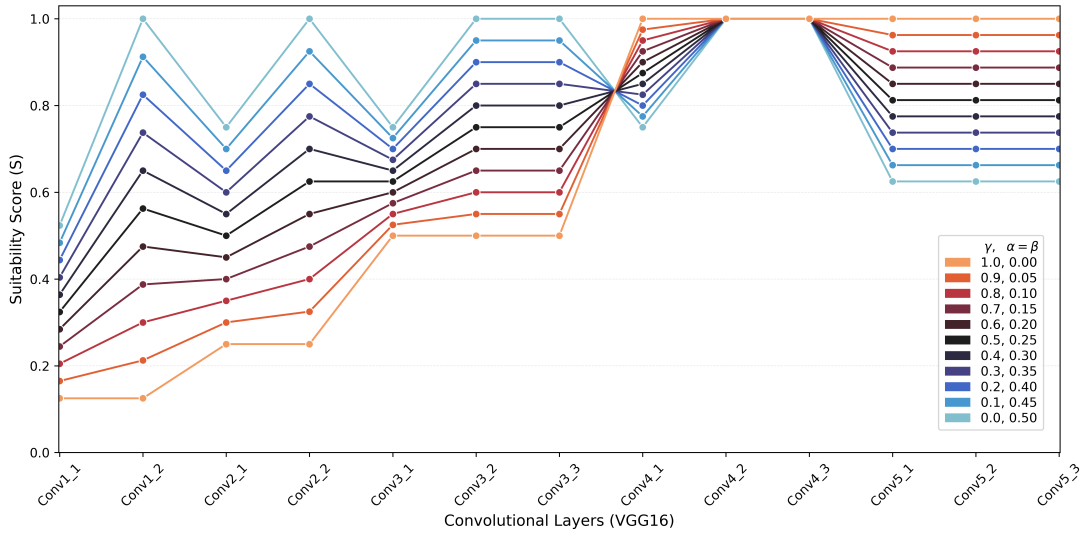



FIGURE 3.6: Impact of the feature complexity factor ($\gamma$) on Suitability Score ($S_{\text{score}}$) across the convolutional layers of VGG16.

**Comparative Insights** While both architectures demonstrate an increase in Suitability Score with rising $\gamma$, the degree of improvement is limited. Compared to $\alpha$ and $\beta$, $\gamma$ exerts a weaker influence on the Winograd compatibility of the layers. Moreover, the effect of $\gamma$ appears to be more consistent in AlexNet than in VGG16, where deeper layers exhibit more diverse responses due to complex feature hierarchies.

#### 3.2.3.7 Conclusion on the Impact of Coefficient Factors

The comprehensive analysis of the three key coefficient factors—filter compatibility ($\alpha$), computational efficiency ($\beta$), and feature complexity ($\gamma$)—reveals distinct influences on the Suitability Score ($S_{\text{score}}$) across different convolutional architectures.

The filter compatibility factor ($\alpha$) is identified as the most decisive parameter, enhancing the Suitability Score for layers with smaller kernel sizes. This effect is particularly pronounced in architectures such as VGG16, where consistent 3×3 kernel configurations allow for near-optimal Winograd applicability as $\alpha$ increases.

The computational efficiency factor ($\beta$) also demonstrates a strong influence, especially in layers characterised by high computational loads. In both AlexNet and VGG16, increasing $\beta$ leads to substantial improvements in the Suitability Scores of deeper or more computationally intensive layers, underlining the importance of targeting computational demands for Winograd optimisation.

Conversely, the feature complexity factor ($\gamma$) exhibits a more limited impact. Although increasing $\gamma$ results in slight improvements in Suitability Scores for layers with higher output channel counts, its influence remains secondary relative to $\alpha$ and $\beta$. The findings

suggest that while output feature complexity contributes to the optimisation landscape, it should not be prioritised at the expense of kernel size and computational considerations.

Overall, the results demonstrate that careful and context-specific adjustment of $\alpha$, $\beta$, and $\gamma$ is essential for maximising the effectiveness of the Winograd algorithm across diverse convolutional neural network architectures. Strategic weighting of these factors can enhance computational efficiency and model performance during the optimisation process.

## 3.3 Optimization Steps Using Suitability Score

The overall optimization procedure guided by the suitability score is illustrated in Fig. 3.7. This flowchart provides a high-level overview of the iterative process, where convolutional layers are categorized, adjusted, and evaluated until the defined constraints on accuracy, computational load, and training time are satisfied. The subsequent subsections present each stage of this procedure in detail.

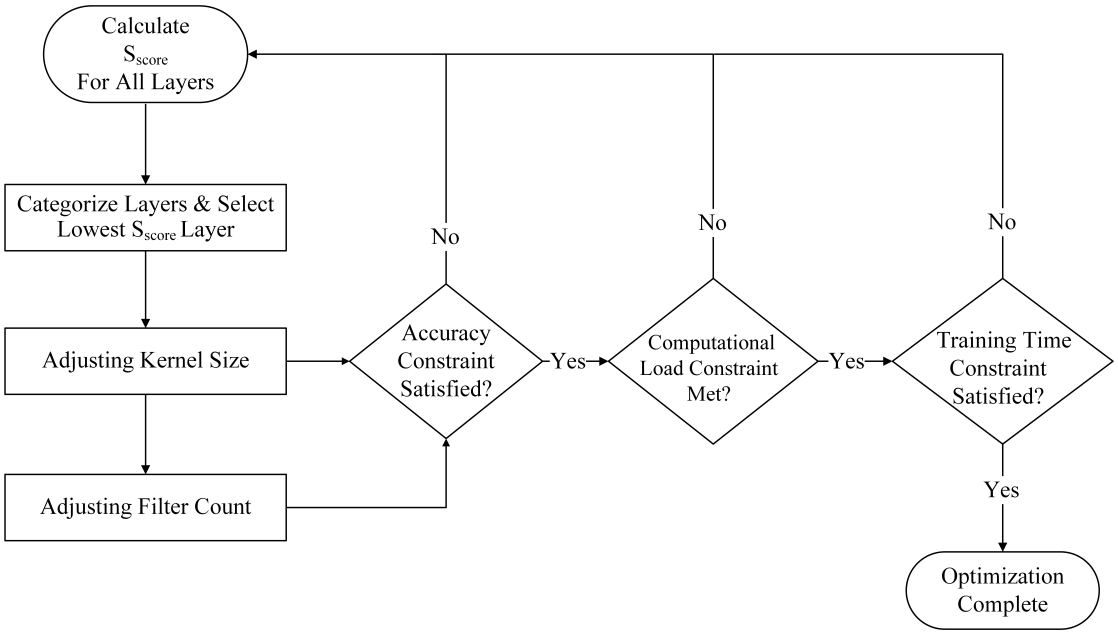



FIGURE 3.7: Optimization steps using $S_{\text{score}}$ for convolutional layers

### 3.3.1 Categorizing Convolutional Layers Based on Suitability Score

The Suitability Score provides a systematic method for evaluating and optimising convolutional layers for computational efficiency. Therefore, it can be considered an indicator of which layers require potential optimisation or which can maximise computational efficiency. $S_{\text{score}}$ can take values between 0 and 1, as shown in $S_{\text{score}} \in [0, 1]$.

Different score ranges reflect varying degrees of compatibility with the Winograd algorithm, directly influencing the layer's computational efficiency and optimisation potential. Therefore, understanding what each score range signifies is essential for effectively applying the Winograd algorithm.

In the following subsections, the suitability score thresholds are categorized into low, medium, and high ranges, each explained through mathematical justification. In determining the threshold values for $S_{\text{score}}$, balanced combination coefficiencies ($\alpha = 0.4, \beta = 0.4, \gamma = 0.2$) are preferred. This approach gives equal weight to kernel size and computational load while also considering the number of output channels to a limited extent. Thus, a more consistent and generally valid fitness score is obtained, especially in large-scale models.

#### 3.3.1.1 Low Suitability Score

A Low Suitability Score, defined as a value between 0 and 0.4, indicates that the layer is less compatible with the Winograd algorithm compared to other layers. In such cases, unless substantial modifications are made to the structure or parameters of the layer, the application of the Winograd algorithm is likely to have only a minimal impact.

The upper bound of this range, $S_{\text{score}} = 0.4$, is derived from the balanced coefficient configuration. This value corresponds to scenarios where only a single factor contributes fully to the suitability score while the others have no influence. For example, if only the kernel size is favorable (i.e., $K_{\text{ref}}/K = 1$) and the other factors are not (i.e., both ratios are zero), the score is calculated as:

$$S_{\text{score}} = \alpha \cdot 1 + \beta \cdot 0 + \gamma \cdot 0 = 0.4 \tag{3.6}$$

Similarly, if only the computational cost is favorable (i.e., $\text{MACs}/\text{MACs}_{\text{ref}} = 1$), and the other factors contribute nothing, the result remains:

$$S_{\text{score}} = \alpha \cdot 0 + \beta \cdot 1 + \gamma \cdot 0 = 0.4 \tag{3.7}$$

Considering equal contribution from the kernel size and MAC operations, and disregarding the effect of output channels, the threshold value is computed as:

$$S_{\text{score}} = \alpha \cdot 0.5 + \beta \cdot 0.5 + \gamma \cdot 0 = 0.4 \tag{3.8}$$

This exclusion of the $\gamma$ in specific threshold calculations stems from its relatively lower influence on the applicability of the Winograd algorithm when compared to kernel size ($\alpha$) and computational cost ($\beta$).

#### 3.3.1.2 Medium Suitability Score

A Medium Suitability Score, defined as a value greater than 0.4 and less than 0.7, indicates that the layer demonstrates moderate compatibility with the Winograd algorithm. Nevertheless, further optimisation of the layer's structure or parameters may be required in order to leverage the potential benefits offered by the algorithm fully.

To support this interval mathematically, two illustrative scenarios are considered. In the first scenario, the kernel size is ideal ($K_{\text{ref}}/K = 1$), while the MAC and output channel contributions are moderate ($= 0.5$). In the second case, the computational complexity is ideal ($\text{MACs}/\text{MACs}_{\text{ref}} = 1$), with the kernel and output channel values being moderate. Assuming a balanced coefficient configuration ($\alpha = 0.4, \beta = 0.4, \gamma = 0.2$), both cases yield a suitability score of:

$$S_{\text{score}} = \alpha \cdot 1 + \beta \cdot 0.5 + \gamma \cdot 0.5 = 0.7 \tag{3.9}$$

$$S_{\text{score}} = \alpha \cdot 0.5 + \beta \cdot 1 + \gamma \cdot 0.5 = 0.7 \tag{3.10}$$

These examples support the idea that medium scores emerge when one parameter strongly aligns with the Winograd characteristics, and the others offer partial support. Such layers may benefit from further optimisation but already show a reasonable level of compatibility.

#### 3.3.1.3 High Suitability Score

A High Suitability Score, defined as a value between 0.7 and 1, implies that the layer is already well aligned with the advantages offered by the Winograd transformation and therefore requires minimal or no further optimisation. In such cases, the layer is considered highly suitable for the direct application of the Winograd algorithm, thereby enabling immediate performance improvements.

This score level typically emerges when multiple factors—kernel size, computational load, and feature complexity—jointly contribute to the suitability. For instance, in a balanced configuration with $\alpha = 0.4$, $\beta = 0.4$, and $\gamma = 0.2$, if all the ratios are optimal (i.e., each equals 1), the score is calculated as:

$$S_{\text{score}} = \alpha \cdot 1 + \beta \cdot 1 + \gamma \cdot 1 = 1 \tag{3.11}$$

Such a result signifies that the layer meets all conditions for Winograd optimization and can be directly utilized for acceleration without needing structural adjustments. These layers are ideal targets for immediate algorithm application in performance-critical systems.

In summary, a high Suitability Score means that the layer can directly benefit from the Winograd algorithm without requiring further modification. Conversely, a low score indicates that the layer needs additional optimization to further benefit from the algorithm. In addition, the value ranges defined for the Suitability Score are based on the **balanced coefficient combination** scenario proposed in this study. These thresholds aim to offer a general categorization framework for layer suitability. However, the coefficient weights can be adjusted depending on the design priorities, such as minimizing latency, reducing memory bandwidth, or maximizing speed-up. Therefore, the classification ranges for $S_{\text{score}}$ can be redefined accordingly.

### 3.3.2 Adjusting Kernel Size

Low $S_{\text{score}}$ in convolutional layers is usually due to the kernel size not being configured with a 3x3 kernel size, which is well compatible with the strengths of the Winograd algorithm. Therefore, the first step to optimize layers with low $S_{\text{score}}$ is to adjust the kernel size it is set according in (3.12). If the kernel size exceeds 5, it is set to 5 or 3.

$$K = \begin{cases} 5, & \text{if } K > 5 \\ 3, & \textit{otherwise} \end{cases} \tag{3.12}$$

Since Winograd works more efficiently with a 3x3 kernel, a 3x3 kernel is recommended. However, kernel size adjustment can be made according to the project's purpose. The purpose of adjusting kernel size is to increase the compatibility of the CNN structure with the Winograd algorithm. However, decreasing kernel size causes $C_{\text{out}}$ to increase. In this case, in the following layers, due to the change in $C_{\text{out}}$, MACs may increase. In this case, the number of kernels will be adjusted for the next steps.

Also, after adjusting the kernel size for the specified layers, the modified model needs to be retrained and evaluated to measure its performance. This iterative process allows us to evaluate kernel size's impact on accuracy and computational efficiency.

### 3.3.3 Adjusting Filter Count

After adjusting the kernel size for all convolutional layers, the next step focuses on increasing the $S_{\text{score}}$ value of low-scoring layers. Since layers with high MACs and $C_{\text{out}}$ influence the overall computational load, adjusting these layers becomes essential for achieving balanced resource utilization in CNN architectures. Since these adjustments impact multiple layers, careful evaluation is necessary to avoid unintended degradation in performance. In the filter count adjustment step, since $C_{\text{ref}}$ represents the maximum number of filters in a layer, the optimization first starts by reducing the number of filters in layers with high $C_{\text{ref}}$ values. The new reference filter count is assigned to the second-highest filter count in the network. This adjustment is defined mathematically as in (3.13).

$$C_{new\_ref} = \max\{x \mid x \in C \text{ and } x < C_{ref}\} \tag{3.13}$$

Where:

- $C_{new\_ref}$ represents the new reference filter count after adjustment.
- $C$ denotes the set of all filter counts across convolutional layers.
- $C_{ref}$ is the original reference filter count, representing the maximum filter count in the set.
- $\max\{\cdot\}$ selects the maximum value within the defined set.
- The condition $x < C_{ref}$ ensures that only values smaller than the original reference filter count are considered.

For example, if the filter counts are $\{64, 128, 256, 512, 512\}$ and $C_{ref} = 512$, the adjusted reference filter count will be 256. Consequently, the layers that originally had 512 filters will now be updated to have 256 filters. This update will change the MACs for these layers. Therefore, the next step should focus on the layer with the highest MACs.

After calculating the MACs for each layer, the next step involves adjusting the filter count of the layer with the highest MACs. Adjusting the number of filters impacts the MACs calculation, thereby effectively decreasing the computational load. For this purpose, the filter count of the layer with the highest MACs is adjusted to match the second-highest MACs. Since MACs are proportional to the filter count, this adjustment is performed by scaling the filter count accordingly, as in (3.14).

$$C_{out\_new} = C_{out\_old} \times \frac{\text{Target MACs}}{\text{Initial MACs}} \tag{3.14}$$

Where:

- $C_{out\_new}$ represents the updated filter count for the layer with the highest MACs.
- $C_{out\_old}$ is the original filter count for the same layer.
- Target MACs corresponds to the second highest MACs value in the CNN model.
- Initial MACs refers to the original MACs value of the layer with the highest MACs.

By reducing the filter count in this manner, the MACs of the layer with the highest computational demand are aligned with the second-highest MACs. This adjustment helps balance the network's overall computational load, preventing any single layer from dominating the total MACs. However, this may cause the $C_{ref}$ value to change again. To prevent excessive accuracy loss, the adjustment process should adopt an iterative approach with continuous evaluation Therefore, after each kernel amount changes, the $S_{score}$ score is recalculated, and the kernel number is readjusted according to $C_{ref}$ and $MACs_{ref}$.

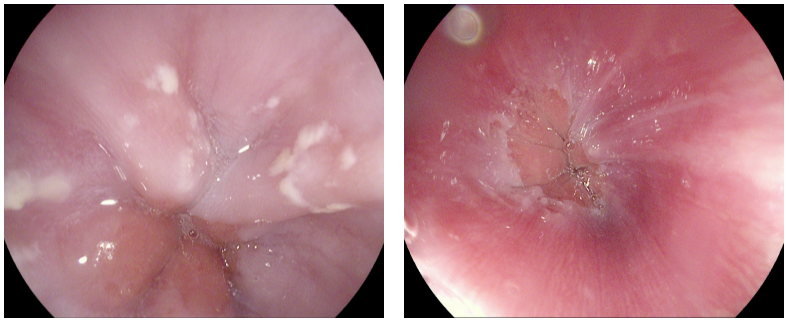
It is important to note that reducing the filter count may impact the model's learning capacity and performance. Therefore, such adjustments should be carefully assessed to ensure that computational gains are achieved without compromising model accuracy. After each modification, the model must be retrained and reevaluated to ensure that the adjustments improve both computational efficiency and overall accuracy. This iterative approach ensures that all layers are progressively optimised while balancing computational demand and model performance. Finally, optimisation steps continue until the model starts to fall below the expected accuracy percentage. This approach ensures that computational gains are maximised while preserving model accuracy during the entire optimisation process. When the optimization is completed, it will be observed that the low $S_{score}$ in the layers increase compared to their initial values. Thus, the layers will become more compatible with the Winograd method.

## 3.4 Model Optimisation and Performance Evaluation

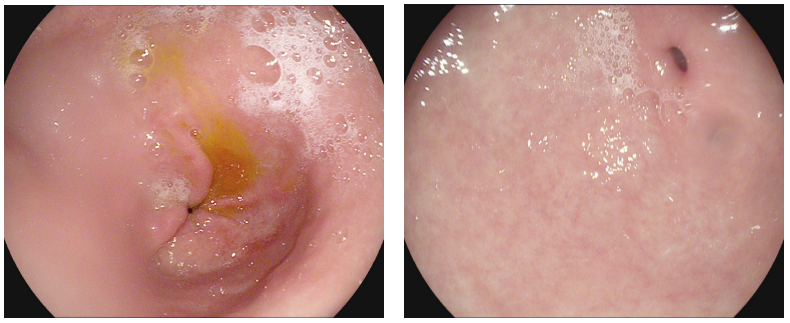
Experiments conducted on two structurally distinct CNN architectures, Alexnet and VGG16, assess the effectiveness and generalisability of the proposed suitability score-based optimisation approach. These models were selected due to their contrasting features. AlexNet contains fewer convolutional layers with various kernel sizes (11×11, 5×5, and 3×3). Furthermore, VGG16 has a deeper architecture consisting of multiple convolutional layers with uniform 3×3 kernels. This contrast allows us to evaluate the adaptability of the optimisation strategy across different architectural patterns. The following sections present the experimental setup and detailed performance evaluations.

### 3.4.1 Experimental Settings

The effectiveness of the proposed optimisation approach is evaluated using a publicly available oesophageal endoscopy image dataset [118]. The dataset contains 11662 RGB images; however, for computational feasibility and to facilitate the iterative optimisation procedure, a subset of 800 images was randomly selected. This subset comprises 400 oesophageal cancer images and 400 healthy images, ensuring class balance.



(A) Diseased (B) Diseased



(C) Healthy (D) Healthy

FIGURE 3.8: Diseased and healthy endoscopy images

The images in the dataset have resolutions that vary between 1012 and 1221 pixels. All images were resized and preprocessed before being fed into the CNN models. Figure 3.8 displays sample images from both diseased and healthy classes.

### 3.4.2 Performance Evaluation of Iterative Refinement in AlexNet

Table 3.3 illustrates the iterative optimisation process applied to the AlexNet architecture using the proposed Suitability Score. Initially, the default AlexNet configuration employs convolutional layers with varying kernel sizes (11×11, 5×5, and 3×3) and a large stride of 4×4 in the first layer. This setup results in a total MAC count of approximately $10.72 \times 10^8$, an epoch time of 585.28 seconds, and a test accuracy of 84.40%.

In the first iteration, the kernel sizes of the first and second layers were reduced to enhance compatibility with the Winograd algorithm. However, the stride in the first layer was retained as 4×4. This mismatch between kernel and stride negatively impacted

model performance, reducing accuracy to 68.12% and F1 score to 63.18%, despite slight improvements in computational cost.

To address this issue, the second iteration harmonised both kernel size and stride, setting all kernels to 3×3 and all strides to 1×1. While this alignment improved Winograd's suitability, it introduced a substantial computational overhead. The MAC count increased to $126.48 \times 10^8$, and the epoch time rose to 3668.97 seconds representing increases of more than 11.79× and 6.26×, respectively, compared to the default model. Furthermore, accuracy and F1 score dropped to 56.52% and 45.19%, indicating poor performance despite increased complexity.

From iterations 3 through 6, the number of filters in high-complexity layers was gradually reduced. This led to a progressive recovery in model performance while reducing MACs. For example, iteration 5 has an epoch time of 1476.86 seconds and improved accuracy to 73.75%. Suitability scores across layers also increased, indicating better alignment with the Winograd method.

Iterations 7 and 8 involved more aggressive compression. The number of filters was reduced notably in later layers, leading to a compact network with total MACs as low as $0.86 \times 10^8$ in iteration 8. Remarkably, despite this drastic reduction in computational load, iteration 8 achieved the highest recorded accuracy of 97.50% and F1 score of 97.50%, with an epoch time of only 350.47 seconds. Compared to the default AlexNet, this reflects a ∼12.46× reduction in MACs and 1.67× reduction in training time while also enhancing predictive performance.

| CNN Model | Conv. Layer | Kernel Size | Filters | Stride | MACs ($10^8$) | $S_{score}$ | Accuracy (%) | F1 Score (%) | Epoch Time (s) |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 11x11 | 96 | 4x4 | 1.02 | 0.24 | | | |
| | 2 | 5x5 | 256 | 1x1 | 4.48 | 0.77 | | | |
| AlexNet | 3 | 3x3 | 384 | 1x1 | 1.49 | 0.73 | 84.37 | 84.40 | 585.28 |
| | 4 | 3x3 | 384 | 1x1 | 2.24 | 0.80 | | | |
| | 5 | 3x3 | 256 | 1x1 | 1.49 | 0.66 | | | |
| | 1 | **5x5** | 96 | 4x4 | 0.22 | 0.32 | | | |
| | 2 | 3x3 | 256 | 1x1 | 1.86 | 0.82 | | | |
| Iteration 1 | 3 | 3x3 | 384 | 1x1 | 1.73 | 0.87 | 68.12 | 63.18 | 495.48 |
| | 4 | 3x3 | 384 | 1x1 | 2.60 | 1.00 | | | |
| | 5 | 3x3 | 256 | 1x1 | 1.73 | 0.80 | | | |
| | 1 | 3x3 | **48** | 1x1 | 1.28 | 0.46 | | | |
| | 2 | 3x3 | 256 | 1x1 | 28.20 | 0.80 | | | |
| Iteration 2 | 3 | 3x3 | 384 | 1x1 | 27.70 | 0.87 | 55.62 | 45.19 | 3668.97 |
| | 4 | 3x3 | 384 | 1x1 | 41.60 | 1.00 | | | |
| | 5 | 3x3 | 256 | 1x1 | 27.70 | 0.80 | | | |
| | 1 | 3x3 | 48 | 1x1 | 1.28 | 0.47 | | | |
| | 2 | 3x3 | 256 | 1x1 | 28.20 | 0.93 | | | |
| Iteration 3 | 3 | 3x3 | 384 | 1x1 | 27.70 | 0.99 | 51.24 | 36.05 | 3049.09 |
| | 4 | 3x3 | **96** | 1x1 | 10.40 | 0.60 | | | |
| | 5 | 3x3 | 256 | 1x1 | 6.94 | 0.63 | | | |
| | 1 | 3x3 | 48 | 1x1 | 1.28 | 0.49 | | | |
| | 2 | 3x3 | 256 | 1x1 | 28.20 | 1.00 | | | |
| Iteration 4 | 3 | 3x3 | **256** | 1x1 | 18.50 | 0.86 | 60.00 | 52.70 | 2783.16 |
| | 4 | 3x3 | 96 | 1x1 | 6.94 | 0.57 | | | |
| | 5 | 3x3 | **96** | 1x1 | 6.94 | 0.70 | | | |
| | 1 | 3x3 | 48 | 1x1 | 1.28 | 0.67 | | | |
| | 2 | 3x3 | **64** | 1x1 | 7.06 | 0.93 | | | |
| Iteration 5 | 3 | 3x3 | **64** | 1x1 | 1.16 | 0.60 | 73.75 | 68.12 | 1476.86 |
| | 4 | 3x3 | 96 | 1x1 | 1.73 | 0.70 | | | |
| | 5 | 3x3 | **64** | 1x1 | 1.73 | 0.63 | | | |
| | 1 | 3x3 | **24** | 1x1 | 0.32 | 0.76 | | | |
| | 2 | 3x3 | **16** | 1x1 | 0.44 | 0.85 | | | |
| Iteration 6 | 3 | 3x3 | 64 | 1x1 | 0.29 | 0.86 | 94.99 | 94.98 | 465.49 |
| | 4 | 3x3 | **24** | 1x1 | 0.43 | 0.87 | | | |
| | 5 | 3x3 | 64 | 1x1 | 0.43 | 0.99 | | | |
| | 1 | 3x3 | **12** | 1x1 | 0.16 | 0.66 | | | |
| | 2 | 3x3 | 16 | 1x1 | 0.22 | 0.76 | | | |
| Iteration 7 | 3 | 3x3 | 64 | 1x1 | 0.29 | 1.00 | 97.25 | 94.66 | 398.63 |
| | 4 | 3x3 | **12** | 1x1 | 0.22 | 0.74 | | | |
| | 5 | 3x3 | 64 | 1x1 | 0.22 | 0.90 | | | |
| | 1 | 3x3 | 10 | 1x1 | 0.13 | 0.67 | | | |
| | 2 | 3x3 | **12** | 1x1 | 0.22 | 0.90 | | | |
| Iteration 8 | 3 | 3x3 | **40** | 1x1 | 0.23 | 0.98 | 97.50 | 97.50 | 350.47 |
| | 4 | 3x3 | 12 | 1x1 | 0.14 | 0.70 | | | |
| | 5 | 3x3 | **40** | 1x1 | 0.14 | 0.83 | | | |

TABLE 3.3: Iterative improvement of AlexNet MACs, $S_{score}$, and Performance Metrics (with $\alpha = 0.4$, $\beta = 0.4$, $\gamma = 0.2$)

### 3.4.3 Performance Evaluation of Iterative Refinement in VGG16

Table 3.4 presents the effects of the optimisation steps applied to the VGG16 model in detail. In the initial configuration, the suitability scores in many model layers remain below 0.7, indicating limited compatibility with the Winograd algorithm. Moreover, the initial model required approximately $153.5 \times 10^8$ MAC operations, with an epoch time of 3240.89 seconds.

In Iteration 1, adjustments to kernel sizes led to improved suitability scores in several layers. Both accuracy and the F1 score slightly increased, reaching 96.88%. From Iteration 2 through Iteration 6, a progressive reduction in the number of filters in layers with high MAC values yielded higher suitability scores. Following retraining, the model maintained its accuracy while achieving significant performance gains. For instance, the epoch time dropped to 501.39 seconds in Iteration 5 and to 363.88 seconds in Iteration 6, achieving speedups of 6.5× and 8.9×, respectively, compared to the default model.

Beyond Iteration 7, more aggressive compression continued to increase suitability scores, but slight decreases in accuracy and F1 score were observed, indicating a limit in how far compression could be applied without affecting performance.

In the final iteration, the total number of MAC operations was reduced from $153.5 \times 10^8$ to $1.90 \times 10^8$, corresponding to an approximate 80.91× reduction in computational complexity. Similarly, the epoch duration decreased from 3240.89 seconds to 155.10 seconds, yielding a 20.89× improvement in processing speed. Remarkably, this substantial reduction in computational cost was achieved without compromising accuracy; on the contrary, model accuracy improved from 95.63% to 97.50%.

In conclusion, the proposed iterative optimization strategy enhanced computational efficiency and Winograd compatibility while maintaining or enhancing the model's predictive performance.

| Model | Conv Layer | Filters | MACs ($10^8$) | $S_{\text{score}}$ | Accuracy (%) | F1 Score (%) | Epoch Time (s) |
|---|---|---|---|---|---|---|---|
| VGG16 | 1 | 64 | 0.86 | 0.44 | 95.63 | 95.63 | 3240.89 |
| | 2 | 64 | 18.5 | 0.83 | | | |
| | 3 | 128 | 9.25 | 0.65 | | | |
| | 4 | 128 | 18.5 | 0.85 | | | |
| | 5 | 256 | 9.25 | 0.70 | | | |
| | 6 | 256 | 18.5 | 0.90 | | | |
| | 7 | 256 | 18.5 | 0.90 | | | |
| | 8 | 512 | 9.25 | 0.80 | | | |
| | 9 | 512 | 18.5 | 1.00 | | | |
| | 10 | 512 | 18.5 | 1.00 | | | |
| | 11 | 512 | 4.62 | 0.70 | | | |
| | 12 | 512 | 4.62 | 0.70 | | | |
| | 13 | 512 | 4.62 | 0.70 | | | |
| Iteration 1 | 1 | 64 | 0.86 | 0.47 | 96.88 | 6.88 | 2622.53 |
| | 2 | 64 | 18.5 | 0.85 | | | |
| | 3 | 128 | 9.25 | 0.70 | | | |
| | 4 | 128 | 18.5 | 0.90 | | | |
| | 5 | 256 | 9.25 | 0.80 | | | |
| | 6 | 256 | 18.5 | 1.00 | | | |
| | 7 | 256 | 18.5 | 1.00 | | | |
| | 8 | **256** | 4.62 | 0.70 | | | |
| | 9 | **256** | 4.62 | 0.70 | | | |
| | 10 | **256** | 4.62 | 0.70 | | | |
| | 11 | **256** | 1.16 | 0.63 | | | |
| | 12 | **256** | 1.16 | 0.63 | | | |
| | 13 | **256** | 1.16 | 0.63 | | | |
| Iteration 2 | 1 | 64 | 0.86 | 0.49 | 96.25 | 96.25 | 1432.31 |
| | 2 | **32** | 9.25 | 0.83 | | | |
| | 3 | 128 | 4.62 | 0.70 | | | |
| | 4 | **64** | 9.25 | 0.85 | | | |
| | 5 | 256 | 4.62 | 0.80 | | | |
| | 6 | **128** | 9.25 | 0.90 | | | |
| | 7 | **128** | 4.62 | 0.70 | | | |
| | 8 | 256 | 2.31 | 0.70 | | | |
| | 9 | 256 | 4.62 | 0.80 | | | |
| | 10 | 256 | 4.62 | 0.80 | | | |
| | 11 | 256 | 1.16 | 0.65 | | | |
| | 12 | 256 | 1.16 | 0.65 | | | |
| | 13 | 256 | 1.16 | 0.65 | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 64 | 0.86 | 0.54 | | | |
| | 2 | 32 | 9.25 | 0.85 | | | |
| | 3 | 128 | 4.62 | 0.80 | | | |
| | 4 | 64 | 9.25 | 0.90 | | | |
| | 5 | 64 | 2.31 | 0.70 | | | |
| | 6 | 128 | 4.62 | 0.80 | | | |
| Iteration 3 | 7 | 128 | 4.62 | 1.00 | 94.38 | 94.37 | 1013.92 |
| | 8 | **128** | 1.16 | 0.65 | | | |
| | 9 | **128** | 1.16 | 0.65 | | | |
| | 10 | **128** | 1.16 | 0.65 | | | |
| | 11 | **128** | 0.29 | 0.61 | | | |
| | 12 | **128** | 0.29 | 0.61 | | | |
| | 13 | **128** | 0.29 | 0.61 | | | |
| | 1 | 64 | 0.86 | 0.58 | | | |
| | 2 | **16** | 0.46 | 0.83 | | | |
| | 3 | 128 | 2.31 | 0.80 | | | |
| | 4 | **32** | 0.46 | 0.85 | | | |
| | 5 | 128 | 1.16 | 0.70 | | | |
| | 6 | 128 | 4.62 | 1.00 | | | |
| Iteration 4 | 7 | 128 | 4.62 | 1.00 | 98.12 | 98.12 | 808.28 |
| | 8 | 128 | 1.16 | 0.70 | | | |
| | 9 | 128 | 1.16 | 0.70 | | | |
| | 10 | 128 | 1.16 | 0.70 | | | |
| | 11 | 128 | 0.28 | 0.62 | | | |
| | 12 | 128 | 0.28 | 0.62 | | | |
| | 13 | 128 | 0.28 | 0.62 | | | |
| | 1 | **32** | 0.43 | 0.53 | | | |
| | 2 | 16 | 2.31 | 0.82 | | | |
| | 3 | 128 | 2.31 | 1.00 | | | |
| | 4 | **16** | 2.31 | 0.82 | | | |
| | 5 | 128 | 0.57 | 0.70 | | | |
| | 6 | **64** | 2.31 | 0.90 | | | |
| Iteration 5 | 7 | **64** | 1.16 | 0.70 | 98.12 | 98.13 | 501.39 |
| | 8 | **64** | 0.57 | 0.70 | | | |
| | 9 | 128 | 1.16 | 0.80 | | | |
| | 10 | 128 | 1.16 | 0.65 | | | |
| | 11 | 128 | 0.28 | 0.65 | | | |
| | 12 | 128 | 0.28 | 0.65 | | | |
| | 13 | 128 | 0.28 | 0.65 | | | |
| | 1 | 32 | 0.43 | 0.57 | | | |
| | 2 | 16 | 2.31 | 0.85 | | | |
| | 3 | 64 | 1.16 | 0.80 | | | |
| | 4 | 16 | 1.16 | 0.65 | | | |
| | 5 | **64** | 0.28 | 0.65 | | | |
| | 6 | 64 | 1.16 | 0.80 | | | |
| Iteration 6 | 7 | 64 | 1.16 | 0.80 | 98.75 | 98.75 | 363.88 |
| | 8 | 64 | 0.28 | 0.65 | | | |
| | 9 | **64** | 0.28 | 0.65 | | | |
| | 10 | **64** | 0.28 | 0.65 | | | |
| | 11 | **64** | 0.07 | 0.61 | | | |
| | 12 | **64** | 0.07 | 0.61 | | | |
| | 13 | **64** | 0.07 | 0.61 | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | **16** | 0.21 | 0.60 | | | |
| | 2 | **8** | 0.57 | 0.82 | | | |
| | 3 | 32 | 0.28 | 0.70 | | | |
| | 4 | **8** | 0.28 | 0.62 | | | |
| | 5 | 64 | 0.14 | 0.70 | | | |
| | 6 | **32** | 0.57 | 0.90 | | | |
| Iteration 7 | 7 | **32** | 0.28 | 0.70 | 97.50 | 97.50 | 206.26 |
| | 8 | 64 | 0.14 | 0.70 | | | |
| | 9 | 64 | 0.28 | 0.80 | | | |
| | 10 | 64 | 0.28 | 0.80 | | | |
| | 11 | 64 | 0.07 | 0.65 | | | |
| | 12 | 64 | 0.07 | 0.65 | | | |
| | 13 | 64 | 0.07 | 0.65 | | | |
| | 1 | 16 | 0.217 | 0.65 | | | |
| | 2 | 8 | 0.578 | 0.85 | | | |
| | 3 | 32 | 0.289 | 0.80 | | | |
| | 4 | 8 | 0.289 | 0.80 | | | |
| | 5 | 32 | 0.072 | 0.65 | | | |
| | 6 | 32 | 0.289 | 0.80 | | | |
| Iteration 8 | 7 | 32 | 0.289 | 0.80 | 96.88 | 96.87 | 173.78 |
| | 8 | **32** | 0.072 | 0.65 | | | |
| | 9 | **32** | 0.072 | 0.65 | | | |
| | 10 | **32** | 0.072 | 0.65 | | | |
| | 11 | **32** | 0.018 | 0.61 | | | |
| | 12 | **32** | 0.018 | 0.61 | | | |
| | 13 | **32** | 0.018 | 0.61 | | | |
| | 1 | **8** | 0.108 | 0.60 | | | |
| | 2 | 8 | 0.289 | 0.85 | | | |
| | 3 | 32 | 0.289 | 1.00 | | | |
| | 4 | 8 | 0.289 | 0.85 | | | |
| | 5 | 32 | 0.072 | 0.70 | | | |
| | 6 | 32 | 0.289 | 1.00 | | | |
| Iteration 9 | 7 | 32 | 0.289 | 1.00 | 97.50 | 97.50 | 155.10 |
| | 8 | 32 | 0.072 | 0.70 | | | |
| | 9 | 32 | 0.072 | 0.70 | | | |
| | 10 | 32 | 0.072 | 0.70 | | | |
| | 11 | 32 | 0.018 | 0.62 | | | |
| | 12 | 32 | 0.018 | 0.62 | | | |
| | 13 | 32 | 0.018 | 0.62 | | | |

TABLE 3.4: Comparison of VGG16 and its Iterations on MACs, Suitability Score, Test Accuracy, F1-Score, and Epoch Time.

## 3.5 Summary

This chapter presented an iterative optimisation framework guided by a Suitability Score, aiming to align CNN architectures more effectively with Winograd convolution-based accelerators. By evaluating kernel size, computational complexity, and output channels, the score enabled targeted modifications to individual layers, enhancing computational efficiency and hardware compatibility without compromising model accuracy.

The methodology demonstrated substantial reductions in total MAC operations and training time through systematic refinements applied to benchmark architectures such as AlexNet and VGG16. Importantly, these improvements were achieved while maintaining — and in some cases even improving — inference accuracy, highlighting the effectiveness of selective, layer-wise optimisation over uniform network compression strategies.

Although the MAC counts presented in Section 3.4 were calculated based on conventional convolution operations (2.6), the optimised models remain fully deployable on standard platforms. Furthermore, when deployed with Winograd convolution, an additional MAC reduction of approximately 2.25$\times$ can be realised in compatible layers, offering further acceleration potential.

Critically, the proposed Suitability Score-based framework addresses key limitations that hinder the practical adoption of Winograd-based acceleration—specifically, filter size constraints and inefficiencies arising from uniform application. By guiding architectural modifications tailored to each layer, the approach improves the practical feasibility of deploying Winograd-optimised CNNs on resource-constrained platforms such as FPGAs.

Overall, this chapter established a hardware-aware, structure-specific optimisation methodology that strengthens CNN models' computational and practical deployability aspects. These advances form the foundation for the next chapter, which builds upon the optimised models by introducing adaptive hardware pipelining strategies to enhance inference throughput and resource efficiency in FPGA deployments.

# Chapter 4

# Adaptive Hardware Parallelism for Efficient CNN Acceleration on FPGAs

## 4.1 Introduction

Building upon the previous chapter's analysis of convolutional layer characteristics using Suitability Scores, this chapter presents a hardware-level optimisation methodology to enhance CNN acceleration efficiency on FPGAs. The Suitability Score framework, originally developed to quantify convolutional layers' computational intensity and structural complexity, is adapted in this chapter to guide layer-specific hardware parallelism. By aligning pipelining strategies with the computational profile of each layer, the proposed approach enables more effective balancing of performance and resource usage in hardware deployment.

Conventional FPGA-based CNN implementations typically apply uniform hardware parameters, such as fixed pipeline depth and loop unrolling factors, across all layers [119]. However, this layer-agnostic approach overlooks the diverse computational demands and structural complexity of different CNN layers, often leading to inefficient hardware utilisation and poor scalability when targeting larger or more complex models [81].

This chapter proposes a simulation-guided hardware optimisation strategy that dynamically assigns customised pipeline initiation intervals ( $I_I$) for each convolutional layer. By analysing the computational intensity and structural complexity of each layer, the method adaptively tunes parallelism settings at the HLS stage. This results in improved performance and resource efficiency compared to static, uniform pipelining strategies.

The proposed methodology achieves a balanced trade-off between latency and resource utilisation, particularly with respect to DSP, LUT, and FF usage. This approach addresses key limitations identified in traditional fixed-parameter designs.

Overall, this chapter establishes a structured hardware design methodology that enhances resource-aware CNN deployment, setting the stage for the system-level optimisations that will be explored in Chapter 5.

## 4.2 Software-to-Hardware Optimization Workflow for CNN-to-FPGA Deployment

As illustrated in Figure 4.1, the proposed design flow for CNN-to-FPGA deployment is organised into three main stages:

- **Model Training and Hardware-Level Translation:** The process begins with training and optimising a CNN model using the Suitability Score framework within Python-based environments such as TensorFlow or PyTorch. Once the model is optimised, its structural attributes and learned parameters are systematically extracted. These are translated into an HLS-compatible C++ representation through a custom Python-based automation process, ensuring that the network's parametrised structure is preserved.
- **Hardware-Oriented Optimisation and RTL Generation:** The generated C++ code is then refined through HLS tools to optimise the design for FPGA implementation. During this stage, hardware-specific optimisations such as pipelining, parallelism, and resource sharing are applied. Simulation and design space exploration are carried out to balance latency, throughput, and resource utilisation.
- **FPGA Integration and Bitstream Generation:** After optimisation, the C++ design is synthesised into Register-Transfer Level (RTL) descriptions. The synthesised IP cores are integrated into a larger FPGA system, placed and routed based on the target architecture, and programmed onto the FPGA device as a bitstream. Final verification and validation are performed on the physical hardware.

This structured process streamlines the transition from a trained CNN model to FPGA deployment by systematically handling model export, hardware-oriented optimisation, and physical implementation. Although the design flow automates significant portions of the translation, broader system-level synthesis and integration still follow standard FPGA design methodologies.

The subsequent subsections provide a detailed explanation of each stage of the proposed workflow. First, the model training and preparation phase is described, with a

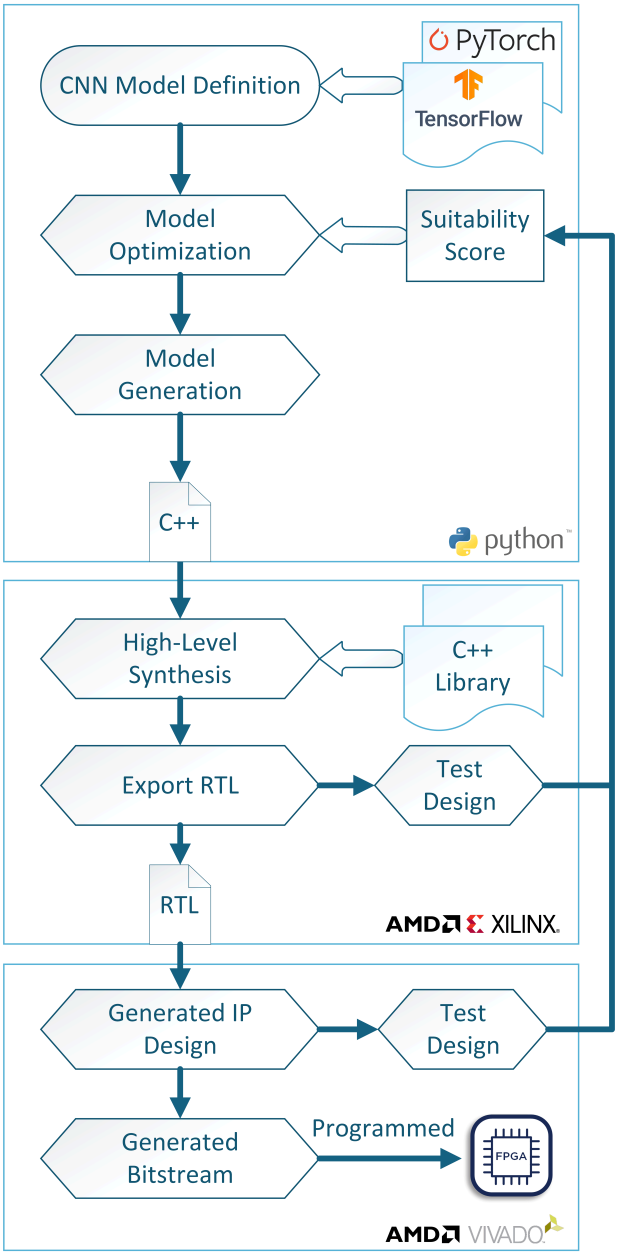



FIGURE 4.1: Automated CNN Model Translation from Python to HLS-Compatible C++ for FPGA Deployment

focus on extracting structural and parametric information for HLS translation. Then, the hardware-oriented optimisation techniques applied during the HLS process are outlined. Finally, the deployment of the optimised CNN model onto the FPGA device is presented, covering the generation of the RTL, IP integration, and bitstream creation.

### 4.2.1 Model Training and Hardware-Level Translation

The CNN model is trained under a structural optimisation regime guided by the proposed *S Score* methodology, as detailed in Chapter 3. In this approach, the model architecture is dynamically adapted during the training process to maximise hardware efficiency metrics while preserving inference accuracy. As a result, the final trained model is inherently optimised for hardware deployment without requiring post-training structural modifications.

Following the S Score-based training, the optimised CNN model is systematically analysed in the Python environment to extract the necessary information for hardware translation. This information is categorised into two groups:

- **Structural attributes:** These include the type of each layer (such as convolutional, pooling, or fully connected), the number of filters, kernel dimensions (height and width), stride values (vertical and horizontal), padding schemes ('valid' or 'same'), the number of input and output channels, and the activation function type (if applicable).
- **Learned parameters:** These are the trained weight tensors and bias vectors associated with each layer.

To facilitate hardware integration, the structural attributes are directly recorded into C/C++ header files (`.h`), where each attribute is defined as a compile-time constant or macro. Meanwhile, the learned parameters are initially saved in structured `.json` files. These `.json` files are subsequently processed by custom Python scripts to generate C header and source files containing parameter arrays. A user-defined data type, `data_t`, is employed for these arrays, which can be configured to represent either floating-point or fixed-point formats depending on hardware requirements. This flexible type system allows hardware implementations to balance precision and resource usage. An illustrative example of the generated header file for the kernel weights of a convolutional layer is shown in Listing 4.1.

```
1 #ifndef CONV1_WEIGHTS_3D
2 #define CONV1_WEIGHTS_3D
3
4 data_t conv1_weights[KERNEL_HEIGHT][KERNEL_WIDTH][IN_CHANNELS][
    OUT_CHANNELS] = {
5     { { {0.234, -0.128, 0.567, ...}, {...}, ... }, {...}, ... },
6     ...};
7 #endif // CONV1_WEIGHTS_3D
```

LISTING 4.1: Example of generated kernel weights header file with parametric dimensions

Once the HLS-compatible C++ code for the CNN model is developed, the hardware structural design becomes fixed and requires no modification unless the high-level network topology itself changes. The parametrised layer templates, which encapsulate all operational logic and dataflow structures, remain unchanged across different training instances.

After each training, the updated structural attributes and the learned parameters (weights and biases) are automatically extracted and exported into C-compatible formats. The structural attributes update the compile-time configuration constants, while the learned parameters are integrated as `const` arrays within the existing HLS C++ design. This process enables the hardware implementation to adapt to the newly trained model without requiring any manual modification to the functional code structure.

As a result, the hardware translation workflow becomes highly efficient: the structural design is created once, and only the parameter arrays need to be updated for different trained models. This method reduces development time, eliminates code redundancy, and ensures consistent hardware behaviour across retrained networks.

### 4.2.2 Hardware-Oriented Optimisation and RTL Generation

In this phase, the HLS-compatible C++ code—generated from the trained and optimised CNN model—is synthesised into RTL representations using Xilinx Vivado HLS 2023.2. Each C++ function corresponds to a specific CNN layer, such as a convolutional, pooling, or fully connected layer, and is individually analysed and translated into a dedicated hardware module.

Several hardware optimisation directives were systematically applied during the high-level synthesis process to enhance performance and resource efficiency. These optimisations included:

- **Loop Pipelining:** Key computational loops were pipelined to reduce $I_I$ and improve overall throughput.
- **Loop Unrolling:** Selective unrolling of inner loops was applied to increase parallelism, subject to resource availability constraints.
- **Array Partitioning:** Arrays representing weights, activations, and intermediate buffers were partitioned to enable concurrent memory accesses.
- **Resource Binding:** Specific operations, particularly multipliers and adders, were explicitly bound to DSP blocks to maximise performance efficiency.
- **Dataflow Optimisation:** Load, compute, and store operations were task-pipelined to achieve higher system-level concurrency.

- **Interface Specification:** AXI4-Stream and AXI4-Lite interfaces were defined for external memory and control signal integration, facilitating seamless system-level integration.

In addition to performance-oriented optimisations, careful monitoring of FPGA resource utilisation was conducted throughout the synthesis process. HLS-generated reports detailing LUT, FF, DSP, and BRAM usage were systematically reviewed. Based on these reports, iterative refinements were applied to balance latency, throughput, and hardware area, ensuring that the final design would meet the constraints of the target FPGA device.

It is important to note that if the synthesised design fails to meet the FPGA resource constraints after high-level synthesis, an additional optimisation step is performed. In such cases, the model structure is revisited at the Python level, and layer-level modifications are applied to adjust computational complexity.

To facilitate this process, the proposed workflow integrates an adaptive Suitability Score strategy that dynamically adjusts the parallelism and resource demands of individual CNN layers according to the available FPGA resources. The adaptive S Score is also utilised during the HLS stage to guide layer-level parallelism decisions, ensuring that the generated hardware modules balance performance and resource utilisation effectively. This adaptive approach enables the model to be systematically re-optimised and re-synthesised as necessary to meet design constraints. Further details regarding the adaptive S Score methodology are presented in Section 4.3 of this chapter.

This stage resulted in a complete set of synthesisable RTL descriptions, including Verilog files and associated synthesis reports. These RTL modules formed the foundation for subsequent FPGA integration and bitstream generation phases.

### 4.2.3 FPGA Integration and Bitstream Generation

After the high-level synthesis phase, the generated RTL modules were integrated into a complete hardware system using Xilinx Vivado. During this stage, the synthesised components were mapped onto the physical resources of the target FPGA, including logic blocks, DSP slices, and BRAMs. Placement and routing processes were conducted to optimise the timing and resource utilisation of the design.

Following successful placement and routing, the hardware system was compiled into a configuration bitstream file, enabling the programming of the FPGA device. Fundamental timing analysis and functional verification steps were performed to ensure the integrated system met initial design specifications and operated reliably under expected conditions.

A PCIe interface was employed during the system integration to facilitate high-speed data transfer between the host machine and the FPGA. Input data for the CNN inference operations was transmitted over the PCIe connection, and the resulting output predictions were likewise retrieved via the same interface, ensuring efficient communication for real-time hardware validation.

Chapter 5 presents the full experimental validation of the optimised CNN models on FPGA hardware, including performance metrics such as inference latency, throughput, and resource utilisation. This subsequent analysis provides a comprehensive evaluation of the hardware deployment results.

## 4.3 Optimising CNN Deployment on FPGA with Layer-Specific Design Strategies

This section introduces a layer-specific optimisation methodology to enhance the overall effectiveness of CNN deployment on FPGA platforms. Specifically, the approach aims to improve *performance*—measured in terms of inference latency, throughput, and model accuracy—and *efficiency*, referring to the optimised use of FPGA hardware resources such as BRAMs, DSPs, and logic slices. Instead of applying a uniform hardware configuration across the entire model, this approach dynamically tunes hardware-level optimisation parameters such as pipelining depth, loop unrolling, and memory partitioning based on the computational characteristics of each convolutional layer.

This methodology considers layer-wise computational requirements and previously defined Suitability Scores (introduced in Chapter 3), enabling differentiated hardware configurations. Layers with intensive arithmetic operations are assigned more optimisation directives to boost parallelism, while less demanding layers are configured conservatively to save area and power.

This layer-aware design strategy improves the balance between performance and resource utilisation and provides a flexible framework applicable to diverse CNN architectures and FPGA platforms. It also facilitates scalable deployment under varying hardware constraints without compromising model fidelity.

This section also discusses hardware-level optimisation of convolutional layers, focusing on implementations based on the Winograd minimal filtering algorithm. The design emphasises architectural decisions such as pipelining, loop unrolling, and memory partitioning, all tailored per layer based on computational demand and resource availability. These strategies aim to enhance latency and throughput performance in FPGA deployments.

### 4.3.1 Architectural Design of Winograd-Based Convolutional Layers on FPGA

This section presents the architectural design of the Winograd-based convolutional processing unit implemented on the FPGA. To enhance computational efficiency, a tiled processing strategy is adopted, where the convolution operation is divided into $m \times m$ output tiles, computed using the Winograd transformation $F(m, r)$ for an $r \times r$ kernel.

In this design, the convolution between a given filter $g$ and an input tile $d$ is performed using the transformation: $Y = A^T \left( (GgG^T) \circ (B^T dB) \right) A$. Where $G$, $B$, and $A$ are precomputed transformation matrices corresponding to the filter, input, and output domains, respectively, and $Y$ denotes the resulting output tile.

The computation proceeds through three sequential stages. Firstly, the **input and filter data** are transformed into the Winograd domain. Specifically, the input tiles are transformed as $V = B^T dB$, while the filters are transformed as $U = GgG^T$, reducing the computational complexity compared to standard convolution. Secondly, the **core computation** is carried out via element-wise multiplication of the transformed input and filter data, producing the intermediate matrix $M = U \circ V$. Finally, in the **output transformation** stage, the matrix $M$ is mapped back to the spatial domain using $Y = A^T MA$, yielding the final output tile.

The hardware realisation of this flow is depicted in Figure 4.2. The architecture is designed in a modular fashion to maximise resource reuse, pipeline efficiency, and throughput. The Winograd PE consists of the following principal components:

- **Input Tile Buffer:** Temporarily stores input tiles fetched from external memory. This buffering ensures that input data is continuously supplied to the transformation units without stalling the pipeline. By decoupling memory access from computation, it maintains high throughput and prevents pipeline stalls due to memory latency.
- **Filter Buffer:** Holds the $3 \times 3$ convolution kernels for each input channel. Storing filters locally minimises repeated memory accesses, reduces external memory bandwidth requirements, and allows fast reconfiguration when switching between different convolutional layers during inference.
- **Transform Units:** Execute the $B^T dB$, $GgG^T$, and $A^T MA$ transformations. These units are heavily pipelined to maximise concurrency and minimise the overall computation time. By decomposing convolution into structured matrix operations, they exploit arithmetic regularity and reduce the number of multiplications compared to direct convolution.

- **ROM Blocks:** Store the constant matrices $A$, $B$, and $G$, eliminating the need to recompute or reload them during operation. This greatly reduces computational overhead, saves logic resources, and simplifies the control path design.
- **Intermediate Buffers:** Temporarily store the intermediate matrices $U$, $V$, and $M$ between transformation stages. These buffers ensure seamless data handoff between pipeline stages, avoiding data hazards and supporting continuous high-throughput pipelined execution.

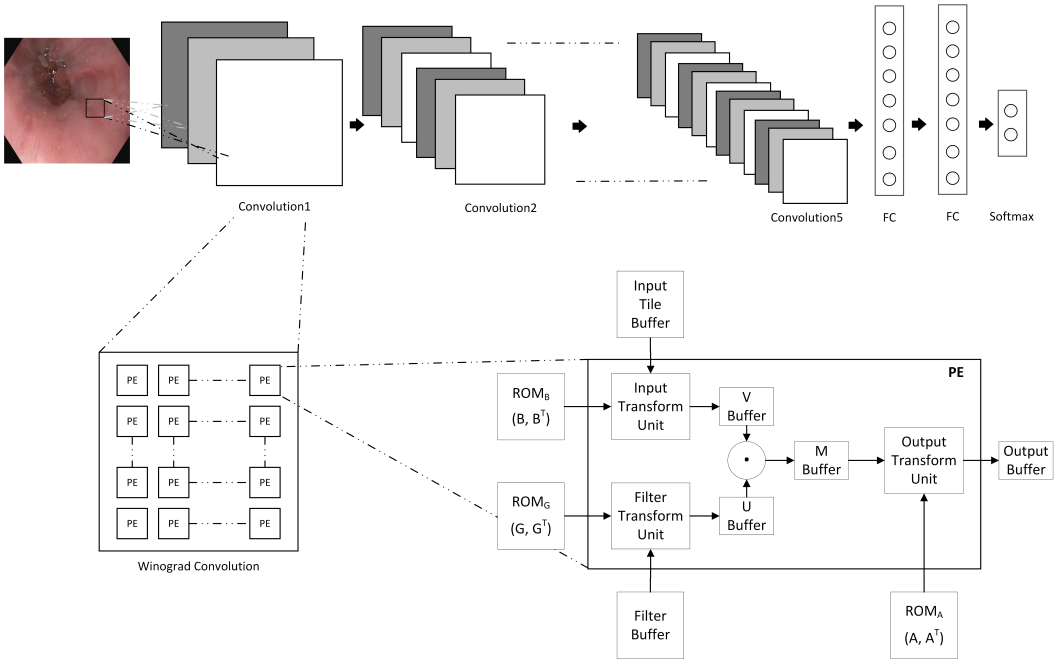



FIGURE 4.2: Winograd Processing Element (PE) Architecture with Input/Output Transform Buffers and Dataflow

This design's modularity enables flexible scaling across different CNN architectures and supports parallel instantiation of multiple PEs, further enhancing computational throughput on FPGA platforms. Overall, the modular Winograd PE architecture is designed to efficiently map convolution operations onto FPGA hardware by leveraging pipelined matrix transformations and localised data storage. This organisation reduces the arithmetic complexity of convolutions and improves dataflow efficiency, contributing to better utilisation of FPGA resources. Moreover, the modular structure facilitates scalable deployment, making it adaptable to different CNN architectures and performance requirements.

However, while the modular design enhances scalability and resource reuse, achieving high computational throughput and low inference latency necessitates further fine-grained optimisations at the operation level. In particular, pipelining techniques are critical to maximising concurrency within the processing elements and minimising execution bottlenecks.

Building upon this foundation, the following section explores how pipelining techniques are systematically employed to optimise computational throughput and minimise latency in FPGA-based CNN acceleration.

### 4.3.2 Pipelining of FPGA-based CNN Acceleration

In high-level synthesis for FPGA-based deep learning acceleration, performance bottlenecks often stem from sequential loop execution and inefficient hardware scheduling. Pipelining and loop unrolling are commonly employed as key optimisation strategies to overcome these limitations. These techniques enable the parallel execution of operations, improving throughput and reducing latency.

While pipelining overlaps the execution stages of consecutive loop iterations, loop unrolling replicates hardware resources to perform multiple iterations simultaneously. However, the benefits of these strategies are tightly coupled with the structure and computational characteristics of the specific CNN layers. Without careful tuning, application of pipelining and unrolling may lead to excessive utilisation of FPGA resources, such as DSP blocks and BRAMs, potentially violating timing or area constraints.

Instead of applying uniform optimisation settings across all convolutional layers, it is crucial to adopt a layer-specific strategy to avoid these issues. This section explores the effectiveness of pipelining and loop unrolling by conducting systematic experiments with varying parameter configurations. In particular, when applied to Winograd-based convolutional layers, the effects of varying pipeline initiation intervals and unrolling factors on performance, latency, and resource utilisation are investigated. These insights support a more adaptive optimisation approach tailored to each layer's computational characteristics.

To analytically characterise these techniques, consider a loop with $N$ iterations and a latency of $L$ clock cycles per iteration. Without pipelining, the total latency is $N \times L$. Applying pipelining with an $\mathrm{I_I}$ of $k$ reduces the latency to approximately $(N-1) \times k + L$, as a new iteration begins every $k$ cycles. When $k$ approaches 1, throughput improves, although this may come at the cost of increased resource usage.

Similarly, unrolling a loop with a factor of $U$ allows $U$ iterations to execute in parallel, reducing the total latency to $\frac{N}{U} \times L$. While pipelining and unrolling are effective methods for accelerating loop execution, they must be carefully calibrated in FPGA implementations to maintain resource efficiency. In CNN accelerators—where computations often consist of deeply nested loops—layer-aware optimization of these techniques is critical for achieving a balanced trade-off between performance and hardware cost.

```
1 for (int f = 0; f < num_filters; f++) {
2     for (int c = 0; c < input_channels; c++) {
3         // Load 3x3 filter from Filter Buffer
```

```
        ExtractFilter(weights, Filter_Buffer, f, c);
        // Transform filter using Filter Transform Unit (G)
        Filter_Transform_Unit(Filter_Buffer, U_Buffer);
        for (int i = 0; i < num_tiles; i++) {
            for (int j = 0; j < num_tiles; j++) {
#pragma HLS PIPELINE II=PIPELINE_FACTOR
                // Load 4x4 tile from Input Tile Buffer
                LoadInputBlock(input_image, Input_Tile_Buffer, i, j, c);
                // Transform input using Input Transform Unit (B)
                Input_Transform_Unit(Input_Tile_Buffer, V_Buffer);
                // Element-wise multiplication in PE
                ComputeHadamardProduct(V_Buffer, U_Buffer, M_Buffer);
                // Output transform (A)
                Output_Transform_Unit(M_Buffer, Output_Tile);
                // Store results
                StoreOutput(output_feature_map, Output_Tile, i, j, f);
            } } } }
```

LISTING 4.2: Winograd Convolution with Hardware-Aligned Terminology

To maximise throughput and reduce latency in FPGA-based CNN implementations, loop pipelining is applied to the core structure of convolutional layers. A simplified view of the pipelined loop structure used in the first convolutional layer is shown in Listing 4.2. The `#pragma HLS PIPELINE` directive is inserted into the inner loops to enable concurrent execution of loading, transformation, and output operations.

### 4.3.3 Impact of Pipeline Depth on Latency and Resource Efficiency

To analyse the effect of pipelining depth on latency across convolutional layers, the Suitability Score-based AlexNet architecture was applied on the Vitis HLS tool. The experiments were conducted using HLS via Vitis HLS, without performing RTL implementation, physical place-and-route, or FPGA bitstream generation. Instead, performance metrics such as latency and resource usage were extracted from the post-synthesis reports generated by Vitis HLS for the target AMD Xilinx Kintex UltraScale+ FPGA (XCKU5P). The design was modelled in C++ and utilised 32-bit single-precision floating-point arithmetic to maintain consistency with standard Python-based inference behaviour.

This experiment focuses on understanding resource limitations and pipelining efficiency under full-precision conditions, isolating the effects of hardware-level scheduling (e.g., pipelining, loop unrolling) without applying quantisation or bitwidth reduction. Each convolutional layer was synthesised individually using different `#pragma HLS PIPELINE` $I_I$ values: $I_I$ = 1, 2, 3, 4, 8, 16. The $I_I$ in HLS refers to the number of clock cycles between the start of consecutive loop iterations. A lower $I_I$ indicates

higher concurrency, resulting in reduced latency, while a higher $I_I$ may lower resource utilisation but increase latency.

TABLE 4.1: Classification of pipelining configurations based on $I_I$ and design trade-offs.

| Configuration | $I_I$ Range | Description |
|---|---|---|
| **Low Latency** | 1–2 | Prioritises speed through aggressive pipelining, with higher resource usage. |
| **Moderate** | 3–4 | Offers a trade-off between latency and hardware utilisation. |
| **Resource-Efficient** | 8 or more | Minimises resource usage at the expense of increased latency. |

To support consistent analysis of the experimental results, pipelining configurations are grouped into three categories based on $I_I$ values and their associated design trade-offs, as shown in Table 4.1. The *Low Latency* configuration ( $I_I$ = 1–2) emphasises minimal execution delay by enabling deeper pipelining, which typically increases hardware resource usage. The *Moderate* configuration ( $I_I$ = 3–4) represents a compromise between latency and resource utilisation. Lastly, the *Resource-Efficient* configuration ( $I_I \geq 8$) reduces logic and memory overhead by limiting concurrency, resulting in higher latency but improved hardware efficiency.

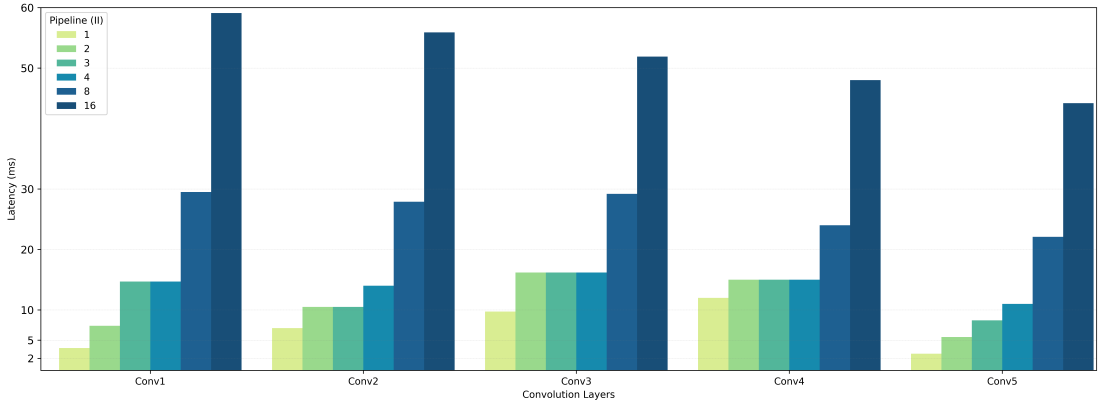



FIGURE 4.3: Latency comparison across different pipeline $\mathbf{I_I}$ for convolutional layers in the Suitability Score-based AlexNet architecture.

The effect of different pipeline $I_I$ was evaluated across all convolutional layers to assess their influence on performance. As shown in Figure 4.3, lower $I_I$ values (e.g., $I_I$=1) provide the lowest latency due to maximum parallelism enabled by the pipeline architecture. Conversely, increasing the $I_I$ value reduces the concurrency of the operations, resulting in higher latency across all layers.

**Conv1 and Conv2**, which operate on high-resolution feature maps (e.g., 224×224×3), exhibit the most dramatic latency degradation as the $I_I$ value increases. This is attributed to the large volume of data that needs to be processed per cycle, making them

highly sensitive to pipeline depth. **Conv3** demonstrates a plateau in latency at $I_I$=3–4, indicating the existence of possible pipeline stalls or memory access bottlenecks that limit performance scaling.

**Conv4**, positioned deeper in the network, processes reduced feature map sizes while still maintaining a substantial filter count. Interestingly, it demonstrates a moderate latency increase pattern compared to the earlier layers. Although the latency difference between $I_I$=1 and $I_I$=16 remains significant, it is not as dramatic as observed in Conv1 or Conv2. This is likely due to a trade-off between reduced input volume and increasing filter complexity. From a hardware standpoint, Conv4 benefits from pipelining but also appears to reach diminishing returns in latency improvement beyond $I_I$=4, suggesting that compute-bound factors (such as DSP saturation or limited unroll efficiency) become more dominant than memory-bound limitations. **Conv5**, which operates on smaller spatial dimensions, maintains relatively low latency, but still exhibits sensitivity to $I_I$ values, confirming that even smaller layers benefit from pipelining to a certain extent.

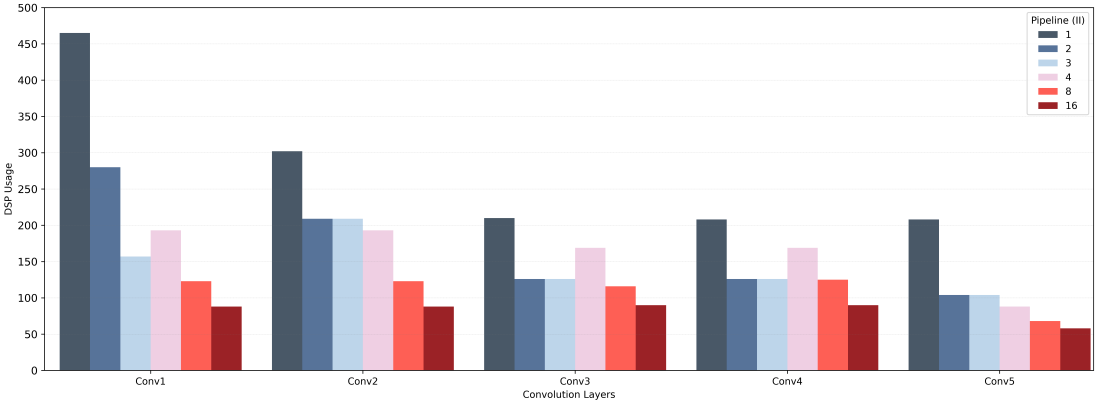



FIGURE 4.4: DSP consumption across different pipeline $\mathbf{I_I}$ for convolutional layers in the Suitability Score-based AlexNet architecture.

As shown in Figure 4.4, lower initiation intervals (e.g., $I_I$=1) demand substantially higher DSP resources across all layers. This is expected, as low-latency pipelining introduces multiple simultaneous operations requiring parallel multiply-accumulate units. Conv1 stands out with the highest DSP usage under full pipelining due to its high-resolution input and large computation volume. Conversely, increasing the $I_I$ value reduces parallel operations, which consequently lowers DSP demand. By $I_I$=16, all layers exhibit reduced DSP usage demonstrating a trade-off between parallelism and resource efficiency.

Similarly, Figure 4.5 reveals that LUT usage inversely correlates with $I_I$ depth. Pipeline control logic, intermediate registers, and operator replication at lower $I_I$ values contribute to elevated LUT usage. Conv1 and Conv2, operating on the largest data sizes, again require the most LUTs, particularly under $I_I$=1 and $I_I$=2. However, hardware

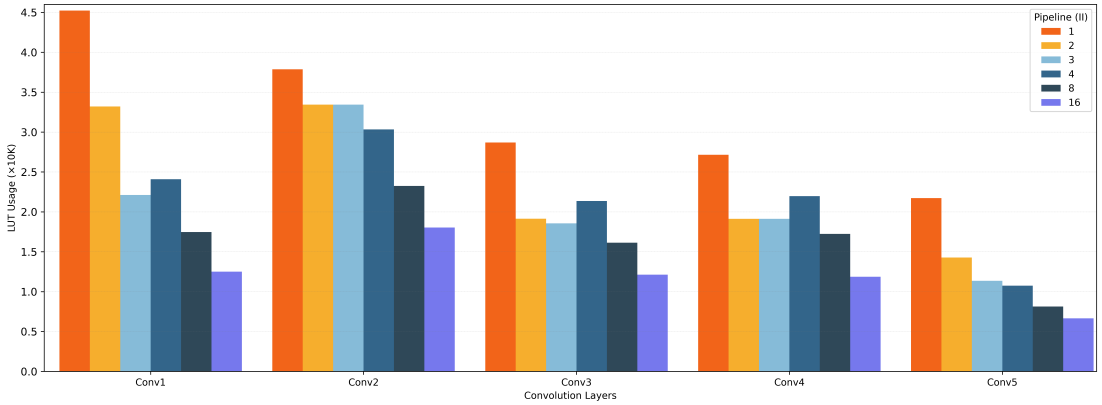



FIGURE 4.5: LUT usage across different pipeline $\mathbf{I_I}$ for convolutional layers.

replication is minimized as $I_I$ increases, yielding substantial reductions in LUT consumption. Conv4 and Conv5, although processing smaller feature maps, still show visible drops in LUT use when moving from $I_I$=1 to $I_I$=16.

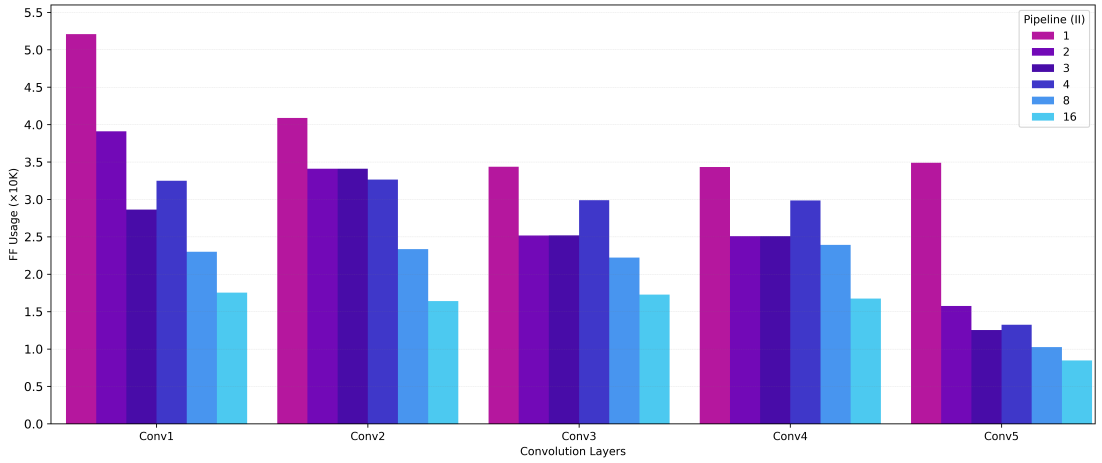



FIGURE 4.6: Flip-Flop (FF) usage across different pipeline $\mathbf{I_I}$ for convolutional layers in the Suitability Score-based AlexNet architecture.

As outlined in Table 4.1, the experimental results demonstrate distinct trade-offs between the three pipelining configurations. The *Low Latency* configuration ( $I_I$ = 1–2) achieves the shortest execution times but leads to the highest usage of DSPs, LUTs, and flip-flops. Conversely, the *Resource-Efficient* configuration ( $I_I \geq 8$) reduces hardware resource utilisation but incurs a notable increase in latency. The *Moderate* configuration ( $I_I$ = 3–4) offers a compromise by reducing resource consumption relative to the Low Latency configuration, while maintaining acceptable latency levels.

These findings confirm that uniform pipelining across layers leads to unbalanced resource usage. Instead, a *layer-specific pipelining strategy*, informed by workload characteristics, provides a more effective trade-off between performance and hardware utilisation. This approach aligns with the Suitability Score-guided methodology, which

recommends deeper pipelining for compute-intensive layers and more relaxed configurations for lighter stages.

### 4.3.4 Adaptive Pipeline Scheduling using Suitability Score Per Layer

The previous section examined latency and resource utilisation trends under various static pipelining configurations applied uniformly across convolutional layers. The results indicated that such uniform strategies may not effectively balance performance and hardware efficiency, particularly under constrained resource budgets. While low-latency pipelining improves the latency of compute-intensive layers, it can lead to excessive DSP and flip-flop allocation in computationally lightweight layers. These observations highlight the need for a more adaptive approach that considers the varying computational characteristics of each layer.

To address this challenge, an *adaptive pipelining strategy* is proposed, in which $I_I$ values are assigned on a per-layer basis, guided by a simplified formulation of the Suitability Score introduced in Chapter 3. Given that the CNN model has already been restructured into a Winograd-compatible format featuring uniform $3 \times 3$ convolution kernels, the filter compatibility component ($\alpha$) becomes constant and thus negligible in the scoring calculation. Accordingly, a layer-specific score, denoted as $S'_{\text{score}}$, is redefined as follows:

$$S'_{\text{score}} = \beta \cdot \left( \frac{\text{MAC}_{\text{layer}}}{\text{MAC}_{\text{ref}}} \right) + \gamma \cdot \left( \frac{C_{\text{out}}}{C_{\text{out,ref}}} \right) \tag{4.1}$$

$\text{MAC}_{\text{layer}}$ represents the number of multiply-accumulate operations in the layer, and $C_{\text{out}}$ denotes the number of output channels. Reference values ($\text{MAC}_{\text{ref}}$ and $C_{\text{out,ref}}$) are selected as the maximum observed values across all layers to ensure proper normalisation. $\beta$ and $\gamma$ are weighting factors assigned to the computational complexity and output feature dimensions, respectively. These weights satisfy the constraint $\beta + \gamma = 1$ to ensure balanced contribution between the two factors.

In the experiments, the parameters were set as $\beta = \gamma = 0.5$, assigning equal weight to computational intensity and feature complexity. Based on the computed $S'_{\text{score}}$ values, pipeline $I_I$ levels are adaptively mapped according to a tiered assignment strategy outlined in Table 4.2. Layers with higher $S'_{\text{score}}$ values are allocated lower $I_I$ settings to maximise parallelism, while layers with lower $S'_{\text{score}}$ values adopt more relaxed pipelining configurations to improve resource efficiency.

TABLE 4.2: Adaptive pipeline assignment strategy based on layer-wise $S'_{\text{score}}$. Lower pipeline $I_I$ values correspond to higher parallelism and throughput, whereas higher $I_I$ values favour reduced resource usage.

| $S'_{\text{score}}$ Range | Assigned Pipeline $I_I$ |
|---|---|
| $S'_{\text{score}} > 0.70$ | 1-2 |
| $0.40 < S'_{\text{score}} \leq 0.70$ | 2–4 |
| $S'_{\text{score}} \leq 0.40$ | 4–8 or more |

#### 4.3.4.1 Validation of Adaptive Pipelining on CNN Deployment

The method was applied to the optimised AlexNet model using the previously computed Suitability Scores to validate the effectiveness of the adaptive pipelining strategy. Specifically, $S'_{\text{score}}$ was employed to determine appropriate pipeline $I_I$ for each convolutional layer, thereby tailoring the pipelining depth to the unique computational demands of each layer.

Based on the $S'_{\text{score}}$ scores, pipeline $I_I$ were assigned to the five convolutional layers in AlexNet. As shown in Table 4.3, Conv1 and Conv4 were allocated an $I_I$ of 3, Conv2 was assigned $I_I = 2$, while Conv3 and Conv5—identified as the most compute-intensive layers—received the lowest $I_I$ setting of 1.

TABLE 4.3: Assigned pipeline $I_I$ for each convolutional layer in the AlexNet model, based on $S'_{\text{score}}$.

| Layer | Filter Count | $S'_{\text{score}}$ | Assigned $I_I$ |
|---|---|---|---|
| Conv1 | 10 | 0.60 | 3 |
| Conv2 | 12 | 0.65 | 2 |
| Conv3 | 40 | 0.99 | 1 |
| Conv4 | 12 | 0.64 | 3 |
| Conv5 | 40 | 0.99 | 1 |

To evaluate the effectiveness of the proposed adaptive pipelining strategy, the CNN model was synthesised and analysed under both fixed- $I_I$ and adaptive- $I_I$ configurations. Table 4.4 summarises the total latency and resource utilisation metrics (DSP, LUT, and FF usage) for each approach.

To assess the effectiveness of the proposed adaptive pipelining strategy, its performance is compared against fixed- $I_I$ configurations ( $I_I$=1, $I_I$=2, $I_I$=3, $I_I$=4) across all key hardware metrics. In terms of latency, the adaptive $S'_{\text{score}}$ method achieves a total execution time of $3.77 \times 10^7$ cycles. This represents only a modest 7.1% increase compared to the low-latency $I_I$=1 setting, while outperforming the moderate $I_I$=2, $I_I$=3, and $I_I$=4 strategies by approximately 30.9%, 30.4%, and 46.8% respectively. These results confirm that

TABLE 4.4: Total latency and resource utilisation for different pipeline strategies.

| Pipeline Strategy | Total Latency (cycles) | DSP Usage | LUT Usage | FF Usage |
|---|---|---|---|---|
| $I_I = 1$ | $3.52 \times 10^7$ | 1393 | 160683 | 196589 |
| $I_I = 2$ | $5.46 \times 10^7$ | 845 | 119187 | 139234 |
| $I_I = 3$ | $5.42 \times 10^7$ | 722 | 104588 | 125556 |
| $I_I = 4$ | $7.09 \times 10^7$ | 812 | 108520 | 138152 |
| **Adaptive $S'_{score}$** | $\mathbf{3.77 \times 10^7}$ | **929** | **136193** | **148416** |

the adaptive scheduling maintains near-optimal latency without the excessive parallelism costs associated with the $I_I$=1 configuration.

Regarding DSP utilisation, the adaptive method achieves a significant 33.3% reduction compared to $I_I$=1 (929 vs. 1393 DSPs). While DSP usage increases slightly relative to $I_I$=2 and $I_I$=4 (by 9.9% and 14.4% respectively), and more notably relative to $I_I$=3 (by 28.7%), these increases are offset by the substantial latency improvements. Such reductions in DSP utilisation directly enable the deployment of larger CNN models or allow fitting multiple CNN instances on a single FPGA device, enhancing system scalability.

For LUT and FF usage, the adaptive strategy consumes moderately higher resources compared to the $I_I$=2, $I_I$=3, and $I_I$=4 strategies. Specifically, LUT usage increases by approximately 14.3%, 30.2%, and 25.5% respectively, while FF usage rises by 6.6%, 18.2%, and 7.4%. These increases are attributed to the additional control and buffering logic introduced by deeper pipelining in computationally intensive layers.

Overall, the adaptive $S'_{score}$-guided pipelining strategy strikes a balanced trade-off: it achieves significant reductions in latency compared to moderate fixed- $I_I$ configurations, while maintaining DSP, LUT, and FF usage at acceptable levels. By dynamically tailoring the pipeline depth to each layer's computational intensity and structural complexity, the strategy ensures that compute-heavy layers receive low-latency pipelining, while lightweight layers are conservatively scheduled, leading to optimal orchestration of FPGA resources without overprovisioning.

These results validate the practical advantages of layer-specific adaptive scheduling for the efficient deployment of FPGA-based CNNs. Moreover, by improving the scalability and hardware portability of CNN deployments, the proposed approach paves the way for broader adoption of FPGAs in resource-constrained CNN applications.

## 4.4 Summary

This chapter presented a simulation-guided hardware optimisation methodology to enhance the efficiency of CNN deployment on FPGAs. The primary focus was on adaptively configuring pipeline parallelism by analysing the unique computational characteristics of each convolutional layer. Rather than applying static, uniform hardware parameters across the network, leveraged layer-wise synthesis insights to assign pipeline $I_I$ based on a simplified Suitability Score formulation. This score captured both computational intensity and structural complexity, enabling more informed hardware design decisions at the HLS level.

The resulting FPGA configurations are optimised by integrating the adaptive strategy into the pre-deployment design flow to balance performance and resource constraints. This early-stage optimisation reduces the reliance on costly trial-and-error iterations during physical place-and-route and accelerates the overall deployment timeline. Additionally, by systematically identifying layer-specific bottlenecks, the approach facilitates finer-grained resource management across CNN layers.

Experimental results demonstrated that conventional static pipelining strategies, while straightforward to implement, often fail to offer a balanced trade-off between latency and hardware utilisation. Low Latency fixed- $I_I$ settings ( $I_I$ = 1) minimised latency but incurred excessive DSP and logic resource usage, making them impractical for resource-constrained FPGA deployments. Moderate fixed- $I_I$ configurations improved resource efficiency but suffered from significant latency penalties.

In contrast, the proposed adaptive pipelining strategy achieved a near-optimal balance. Compared to the aggressive $I_I$=1 configuration, it reduced DSP consumption by over 33%, LUT usage by approximately 15%, and FF usage by around 25%, while incurring only a 7% latency overhead. These improvements reflect the effectiveness of balancing computational throughput and resource efficiency through workload-aware pipelining.

Importantly, while the methodology was evaluated in the context of CNN-based architectures using Winograd-optimised convolution layers, the underlying principles of layer-aware hardware scheduling hold broader applicability. Future work could investigate extending this strategy to alternative deep learning models with different computational patterns, such as transformer-based networks.

Overall, these findings validate the practical advantages of simulation-guided, layer-specific adaptive scheduling for improving the scalability, efficiency, and deployability of FPGA-accelerated CNN inference systems, particularly under stringent resource constraints.

Building upon the findings of this chapter, the next chapter transitions from simulation-guided hardware optimisation to the full-scale hardware deployment of CNN models on FPGA platforms. It details the hardware-software co-design methodology, PCIe-based communication strategies, and real-world evaluation of the implemented accelerator, further demonstrating the practical viability and scalability of the proposed approaches.

# Chapter 5

# Hardware-Software Co-Design and Deployment of Layer-Optimised CNNs on FPGAs

## 5.1 Introduction

This chapter presents the full hardware deployment of the optimised CNN model on an FPGA platform, building upon the architectural and hardware-aware optimisation techniques introduced in previous chapters. The deployment targets oesophageal cancer detection using RGB endoscopic images, which involves strict latency constraints and high accuracy requirements, while also operating under limited hardware resources. This application scenario offers a practical setting for evaluating the effectiveness of the proposed optimisation techniques under realistic deployment conditions.

A hardware-software co-design approach is proposed to enable the deployment of the adaptively optimised CNN model onto FPGA hardware. The objective is to achieve both resource and performance-efficient implementation, facilitating seamless integration between a host CPU and the FPGA accelerator via a high-throughput PCIe interface. In this architecture, the host is responsible for preparing input data, managing memory transactions, and orchestrating inference tasks. At the same time, the FPGA implements a highly parallel, layer-specific pipelined CNN processing engine optimised according to the techniques developed in Chapter 4.

A critical aspect of this deployment is the quantisation of data representations from floating-point to fixed-point formats. Specifically, 16-bit and 8-bit fixed-point precisions are examined to strike a balance between computational efficiency and classification accuracy. The system adopts a modular architecture, where each CNN layer,

implemented as an independent IP core, communicates through a shared DDR4 memory space managed by internal memory controllers and accessed via the PCIe interface.

Unlike conventional CNN deployments on FPGAs, the proposed approach dynamically adapts hardware configurations based on per-layer computational profiling and resource-aware scheduling strategies. This enables a systematic balance between latency, resource efficiency, and inference accuracy in real-world FPGA deployments.

Throughout this chapter, practical challenges associated with deploying a full CNN model on FPGA hardware are addressed, including PCIe-based communication management, DDR4 memory integration, IP core synthesis, and system-level timing closure. Experimental evaluations are conducted to measure latency, resource utilisation, throughput, and power efficiency across different precision settings.

Finally, the implemented FPGA accelerator is benchmarked against conventional GPU-based executions and recent FPGA-based solutions reported in the literature. These comparisons highlight the strengths of the proposed hardware-software co-design approach in achieving high performance and efficient resource utilisation in realistic deployment scenarios.

## 5.2 Hardware-Software Co-Design for CNN Deployment on FPGA

Figure 5.1 illustrates the complete system architecture for FPGA-based CNN acceleration for cancer detection. The design is organised into three hierarchical layers: the host PC interface, the data transfer pipeline, and the FPGA-resident CNN processing engine. Each component optimises high-throughput, low-latency inference under constrained hardware resources.

### 5.2.1 Host PC with PCIe Communication

In this deployment setup, a general-purpose desktop PC running Windows functions as the central control unit for managing CNN inference tasks targeting oesophageal cancer detection from RGB endoscopic images. The host system orchestrates data flow, handles user-level control logic, and prepares input images for FPGA-side processing. Images are preprocessed in C++ environments. They are resized, normalised, and converted into 16-bit fixed-point format to match the quantisation requirements of the FPGA accelerator.

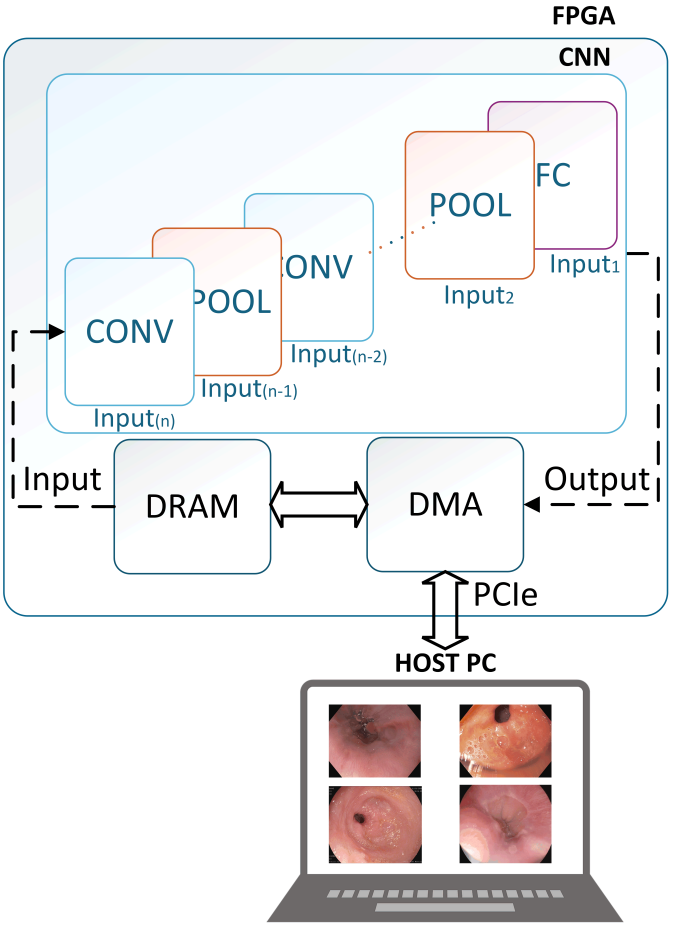



FIGURE 5.1: Overview of the hardware-software co-design for FPGA-based CNN deployment for cancer detection.

### 5.2.2 DMA and On-Board DRAM Buffers

At the boundary between the host and the FPGA, a DMA (Direct Memory Access) controller is instantiated using Xilinx XDMA IP. This module orchestrates the memory-mapped data exchange between the PCIe endpoint and the FPGA's on-chip DRAM. The incoming data is first buffered into external DRAM blocks, where it awaits processing by the CNN pipeline. Using a dual-port DRAM structure, the system achieves concurrent read/write operations, enabling continuous streaming between the host and the FPGA compute core.

### 5.2.3 CNN Processing Pipeline on FPGA

The CNN accelerator is implemented as a single HLS-based Intellectual Property (IP) core that encapsulates the entire network. Within this core, individual convolutional and activation layers are described as modular processing stages and synthesised using

Vitis HLS. These stages are integrated into a deeply pipelined architecture to maximise concurrency and throughput. The main computational stages include:

- **CONV Blocks:** Each convolution block receives input feature maps, applies filter weights and produces intermediate activations. The pipeline $I_I$ of each block is carefully tuned using the Suitability Score framework, allowing compute-heavy layers to operate with minimal $I_I$ values (e.g., $I_I = 1$) and lighter layers with larger $I_I$ values (e.g., $I_I = 3$). This enables balanced parallelism across the architecture.
- **POOL Blocks:** These modules reduce the spatial resolution of feature maps and act as intermediaries between convolutional layers. Implemented using max-pooling or average-pooling strategies, they perform streaming-friendly reduction operations without introducing pipeline stalls.
- **Fully Connected (FC) Block:** Placed at the end of the CNN, this module aggregates the final features and produces classification scores. It is also synthesised with loop pipelining to support high-throughput inference.

As shown in the Figure 5.1, the outputs of one module feed directly into the next without requiring external memory access, establishing an actual dataflow execution pattern.

### 5.2.4 Pipeline-Based Dataflow Execution

The visual layout in Figure 5.1 reflects the pipelined dataflow execution model of the architecture. Data enters from the left through the DRAM, flows sequentially across a series of layer blocks (`CONV` $\rightarrow$ `POOL` $\rightarrow$ `CONV` $\rightarrow$ ... `POOL` $\rightarrow$ `FC`), and exits through the output path to the host.

Each layer begins processing when sufficient input becomes available, enabling overlapped execution across stages. Intermediate values are streamed directly between layers with minimal buffering, avoiding the need for global memory access between stages. This pipeline allows multiple tiles or frames to be processed concurrently, increasing hardware utilisation and system throughput. Layers with high computational load (e.g., Conv3 with 40 filters) are assigned tighter pipeline initiation intervals, whereas less demanding layers use more relaxed configurations to conserve resources.

### 5.2.5 Summary of Key Design Principles

This design demonstrates a robust and scalable deployment framework. Leveraging a modular HLS-based architecture, the system facilitates the seamless integration of

entire CNN models as monolithic IP blocks, eliminating the need for extensive structural modifications. Since the CNN is synthesised via HLS tools, the architecture allows rapid adaptation to different network topologies or updated models by simply regenerating the HLS code. This reconfigurability accelerates the development cycle and supports maintainability across evolving application domains. Moreover, its compatibility with a wide range of FPGA platforms enhances portability. As a result, the proposed architecture provides a reusable and flexible deployment template for high-performance CNN inference under diverse operational and resource-constrained conditions. The key principles that underpin this design are summarised as follows:

- Efficient 64-bit PCIe communication using packed 16-bit fixed-point data.
- Modular IP structure for layer-wise synthesis and optimisation.
- Suitability Score-guided pipelining for resource-performance trade-offs.
- Streaming architecture with AXI interfaces for continuous layer-to-layer dataflow.
- Fully synchronised control logic for DMA coordination and pipeline execution.

## 5.3 Hardware Integration of the CNN Accelerator

This section presents the low-level hardware implementation of the proposed CNN accelerator. The design is deployed and tested on the AMD Kintex™ UltraScale+™ KCU116 FPGA platform in the context of oesophageal cancer detection from RGB endoscopic images. This application provides a realistic environment to evaluate the performance and feasibility of the previously introduced optimisation strategies.

Following the architectural principles established in Section 5.2, this chapter transitions from design abstraction to concrete FPGA implementation. Key design aspects include HLS-based synthesis, RTL-level IP core generation, Vivado block-level integration, and PCIe-based host communication.

The focus is on bridging the gap between simulation-level design and physical deployment by demonstrating how the proposed techniques, including adaptive pipelining, modular interfacing, and quantisation-aware scheduling, can be realised on actual FPGA hardware for a practical and resource-constrained task.

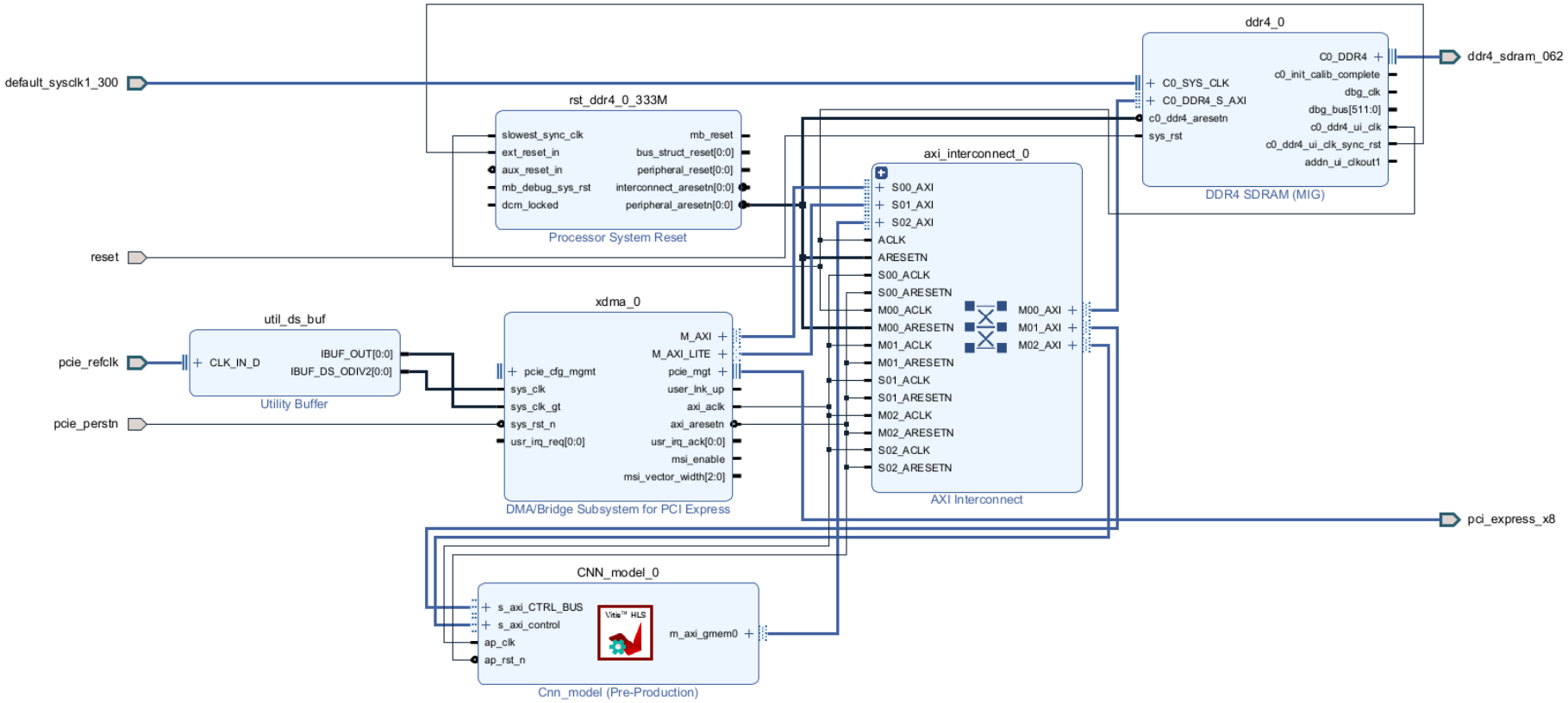



FIGURE 5.2: Vivado IP block design of the FPGA platform illustrating PCIe communication, AXI interconnection, and external DDR4 memory integration for deploying the CNN accelerator.

### 5.3.1 FPGA Platform Integration and Accelerator Interfacing

Figure 5.2 illustrates the Vivado block design, created using the Vivado IP Integrator, for deploying the CNN accelerator onto the AMD Kintex™ UltraScale+™ KCU116 FPGA platform. The architecture integrates key IP blocks and memory subsystems to support high-throughput inference and low-latency communication between the host and the FPGA.

At the core of the communication infrastructure lies the **XDMA** IP core, which facilitates high-throughput PCIe-based data transfer between the host and the FPGA fabric. The XDMA module is configured for 64-bit memory-mapped access, supporting streaming and burst transactions. System-level timing and synchronisation are managed by utility buffer (`util_ds_buf`) and clock/reset generation modules.

Memory access is managed by a **DDR4 Memory Interface Generator (MIG)**, connected through a high-performance AXI interconnect. This enables off-chip buffering of input data, intermediate feature maps, and output values, thus decoupling computation from data movement delays and host bandwidth limitations.

The CNN model (`CNN_model`), which encapsulates the HLS-generated CNN accelerator, is interfaced with the rest of the platform using AXI protocols. Specifically, AXI4-Stream is used for high-speed data paths, while AXI4-Lite enables register-level control and status monitoring. This separation between data and control channels ensures efficient pipelined execution and minimal contention.

In summary, the Vivado block design reflects a scalable and modular hardware-software co-design. It allows the CNN accelerator to be integrated into a heterogeneous computing environment with minimal reconfiguration while preserving performance and ensuring compatibility with diverse CNN topologies. Building on this system-level hardware foundation, the following section details how individual modules interact via standardized AXI protocols to ensure cohesive operation.

### 5.3.2 Memory Access Coordination

Efficient memory access is critical for sustaining high-throughput inference on FPGA platforms. The proposed architecture adopts a structured memory access strategy in which input images and intermediate feature maps are transferred through AXI master ports over the DDR interface, orchestrated by a DMA engine. This design choice ensures systematic memory coordination across all pipeline stages, which is vital for predictable and sustained throughput.

Due to the large input data size and intermediate activations in convolutional neural networks, DDR memory is used instead of streaming interfaces. While AXI-Stream

can offer lower latency for small, sequential data flows, it becomes impractical for bulk data movement in high-resolution medical images or deep CNNs with multiple layers. Thus, DDR enables batch-wise access to large datasets, ensuring the accelerator's scalability for complex inference tasks. Furthermore, DDR usage enables layer-wise decoupling, allowing greater flexibility in managing multi-stage processing pipelines

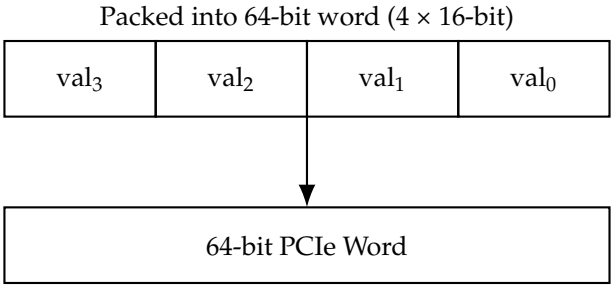



FIGURE 5.3: Packing four 16-bit values into one 64-bit word enables full PCIe bus utilisation and minimises data transfer cycles

However, DDR-based access introduces two key challenges: increased memory latency due to external access cycles, and potential bandwidth contention when multiple modules simultaneously interact with the DDR interface. In the case of memory latency, each DDR transaction involves a fixed access latency, often in the range of several clock cycles, which can accumulate when accessing large volumes of feature map data. Bandwidth contention occurs when multiple AXI master ports—such as those from parallel CNN layers or DMA engines—attempt concurrent access to the shared DDR memory, resulting in arbitration delays and reduced effective throughput. The architecture aligns all memory transfers to 64-bit word boundaries to address these limitations. Since the internal data format uses a 16-bit fixed-point representation, four values are packed into each 64-bit PCIe word, as illustrated in Figure 5.3. This alignment minimises the number of required memory transactions, reduces overhead, and maximises throughput.

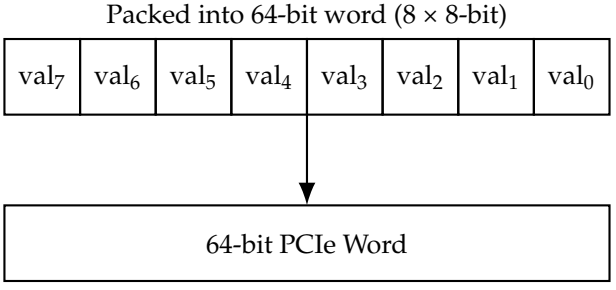



FIGURE 5.4: Packing eight 8-bit values improves bandwidth efficiency and supports quantised inference with minimal changes to the memory structure.

To support multiple precision configurations, Figure 5.4 illustrates a similar strategy for 8-bit, where eight values are packed into a single 64-bit PCIe word. This approach increases data packing density and reduces memory bandwidth requirements, making it ideal for resource-constrained or high-throughput systems. Accordingly, the proposed architecture is designed to support 16-bit and 8-bit fixed-point representations with minimal structural modifications, offering flexibility across deployment scenarios.

Although transitioning to 8-bit precision improves memory efficiency and bandwidth utilisation, it may lead to reduced numerical accuracy depending on the model's sensitivity to quantisation. While 8-bit quantisation can deliver considerable gains in memory usage and throughput, it should be applied cautiously, particularly in applications where classification accuracy is critical. This makes 8-bit inference particularly attractive for edge deployment scenarios where computational resources, power consumption, and memory footprint are highly constrained. Therefore, a careful evaluation of model performance under both 8-bit and 16-bit configurations is recommended, allowing designers to make informed trade-offs between accuracy and efficiency based on the target application's requirements.

All intermediate feature maps are stored in external DDR memory between consecutive layers. This memory-based decoupling simplifies the overall system design by allowing each CNN layer to operate independently without requiring tightly coupled inter-layer handshaking. It also promotes modularity and resource reuse, as the same memory space can be repurposed across different stages of computation.

While using DDR for intermediate buffering introduces additional latency due to read-/write cycles, it offers advantages in control synchronisation and design scalability. Specifically, external buffering enables clear separation between the compute and memory phases, simplifying the scheduling logic and supporting multi-stage pipeline designs with minimal interdependence.

Loop pipelining is applied within each CNN layer to minimise the latency overhead associated with DDR access, ensuring that computational throughput remains high. Furthermore, memory access patterns are carefully optimised to reduce stalls and improve bandwidth utilisation, preserving the system's overall inference efficiency even under bandwidth-limited conditions. In parallel with efficient memory access, control synchronization ensures that data movement and computation are correctly aligned across processing stages.

### 5.3.3 Control Synchronization and Address Mapping

In the proposed FPGA-based CNN accelerator, control coordination between the host system and the hardware modules is implemented via AXI4-Lite interfaces, enabling

low-latency synchronization without excessive resource overhead. The proposed design encapsulates the entire CNN inference process within a single IP core and employs a centralized control scheme managed by the host CPU.

The control interface exposes key signals—`ap_start`, `ap_done`, `ap_idle`, and `ap_ready`—which govern the execution of the accelerator. These signals are mapped to memory-mapped control registers, allowing software to initiate the computation pipeline, poll status flags, and retrieve outputs. To enhance responsiveness, an interrupt mechanism is also supported, whereby the accelerator notifies the host via an interrupt flag upon completion of the inference task, eliminating the need for constant polling.

Table 5.1 summarizes the key control registers used in this architecture. It includes fields for execution control, interrupt handling, address specification, and output validation. These registers are accessed via AXI4-Lite transactions initiated by the host.

TABLE 5.1: AXI4-Lite register map and memory-mapped control interface for the CNN accelerator.

| **Address Offset** | **Register / Description** |
|---|---|
| 0x00 | Control Signals (`ap_start`, `ap_done`, `ap_idle`, `ap_ready`) |
| 0x04 | Global Interrupt Enable Register |
| 0x08 | IP Interrupt Enable Register |
| 0x0C | IP Interrupt Status Register |
| 0x10 | `Output_Value` |
| 0x14 | `Output_Value_ap_vld` (valid signal) |
| 0x00010010 | `Input_Image[31:0]` |
| 0x00010014 | `Input_Image[64:32]` |
| 0x80000000 | DDR Memory Base Address (AXI full interface) |

A dedicated 64-bit input address pointer indicates the base memory location of the input image in DDR. As AXI4-Lite operates with a 32-bit width, this pointer is split across two separate registers for the lower and upper halves. This mechanism allows flexible support for large image sizes and deep feature maps stored externally in DDR memory.

Control synchronization is tightly coupled with the internal pipelining logic of the accelerator. Each layer processes its input independently, but control signals ensure that data dependencies across layers are respected. By synchronising control signals with the pipelined stages, the design achieves deterministic timing and real-time responsiveness, which are essential for edge-oriented CNN deployment.

### 5.3.4 Host–FPGA Communication via PCIe and DMA

The communication between the host system and the FPGA accelerator is established over a PCIe Gen3 x8 interface using Xilinx's XDMA IP core. Three types of interface endpoints are utilized to enable efficient data exchange and control:

- **H2C (Host-to-Card)**: Used for writing input image data from the host to the FPGA's external DDR memory. This transfer is performed via burst DMA operations aligned to 64-bit boundaries to ensure optimal PCIe throughput.
- **C2H (Card-to-Host)**: Reserved for reading data from the FPGA back to the host, such as classification results or debug output. While not required for all applications, this channel enables result verification or feedback in closed-loop systems.
- **User (MMIO)**: A memory-mapped I/O interface used for lightweight control signaling. Through this interface, the host can configure registers—such as writing 64-bit input base addresses, asserting the `ap_start` signal to initiate computation, and polling flags like `ap_done` and `ap_idle` to monitor execution status.

During execution, the host transfers the input data via the H2C interface into a reserved DDR memory address. Using the user interface, it then writes this base address to the control registers. Once the computation is triggered, the FPGA processes the data, and the result is read from a predefined register via the same user path or optionally via C2H if large data retrieval is necessary.

This hybrid communication model enables high-throughput data ingestion while retaining low-latency control signalling. It also supports real-time deployment scenarios where deterministic behaviour is critical.

### 5.3.5 Precision Configuration in FPGA-Based CNN Accelerators

This design utilises a 16-bit fixed-point representation to strike a balance between numerical accuracy and hardware efficiency. Compared to 32-bit floating-point arithmetic, fixed-point formats drastically reduce DSP utilisation and allow for deeper pipelining within computational layers. Moreover, the 16-bit format aligns naturally with the 64-bit PCIe interface, enabling four values to be packed into each transfer word—thereby maximising throughput during memory operations.

The fixed-point data type used in this work follows the `ap_fixed<16,8>` format, where the total word length is 16 bits, and 8 bits are allocated to the integer part, including the sign bit. This configuration leaves 8 bits for the fractional part, offering a practical trade-off between dynamic range and precision. It supports efficient arithmetic operations while ensuring sufficient accuracy for typical CNN inference workloads.

Although 8-bit quantisation further improves memory efficiency and throughput, allowing up to eight values per PCIe word, it may introduce notable accuracy degradation. The accelerator was synthesised and tested using an 8-bit configuration in preliminary experiments. While resource consumption and data transfer rates improved, inference accuracy decreased, particularly in early convolutional layers, which are more sensitive to quantisation noise.

All input images were normalised to the $[0, 1]$ range during training and testing, using a 32-bit floating-point representation. This preprocessing step improves the model's resilience to quantisation effects. Confining the input dynamic range to a well-bounded interval mitigates the risk of numerical saturation, overflow, or rounding errors during fixed-point computation. Consequently, even when weights and biases are quantised to 16 or 8 bits, the classification accuracy remains relatively stable compared to the full-precision baseline.

In proposed design, convolution weights and biases are quantised with three decimal digits of precision (e.g., 0.123), ensuring sufficient dynamic range for network expressiveness. This choice strikes a balance between arithmetic resolution and hardware simplicity, as more fractional digits would require additional bits and increase DSP resource usage. Empirical evaluations showed that this level of quantisation preserves inference accuracy while minimising hardware overhead. Nonetheless, the trade-off between quantisation depth and representational accuracy remains an open parameter, and deeper precision may be explored in future optimisation stages.

## 5.4 Evaluation and Results

This section evaluates the proposed FPGA-based CNN accelerator, emphasising key metrics for real-time deployment scenarios. The analysis focuses on inference latency, classification accuracy, throughput, power consumption, and hardware resource utilisation under varying numerical precision settings.

Specifically, 16-bit and 8-bit fixed-point implementations are compared to assess the trade-offs between numerical precision and system-level efficiency. The evaluation explores how quantisation levels, data representations, and architectural choices affect performance—including processing speed, resource utilisation, and energy efficiency.

Furthermore, the accelerator's power efficiency is assessed using Xilinx Vivado's post-implementation power analysis tools. Power estimates were obtained for idle and active inference conditions to offer insight into the viability of different configurations under resource-constrained environments.

### 5.4.1 Dataset Description and Selection Criteria

The optimised AlexNet architecture, trained using the TensorFlow framework with the Keras API in Python, was evaluated on a dataset of 1,000 randomly selected test images. These images were selected from the dataset described in Section 3.4.1, specifically from a held-out validation set not used during training to ensure unbiased evaluation. The subset was balanced, comprising 500 disease-positive and 500 healthy control samples. All images were resized to 224×224 pixels and normalised according to the CNN's pre-processing pipeline.

The trained model was deployed on an FPGA using fixed-point quantisation and adaptive pipelining strategies. 16-bit and 8-bit fixed-point implementations were derived directly from the pre-trained 32-bit floating-point model without retraining or fine-tuning after quantisation. Static quantisation was applied to reduce bitwidths while preserving the original trained weights. This setup accurately evaluates inference performance and resource usage under realistic deployment conditions.

### 5.4.2 Experimental Setup

All experiments were conducted on a workstation equipped with an $11^{th}$ Gen Intel Core i7-11700K CPU running at 3.60 GHz, featuring eight cores and 16 threads, supported by 32 GB of RAM and an NVIDIA GeForce RTX 2070 GPU. The host application was developed in C++ using Microsoft Visual Studio 2022.

The FPGA implementation was deployed and tested on an AMD Kintex UltraScale+ KCU116 evaluation board. The accelerator was synthesised using Xilinx Vivado and Vitis HLS version 2023.2. Communication between the host and the FPGA was established via a PCIe Gen3 x8 interface using the XDMA IP core.

### 5.4.3 Classification Accuracy Evaluation

This section evaluates the impact of numerical precision on the classification accuracy of the proposed CNN accelerator deployed on FPGA. To quantify any potential loss in inference quality, both 16-bit and 8-bit fixed-point configurations were tested and compared to the original floating-point model.

Classification accuracy was assessed as the percentage of correctly predicted labels from the total test samples. The model was evaluated under three numerical precision configurations: 32-bit floating-point, 16-bit fixed-point, and 8-bit fixed-point. Each version was tested using the same validation dataset, which was excluded from the training phase to ensure unbiased evaluation.

Table 5.2 presents the classification accuracy achieved under each precision setting. The results show that the 16-bit fixed-point implementation achieves 97.45% accuracy, closely matching the 32-bit floating-point baseline of 97.50%. This negligible 0.05% drop confirms that 16-bit quantisation preserves model performance while reducing hardware complexity.

TABLE 5.2: Classification accuracy comparison across different numerical precisions.

| Precision | Accuracy (%) | Relative Drop (%) |
|---|---|---|
| 32-bit Floating-point | 97.50 | - |
| 16-bit Fixed-point | 97.45 | 0.05 |
| 8-bit Fixed-point | 58.12 | 39.38 |

In contrast, the 8-bit fixed-point configuration results in a notable accuracy reduction, achieving only 58.12%. This degradation is primarily attributed to quantisation errors, particularly affecting earlier convolutional layers which are more sensitive to reduced numerical resolution. While 8-bit quantisation is beneficial for resource-constrained applications due to its lower memory and computational demands, it may not be suitable for tasks requiring high classification reliability.

Beyond overall classification accuracy, sensitivity and specificity are required evaluation metrics for medical image analysis, particularly in cancer detection tasks. For the 16-bit fixed-point implementation, the model achieved a **sensitivity of 98.75%** and a **specificity of 96.50%**. These results indicate the model's strong ability to correctly identify cancerous samples while minimising false negatives. This level of performance affirms the suitability of 16-bit quantisation not only in terms of hardware efficiency, but also in maintaining diagnostic reliability.

This evaluation highlights the effectiveness of 16-bit fixed-point arithmetic for FPGA-based CNN deployment, offering a compelling trade-off between accuracy and hardware efficiency. Subsequent sections provide further details on latency and resource utilisation.

### 5.4.4 Resource Utilisation Analysis

To assess the hardware efficiency of the proposed FPGA-based CNN accelerator, the utilisation of essential resources—including LUTs, FFs, BRAMs, and DSP slices—was analysed using Xilinx Vivado's post-synthesis reports. Table 5.3 presents the resource utilisation breakdown for key hardware modules: the CNN model IP, the DDR4 memory interface (MIG), and the XDMA PCIe communication core.

While earlier synthesis via Vitis HLS provided an initial estimate of resource consumption for the CNN module in isolation, this Vivado-level synthesis reflects the actual

usage after full system integration. The Vivado design encompasses not only the CNN model core but also memory controllers, communication infrastructure, and supporting IP blocks, offering a more complete and accurate assessment of resource requirements.

TABLE 5.3: Resource utilisation summary of core IP modules and Accelerator

| Module | LUTs | FFs | BRAMs | DSPs |
|---|---|---|---|---|
| CNN Model | 40,337 | 8,276 | 125.5 | 85 |
| DDR4 Controller (MIG) | 10,962 | 12,360 | 25.5 | 3 |
| XDMA PCIe Core | 41,366 | 43,182 | 74 | 0 |
| **CNN Accelerator** | **95,788** | **80,648** | **225** | **88** |
| **CNN Accelerator Utilisation (%)** | **44.15** | **18.59** | **46.88** | **4.82** |

Auxiliary components, such as AXI interconnects, clocking wizards, and reset modules, are not listed individually in the Table 5.3; however, their contributions are accounted for in the "CNN Accelerator" entry. This row represents the overall synthesis of the complete system.

As anticipated based on the computational structure of the CNN, DSP slices are primarily utilised by the convolutional layers, which dominate arithmetic operations. In contrast, the MIG and XDMA cores contribute more to logic and BRAM usage due to memory buffering and data-handling requirements. This differentiated usage pattern highlights a well-partitioned architecture that leverages domain-specific strengths of FPGA resources.

The reported utilisation percentages are calculated relative to the total available resources on the AMD Xilinx KCU116 FPGA, which provides 216,960 LUTs, 433,920 FFs, 480 BRAMs, and 1,824 DSP slices. The Table 5.3 shows that the proposed design consumes less than 50% of any single resource category, demonstrating a well-optimised and compact architecture suitable for resource-constrained environments.

Overall, the results affirm the viability of deploying the proposed accelerator on a mid-range FPGA platform such as the AMD Xilinx KCU116. The analysis also provides a critical baseline for exploring scalability in future designs.

### 5.4.5 Performance Comparison Across Platforms and Configurations

This section presents a comparative evaluation of inference performance across multiple hardware configurations and execution paradigms to comprehensively assess the effectiveness of the proposed FPGA-based CNN accelerator. The analysis spans GPU

and FPGA platforms, considering variations in model structure, parallelisation strategy, and deployment context. The objective is to elucidate the performance trade-offs between conventional GPU-based processing and custom FPGA implementations, particularly under edge-oriented constraints.

The following four configurations were evaluated:

- **GPU (NVIDIA GeForce RTX 2070, S-score Optimised)**: An optimised version of AlexNet was used, incorporating structural pruning and quantisation guided by S-score analysis. This model was executed using TensorFlow in an optimised configuration, representing a software-level deployment approach commonly adopted in GPU-based inference setups.
- **FPGA (Inter-Layer Pipeline Execution)**: In this configuration, the optimised model was deployed on the FPGA, explicitly enabling inter-layer pipelining. This strategy overlaps the execution of consecutive CNN layers by allowing one layer to start processing as soon as partial results become available from the previous layer. This configuration enhances throughput and more effectively utilises FPGA resources by leveraging fine-grained parallelism across layers and minimising idle cycles.

The performance comparison focuses on three key metrics: throughput (images processed per second), power and hardware efficiency. All FPGA-based results were obtained through real hardware deployment on the AMD Xilinx KCU116 board. However, tower consumption was estimated using post-implementation static power analysis provided by Xilinx Vivado 2023.2. For GPU-based results, inference times were recorded using TensorFlow with batch size set to 1, and power measurements were taken using NVIDIA System Management Interface (`nvidia-smi`) while isolating GPU activity to minimise background interference. Each configuration was tested over 10 independent runs (N=10), and the reported metrics represent the average values across these runs to ensure statistical robustness.

TABLE 5.4: Performance and energy efficiency comparison between GPU and FPGA implementations of AlexNet.

| **Platform** | **Throughput (images/sec)** | **Power (W)** | **Efficiency (images/sec/W)** |
|---|---|---|---|
| NVIDIA GeForce RTX 2070 | 51.62 | 11.82 | 4.36 |
| Xilinx KCU116 | 76.19 | 8.18 | 9.38 |

Table 5.4 highlights the performance and energy efficiency differences between the proposed FPGA-based accelerator and a conventional GPU-based deployment of the same optimised AlexNet model. While the GPU (NVIDIA GeForce RTX 2070) achieved a throughput of 51.62 images per second, the FPGA implementation outperformed it by

achieving 76.19 images per second—representing an improvement of approximately **47.6%** in processing speed. This gain is particularly notable considering that the FPGA operates at a lower clock frequency and under tighter resource constraints compared to the GPU.

In terms of power consumption, the FPGA consumes only **8.18 W**, compared to the GPU's **11.82 W**, resulting in a substantially better energy efficiency. The proposed FPGA design achieved an energy efficiency of **9.38 images/sec/W**, more than **2.15 times higher** than the GPU's **4.36 images/sec/W**.

These results demonstrate the advantages of the proposed hardware-software co-design approach in targeting energy-constrained environments. By combining structured model optimisations with layer-specific hardware parallelism strategies, the FPGA accelerator achieves superior throughput and energy efficiency without sacrificing inference accuracy. This makes it a highly viable candidate for real-time CNN applications, where power budgets and form factor constraints are critical.

## 5.5 Comparison with Existing Works

This section analyses the proposed FPGA-based CNN accelerator in comparison with existing FPGA implementations of the AlexNet architecture. The comparison covers several key metrics: inference latency, throughput (images processed per second), resource utilisation, power consumption, and classification accuracy. While hardware efficiency is essential for deployment under real-time and resource-constrained conditions, maintaining high inference accuracy remains equally critical, especially in medical image analysis tasks such as cancer detection.

The evaluation of the proposed accelerator is based on real hardware deployment using the AMD Kintex™ UltraScale+™ KCU116 FPGA platform with PCIe-based communication to a host system. The experimental setup described in Section 5.4 is also applicable to this comparison.

Table 5.5 provides a comparative overview of FPGA-based implementations of the AlexNet architecture, examining performance across key dimensions including inference throughput, accuracy, power consumption, and resource utilisation. A critical insight from this comparison is the superior balance achieved by the proposed design between classification accuracy and inference speed, particularly in realistic deployment conditions with RGB inputs ($224 \times 224 \times 3$).

TABLE 5.5: Comparison with AlexNet FPGA Implementations

| Metric | Liang et al. [46] | Zhang et al. [90] | Li et al. [91] | Vestias et al. [92] | Wang et al. [93] | Neelam et al. [94] | Zayed et al. [120] | Proposed Approach |
|---|---|---|---|---|---|---|---|---|
| Device | Xilinx ZCU102 | Xilinx ZCU104 | Intel Arria 10 | Zynq-7020 | Zynq-7045 | 5×XC7A200T | PYNQ-Z2 | Xilinx KCU116 |
| Precision | 16-bit fixed | 8-bit | 8–16-bit | 8-bit fixed | 8-bit fixed | 32-bit floating | – | 16-bit fixed |
| Freq (MHz) | 200 | 300 | 200 | 200 | 200 | 100 | – | 137 |
| Throughput (img/s) | – | 9.73 | – | 229 | 1369.6 | 15 | ˜2.16 | **76.19** |
| Throughput (GOPS) | **1006.4** | 14.11 | 1.46 | 332 | 493 | 56 | – | 5.16 |
| Power (W) | 23.6 | 17.67 | 27.2 | – | 4.2 | – | – | 8.12 |
| Efficiency (img/s/W) | – | 0.55 | – | – | **325.3** | – | – | 9.38 |
| LUT Usage | 600,000 | 101,953 | 360,000 | 46,914 | 105,673 | 490,200 | – | 95,788 |
| DSP Usage | 2520 | 696 | 410 | 212 | 880 | **11,820** | – | 88 |
| FF Usage | – | 127,577 | 523,700 | – | 94,149 | – | – | 80,648 |
| BRAM Usage | 1824.8 | 198.5 | 1,366 | 126 | 463 | 1.64 Mb | – | 225 |
| Accuracy (%) | – | **99.1** | – | 54.7 | 49.3 | – | 98.33 | **97.45** |
| Input Size | 224×224 | 28×28 | – | 224×224×3 | 224×224×3 | 224×224 | 92×112 | 224×224×3 |

*Note*: "–" indicates that the corresponding information was not explicitly reported in the original study.

It is important to note that the reported accuracy values across existing studies may not be directly comparable due to differences in datasets. However, the comparison considers top-1 classification accuracy as a reference point to provide an indicative understanding of inference reliability. This is motivated by the overarching design goal of achieving high inference speed and robust predictive performance, particularly in applications where classification fidelity is critical.

Among the listed implementations, only a limited number report both high classification accuracy and utilise full-resolution inputs. For instance, Zayed et al. report the classification accuracy of 98.33%, but their system achieves a throughput of only 2.16 images/s, limiting its applicability in real-time scenarios. In contrast, the proposed accelerator delivers a comparable accuracy of **97.45%** while sustaining a throughput of **76.19 images/s**, amounting to a **35.3**$\times$ increase in processing speed with minimal reduction in prediction quality.

While Zhang et al. report the highest classification accuracy of 99.1%, this result is obtained using an input resolution of $28 \times 28$, which is lower than typical input sizes used in contemporary CNN deployments. Smaller input dimensions reduce the computational workload and may simplify the classification task. In contrast, the proposed accelerator is evaluated on full-resolution $224 \times 224 \times 3$ images, consistent with the original AlexNet configuration and more representative of practical use cases such as medical image analysis.

Among the listed implementations, Vestias et al. and Wang et al. employ the same input resolution ($224 \times 224 \times 3$) as the proposed design, enabling a fair comparison in terms of practical deployment settings. Although these works report higher throughput values—229 images/s and 1369.6 images/s, respectively—their classification accuracies are notably low, at only 54.7% and 49.3%. This substantial drop in accuracy may stem from aggressive quantisation strategies, as both designs utilise 8-bit fixed-point arithmetic. In contrast, the proposed accelerator was initially tested using 8-bit precision but exhibited degraded accuracy. Therefore, it was implemented with 16-bit fixed-point precision to preserve inference reliability. As a result, the proposed design achieves a throughput of 76.19 images/s while maintaining a 97.45% accuracy, offering a more balanced trade-off suitable for medical image analysis tasks, where predictive performance is as critical as execution speed.

Liang et al., Li et al., and Neelam et al. do not report classification accuracy, which limits a comprehensive evaluation of their applicability in tasks requiring reliable inference. Nevertheless, throughput values provide a basis for assessing computational speed. Liang et al. report an impressive arithmetic throughput of 1006.4 GOPS but do not include image-level throughput, making practical comparisons difficult. Li et al. achieve only 1.46 images/s, reflecting limited real-time viability. Neelam et al. use 32-bit floating-point precision and an exceptionally high number of DSPs (11,820), yet

achieves only 15 images/s, indicating low resource efficiency. By contrast, the proposed approach delivers 76.19 images/s using 16-bit fixed-point arithmetic and just 88 DSPs, offering higher throughput with substantially fewer hardware resources, highlighting improved efficiency in resource utilisation. These comparisons further highlight the effectiveness of the proposed accelerator in balancing computational throughput with hardware efficiency, even when classification accuracy is unavailable for baseline references.

The proposed accelerator achieves high efficiency with modest requirements regarding hardware resource usage. It delivers a throughput of 76.19 images/s using only 88 DSPs, 95,788 LUTs, 80,648 FFs, and 225 BRAMs. In comparison, Liang et al. use 2520 DSPs and 600,000 LUTs, but do not report image-level throughput, making it hard to assess practical performance. Neelam et al. use an extremely high number of 11,820 DSPs and 1.64 Mb of BRAM, yet reach only 15 images/s, showing low resource efficiency. Li et al. achieve just 1.46 images/s while consuming 523,700 FFs, which is more than six times the FFs used in proposed design. Zhang et al. process 9.73 images/s, but with a low input resolution of 28 × 28 pixels. These results show that the proposed design achieves better performance regarding hardware consumption, making it suitable for efficient FPGA-based deployment.

Cancer detection requires high inference accuracy, support for full-resolution input images, and low-latency operation to be viable in real-time clinical settings. The proposed accelerator meets these criteria by achieving 97.45% classification accuracy on 224 $\times$ 224 $\times$ 3 inputs, while delivering an inference rate of 76.19 images per second. Compared to prior works that report similar accuracy levels—such as Zayed et al. (98.33%) and Zhang et al. (99.1%)—the proposed design processes images **35.3**$\times$ and **7.8**$\times$ faster, respectively. This demonstrates a strong balance between inference speed and predictive quality. Unlike designs that trade accuracy or resolution for high throughput, this work offers a robust and efficient solution tailored to real-world medical image analysis demands.

## 5.6 Summary

The primary objective of this chapter was to demonstrate the practical deployment of the optimised CNN accelerator on real FPGA hardware, validating the architectural and optimisation techniques developed in earlier chapters. The proposed system achieved high inference accuracy, low latency, and efficient resource usage on the AMD Kintex™ UltraScale+™ KCU116 FPGA platform. Specifically, the accelerator performed a classification accuracy of 97.45% on RGB endoscopic images, closely aligning with clinical deployment requirements for tasks such as cancer detection.

Experimental results confirmed the system's ability to deliver 76.19 images/sec throughput while consuming only 8.18 W of power, yielding an energy efficiency of 9.38 images/sec/W. Compared to a GPU-based baseline (NVIDIA RTX 2070), this represents a 47.6% improvement in processing speed and more than 2.15× higher energy efficiency. Furthermore, the design achieved these results using only 88 DSPs and 95,788 LUTs, demonstrating strong performance even under tight hardware constraints.

Comparison with previous FPGA-based implementations of AlexNet shows that the proposed design achieves high throughput while maintaining high classification accuracy. It processes up to 35.3 times more images per second than other designs that report similar accuracy levels. It achieves up to 7.8 times higher throughput than models using reduced input sizes. These results indicate that the proposed accelerator offers a strong balance between inference speed, predictive reliability, and efficient hardware usage, making it suitable for real-time image analysis tasks such as cancer detection.

In summary, the experimental evidence presented in this chapter confirms that the proposed FPGA-based CNN accelerator is well-suited for medical image analysis applications that demand high accuracy, low latency, and efficient hardware utilisation. The system demonstrates reliable inference on high-resolution input data while maintaining resource and power efficiency. These results validate the practical viability of the proposed optimisation techniques and highlight their relevance for real-world deployment in critical scenarios such as cancer detection.

# Chapter 6

# Conclusions and Future Work

This chapter summarises the research conducted throughout the thesis, drawing together the key findings, methodological contributions, and experimental outcomes from each stage. It revisits the primary research objectives and evaluates the extent to which they have been achieved. In addition to consolidating the results, the chapter outlines potential future work directions, providing insights into how the proposed approaches can be further developed or extended to new application domains.

## 6.1 Conclusions

This chapter concludes the thesis by highlighting the three main stages of the work. These stages tackle the core challenges of achieving high-throughput, low-latency, and resource-efficient inference on FPGA platforms. The chapter reflects on how the methods developed throughout the thesis address the demands of real-time, accuracy-critical image analysis, with a particular focus on cancer detection as a representative application.

**Chapter 3** introduces a novel structural optimisation approach for convolutional neural networks, aimed at reducing computational cost while preserving model accuracy on FPGA platforms. This is achieved by introducing the Suitability Score, a metric that quantifies each convolutional layer's alignment with the computational characteristics of Winograd-based accelerators. By applying selective structural modifications based on this score, the chapter demonstrates reductions in MAC operations and training time, while preserving—and in some cases improving—inference accuracy. The proposed optimisation strategy is validated on two structurally distinct CNN models,

confirming its adaptability across different CNN architectures. These results demonstrate that the chapter successfully meets its objective of enabling efficient, hardware-conscious model design and establishes a solid foundation for the hardware-level strategies developed in Chapter 4.

**Chapter 4** presents a simulation-guided hardware optimisation strategy that enhances CNN deployment efficiency on FPGA platforms by enabling adaptive, layer-specific pipelining. Instead of applying static, uniform pipeline configurations, the proposed method dynamically adjusts initiation intervals based on each layer's computational demands and structural complexity. This workload-aware scheduling improves resource utilisation and computational throughput compared to fixed-II designs, as confirmed through experimental results on Winograd-optimised CNNs. The demonstrated reductions in DSP, LUT, and FF usage, with minimal latency overhead, validate the practical advantages of this approach. These results show that Chapter 4 effectively meets its objective of enabling scalable, resource-efficient inference through structured hardware-level optimisation, and lays a solid foundation for the system-level deployment strategies explored in Chapter 5.

In **Chapter 5**, the thesis moves from architectural and simulation-level optimisations to a complete hardware implementation, evaluating the practical application of the proposed CNN accelerator on an FPGA platform. Oesophageal cancer detection is used as a representative use case, due to its combination of accuracy, latency, and resource constraints. The implementation includes PCIe-based host communication, external DDR memory management, fixed-point quantisation, and modular control structures, forming an integrated system for real-time image inference. The evaluation results indicate that the accelerator supports high-throughput and low-latency operation while maintaining acceptable power and resource usage. Comparisons with GPU and FPGA-based implementations demonstrate that the design achieves a balanced trade-off between performance and efficiency. These results show that Chapter 5 meets its objective of assessing the practical deployment of the proposed optimisation techniques in realistic, resource-constrained image analysis tasks.

The cancer detection task provided a high-stakes, computation-intensive scenario that effectively evaluated the robustness, throughput capability, and overall performance of the proposed system. Although this thesis focused on oesophageal cancer detection as a representative case study, the architectural and optimisation techniques developed are not limited to the medical domain. They are designed to address common computational requirements in real-time image classification tasks, including high-resolution inputs, low inference latency, high throughput, and limited hardware resources. The design principles introduced in this work can be adapted to other domains with similar constraints. Potential application areas include industrial inspection, autonomous navigation, surveillance, and embedded vision systems, where accuracy, high-throughput, and energy efficiency are critical.

## 6.2 Future Work

This section outlines potential directions for future work inspired by the findings and limitations identified throughout this thesis. While the proposed framework demonstrates strong performance in real-time CNN deployment on FPGA platforms, several areas remain open for exploration. The suggestions presented here aim to enhance the scalability, flexibility, and generalisability of the current approach for wider adoption in real-world embedded vision systems.

- **Optimisation of Data Transfer and I/O Bottlenecks:** Although the current system utilises PCIe-based communication between the host CPU and the FPGA accelerator, further improvements in data transfer efficiency could improve throughput, reduce end-to-end inference latency, and minimise host-device communication overhead. Future work could explore the adoption of higher-bandwidth PCIe standards (e.g., PCIe Gen4 or Gen5) or investigate the integration of DMA engines with optimised transaction schemes. Such enhancements could reduce transfer-induced latency, minimise host-device synchronisation overheads, and further improve the throughput and responsiveness of real-time inference systems.
- **Extension to Real-Time Video Processing:** The present study focuses on static image inference tasks. A natural progression would involve extending the proposed framework to real-time video stream processing, where consecutive frames must be processed with stringent latency constraints. This would require addressing challenges such as frame buffering, temporal coherence, and maintaining inference throughput at video frame rates (e.g., 30 or 60 FPS). Successfully adapting the accelerator for video-based applications would broaden its applicability to fields such as endoscopic video analysis, surveillance, autonomous driving, and robotics.

These future directions would further enhance the scalability, and real-world applicability of the proposed FPGA-based CNN acceleration framework, supporting broader deployment across diverse, latency-critical embedded vision tasks.

# References

[1] R. Dey, A. Roy, J. Akter, A. Mishra, and M. Sarkar, "AI-Driven Machine Learning for Fraud Detection and Risk Management in US Healthcare Billing and Insurance," *Journal of Computer Science and Technology Studies*, vol. 7, no. 1, pp. 188–198, 2025.

[2] K. Jha, V. Velaga, K. Routhu, G. Sadaram, S. Boppana *et al.*, "Evaluating the Effectiveness of Machine Learning for Heart Disease Prediction in Healthcare Sector," *J Cardiobiol*, vol. 9, no. 1, p. 1, 2025.

[3] F. Krones, U. Marikkar, G. Parsons, A. Szmul, and A. Mahdi, "Review of Multimodal Machine Learning Approaches in Healthcare," *Information Fusion*, vol. 114, p. 102690, 2025.

[4] C. Chakraborty, M. Bhattacharya, S. Pal, and S.-S. Lee, "From Machine Learning to Deep learning: Advances of The Recent Data-driven Paradigm Shift in Medicine and Healthcare," *Current Research in Biotechnology*, vol. 7, p. 100164, 2024.

[5] I. Ahmad and F. Alqurashi, "Early Cancer Detection Using Deep Learning and Medical Imaging: A survey," *Critical Reviews in Oncology/Hematology*, vol. 204, p. 104528, 2024.

[6] P. Sharma, D. R. Nayak, B. K. Balabantaray, M. Tanveer, and R. Nayak, "A Survey on Cancer Detection via Convolutional Neural Networks: Current Challenges and Future Directions," *Neural Networks*, vol. 169, pp. 637–659, 2024.

[7] M. S. Alshuhri, S. G. Al-Musawi, A. A. Al-Alwany, H. Uinarni, I. Rasulova, P. Rodrigues, A. T. Alkhafaji, A. M. Alshanberi, A. H. Alawadi, and A. H. Abbas, "Artificial Intelligence in Cancer Diagnosis: Opportunities and Challenges," *Pathology-Research and Practice*, vol. 253, p. 154996, 2024.

[8] E. I. Obeagu and G. U. Obeagu, "Breast Cancer: A Review of Risk Factors and Diagnosis," *Medicine*, vol. 103, no. 3, p. e36905, 2024.

[9] F. M. Talaat, S. El-Sappagh, K. Alnowaiser, and E. Hassan, "Improved Prostate Cancer Diagnosis Using a Modified ResNet50-based Deep Learning Architecture," *BMC Medical Informatics and Decision Making*, vol. 24, no. 1, p. 23, 2024.

[10] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, and M. Parmar, "A Review of Convolutional Neural Networks in Computer Vision," *Artificial Intelligence Review*, vol. 57, no. 4, p. 99, 2024.

[11] I. H. Sarker, "Deep Learning: a Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions," *SN computer science*, vol. 2, no. 6, pp. 1–20, 2021.

[12] O. Elharrouss, Y. Akbari, N. Almadeed, and S. Al-Maadeed, "Backbones-review: Feature Extractor Networks for Deep Learning and Deep Reinforcement Learning Approaches in Computer Vision," *Computer Science Review*, vol. 53, p. 100645, 2024.

[13] X. Wei, W. Liu, L. Chen, L. Ma, H. Chen, and Y. Zhuang, "FPGA-Based Hybrid-Type Implementation of Quantized Neural Networks for Remote Sensing Applications," *Sensors*, vol. 19, no. 4, p. 924, 2019.

[14] Z. Liu, P. Chow, J. Xu, J. Jiang, Y. Dou, and J. Zhou, "A Uniform Architecture Design for Accelerating 2D and 3D CNNs on FPGAs," *Electronics*, vol. 8, no. 1, p. 65, 2019.

[15] W. Li, C. He, H. Fu, J. Zheng, R. Dong, M. Xia, L. Yu, and W. Luk, "A Real-Time Tree Crown Detection Approach for Large-Scale Remote Sensing images on FPGAs," *Remote Sensing*, vol. 11, no. 9, p. 1025, 2019.

[16] L. Chen, X. Wei, W. Liu, H. Chen, and L. Chen, "Hardware Implementation of Convolutional Neural Network-Based Remote Sensing Image Classification method," in *Communications, Signal Processing, and Systems: Proceedings of the 2018 CSPS Volume II: Signal Processing 7th*. Springer, 2020, pp. 140–148.

[17] L. Li, S. Zhang, and J. Wu, "Efficient Object Detection Framework and Hardware Architecture for Remote Sensing images," *Remote Sensing*, vol. 11, no. 20, p. 2376, 2019.

[18] B. J. Kang, H. I. Lee, S. K. Yoon, Y. C. Kim, S. B. Jeong, H. Kim *et al.*, "A survey of fpga and asic designs for transformer inference acceleration and optimization," *Journal of Systems Architecture*, vol. 155, p. 103247, 2024.

[19] N. Zhang, X. Wei, H. Chen, and W. Liu, "FPGA Implementation for CNN-Based Optical Remote Sensing Object Detection," *Electronics*, vol. 10, no. 3, p. 282, 2021.

[20] C.-E. Vasile, A.-A. Ulmămei, and C. Bîră, "Image Processing Hardware Acceleration—A Review of Operations Involved and Current Hardware Approaches," *Journal of Imaging*, vol. 10, no. 12, p. 298, 2024.

[21] Y. Xu, J. Luo, and W. Sun, "Flare: An FPGA-Based Full Precision Low Power CNN Accelerator with Reconfigurable Structure," *Sensors*, vol. 24, no. 7, p. 2239, 2024.

[22] F. Liu, H. Li, W. Hu, and Y. He, "Review of Neural Network Model Acceleration Rechniques Based on FPGA Platforms," *Neurocomputing*, p. 128511, 2024.

[23] C. Jiang, D. Ojika, B. Patel, and H. Lam, "Optimized FPGA-Based Deep Learning Accelerator for Sparse CNN Using High Bandwidth Memory," in *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2021.

[24] A. Hadj Fredj and J. Malek, "FPGA-Accelerated Anisotropic Diffusion Filter Based on SW/HW-Codesign for Medical Images," *Journal of Real-Time Image Processing*, vol. 18, no. 6, pp. 2429–2440, 2021.

[25] S. Mittal, "A Survey of FPGA-Based Accelerators for Convolutional Neural Networks," *Neural computing and applications*, vol. 32, no. 4, pp. 1109–1139, 2020.

[26] R. Ayachi, Y. Said, and A. Ben Abdelali, "Optimizing Neural Networks for Efficient FPGA Implementation: A Survey." *Archives of Computational Methods in Engineering*, vol. 28, no. 7, 2021.

[27] H. Hong, D. Choi, N. Kim, H. Lee, B. Kang, H. Kang, and H. Kim, "Survey of Convolutional Neural Network Accelerators on Field-Programmable Gate Array Platforms: Architectures and Optimization Techniques," *Journal of Real-Time Image Processing*, vol. 21, no. 3, p. 64, 2024.

[28] R. Gadea-Gironés, J. Fe, and J. M. Monzo, "Task Parallelism-Based Architectures on FPGA to Optimize The Energy Efficiency of AI at The Edge," *Microprocessors and Microsystems*, vol. 98, p. 104824, 2023.

[29] M. Cho and Y. Kim, "FPGA-Based Convolutional Neural network Accelerator with Resource-Optimized Approximate Multiply-Accumulate Unit," *Electronics*, vol. 10, no. 22, p. 2859, 2021.

[30] H.-A. Rashid, U. Kallakuri, and T. Mohsenin, "TinyM2Net-V2: A Compact Low-power Software Hardware Architecture for Multi m odal Deep Neural Networks," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 3, pp. 1–23, 2024.

[31] Y. Hou and Z. B. Chen, "LeNet-5 Improvement Based on FPGA Acceleration," *The Journal of Engineering*, vol. 2020, no. 13, pp. 526–528, 2020.

[32] P. Hobden, S. Srivastava, and E. Nurellari, "FPGA-based CNN for Real-time UAV tracking and Detection," *Frontiers in Space Technologies*, vol. 3, p. 878010, 2022.

[33] M. Garifulla, J. Shin, C. Kim, W. H. Kim, H. J. Kim, J. Kim, and S. Hong, "A Case Study of Quantizing Convolutional Neural Networks for Fast Disease Diagnosis on Portable Medical Devices," *Sensors*, vol. 22, no. 1, p. 219, 2021.

[34] Z. Wang, H. Li, X. Yue, and L. Meng, "Briefly Analysis About CNN Accelerator Based on FPGA," *Procedia Computer Science*, vol. 202, pp. 277–282, 2022.

[35] J. He, M. Zhang, J. Xu, L. Yu, and W. Li, "Optimizing CNN Hardware Acceleration with Configurable Vector Units and Feature Layout Strategies," *Electronics*, vol. 13, no. 6, p. 1050, 2024.

[36] R. T. Syed, Y. Zhao, J. Chen, M. Andjelkovic, M. Ulbricht, and M. Krstic, "FPGA Implementation of a Fault-Tolerant Fused and Branched CNN Accelerator With Reconfigurable Capabilities," *IEEE Access*, vol. 12, pp. 57 847–57 862, 2024.

[37] M. S. AZZAZ, A. MAALI, R. KAIBOU, I. KAKOUCHE, M. SAAD, and H. HAMIL, "FPGA HW/SW Codesign Approach for Real-Time Image Processing Using HLS," in *2020 1st International Conference on Communications, Control Systems and Signal Processing (CCSSP)*. IEEE, 2020, pp. 169–174.

[38] R. Sayed, H. Draz, and H. Azmi, "HW-SW Co-design of Image Classification Accelerator on FPGA: R. Sayed et al." *The Journal of Supercomputing*, vol. 81, no. 8, p. 941, 2025.

[39] A. Lavin and S. Gray, "Fast Algorithms for Convolutional Neural Networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4013–4021.

[40] P. Maji and R. Mullins, "On the Reduction of Computational Complexity of Deep Convolutional Neural Networks," *Entropy*, vol. 20, no. 4, p. 305, 2018.

[41] S. Bouguezzi, H. B. Fredj, T. Belabed, C. Valderrama, H. Faiedh, and C. Souani, "An Efficient FPGA-based Convolutional Neural Network for Classification: AdMobileNet," *Electronics*, vol. 10, no. 18, p. 2272, 2021.

[42] D. Klingelhöfer, Y. Zhu, M. Braun, D. Brüggmann, N. Schöffel, and D. A. Groneberg, "A World Map of Esophagus Cancer Research: a Critical Accounting," *Journal of translational medicine*, vol. 17, no. 1, p. 150, 2019.

[43] H. Fan, M. Ferianc, Z. Que, S. Liu, X. Niu, M. R. Rodrigues, and W. Luk, "FPGA-based Acceleration for Bayesian Convolutional Neural Networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5343–5356, 2022.

[44] S. M. Sait, A. El-Maleh, M. Altakrouri, and A. Shawahna, "Optimization of FPGA-based CNN Accelerators Using Metaheuristics," *The Journal of Supercomputing*, vol. 79, no. 4, pp. 4493–4533, 2023.

[45] D. Huang, X. Zhang, R. Zhang, T. Zhi, D. He, J. Guo, C. Liu, Q. Guo, Z. Du, S. Liu *et al.*, "DWM: A Decomposable Winograd Method for Convolution Acceleration," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 4174–4181.

[46] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 857–870, 2019.

[47] T.-H. Tsai, Y.-C. Ho, and M.-H. Sheu, "Implementation of FPGA-based Accelerator for Deep Neural Networks," in *2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2019, pp. 1–4.

[48] B. Wu, X. Wu, P. Li, Y. Gao, J. Si, and N. Al-Dhahir, "Efficient FPGA Implementation of Convolutional Neural Networks and Long Short-term memory for Radar Emitter Signal Recognition," *Sensors*, vol. 24, no. 3, p. 889, 2024.

[49] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based Accelerators of Deep Learning Networks for Learning and Classification: A Review," *ieee Access*, vol. 7, pp. 7823–7859, 2018.

[50] A. Nechi, L. Groth, S. Mulhem, F. Merchant, R. Buchty, and M. Berekovic, "FPGA-based Deep Learning Inference Accelerators: Where are We Standing?" *ACM Transactions on Reconfigurable Technology and Systems*, vol. 16, no. 4, pp. 1–32, 2023.

[51] H. Tong, K. Han, S. Han, and Y. Luo, "Design of a Generic Dynamically Reconfigurable Convolutional Neural Network Accelerator with Optimal Balance," *Electronics*, vol. 13, no. 4, p. 761, 2024.

[52] D. K. Lam, C. V. Du, and H. L. Pham, "QuantLaneNet: A 640-FPS and 34-GOPS/W FPGA-based CNN Accelerator for Lane Detection," *Sensors*, vol. 23, no. 15, p. 6661, 2023.

[53] S. Gdaim and A. Mtibaa, "Accelerating Convolutional Neural Networks on FPGA Platforms: a High-Performance Design Methodology Using OpenCL," *Journal of Real-Time Image Processing*, vol. 22, no. 2, p. 67, 2025.

[54] Z. Zhao, Y. Chen, P. Feng, J. Li, G. Chen, R. Shen, and H. Lu, "A High-Throughput FPGA Accelerator for Lightweight CNNs With Balanced Dataflow," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2025.

[55] A. Mhaouch, W. Gtifa, and M. Machhout, "FPGA Hardware Acceleration of AI Models for Real-Time Breast Cancer Classification," *AI*, vol. 6, no. 4, p. 76, 2025.

[56] S. Zhu, J. Gao, L. Liu, M. Yin, J. Lin, C. Xu, C. Xu, and J. Zhu, "Public Imaging Datasets of Gastrointestinal Endoscopy for Artificial Intelligence: a Review," *Journal of Digital Imaging*, vol. 36, no. 6, pp. 2578–2601, 2023.

[57] W. Dai and D. Berleant, "Benchmarking Contemporary Deep Learning Hardware and Frameworks: A Survey of Qualitative Metrics," in *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*. IEEE, 2019, pp. 148–155.

[58] J. Zheng, Z. Lv, D. Li, C. Lu, Y. Zhang, L. Fu, X. Huang, J. Huang, D. Chen, and J. Zhang, "FPGA-Based Low-Power High-Performance CNN Accelerator Integrating DIST for Rice Leaf Disease Classification," *Electronics*, vol. 14, no. 9, p. 1704, 2025.

[59] Y. Kim, H. Kim, N. Yadav, S. Li, and K. K. Choi, "Low-Power RTL Code Generation for Advanced CNN Algorithms Toward Object Detection in Autonomous Vehicles," *Electronics*, vol. 9, no. 3, p. 478, 2020.

[60] A. Sateesan, S. Sinha, S. KG, and A. Vinod, "A Survey of Algorithmic and Hardware Optimization Techniques for Vision Convolutional Neural Networks on FPGAs," *Neural processing letters*, vol. 53, no. 3, pp. 2331–2377, 2021.

[61] P. Babu and E. Parthasarathy, "Reconfigurable FPGA Architectures: A Survey and Applications," *Journal of The Institution of Engineers (India): Series B*, vol. 102, pp. 143–156, 2021.

[62] H. Zeng, R. Chen, C. Zhang, and V. Prasanna, "A Framework for Generating High Throughput CNN Implementations on FPGAs," in *Proceedings of the 2018 ACM/SIGDA international symposium on field-programmable gate arrays*, 2018, pp. 117–126.

[63] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels," in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*. IEEE, 2019, pp. 1–8.

[64] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, and Y. Xu, "Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 10, no. 3, pp. 1–23, 2017.

[65] Y. Tang, R. Dai, and Y. Xie, "Optimization of Energy Efficiency for FPGA-based Convolutional Neural Networks Accelerator," in *Journal of Physics: Conference Series*, vol. 1487, no. 1. IOP Publishing, 2020, p. 012028.

[66] M. Qasaimeh, K. Denolf, A. Khodamoradi, M. Blott, J. Lo, L. Halder, K. Vissers, J. Zambreno, and P. H. Jones, "Benchmarking Vision Kernels and Neural Network Inference Accelerators on Embedded Platforms," *Journal of Systems Architecture*, vol. 113, p. 101896, 2021.

[67] F. Yan, A. Koch, and O. Sinnen, "A Survey on FPGA-based Accelerator for ML Models," *arXiv preprint arXiv:2412.15666*, 2024.

[68] E. Valpreda, P. Morì, N. Fasfous, M. R. Vemparala, A. Frickenstein, L. Frickenstein, W. Stechele, C. Passerone, G. Masera, and M. Martina, "HW-Flow-Fusion: Inter-layer Scheduling for Convolutional Neural Network Accelerators with Dataflow Architectures," *Electronics*, vol. 11, no. 18, p. 2933, 2022.

[69] Z. Li, F. Z. Hong, and C. P. Yue, "FPGA-based Acceleration of Neural Network for Image Classification Using Vitis AI," *arXiv preprint arXiv:2412.20974*, 2024.

[70] J. Wang and S. Gu, "FPGA Implementation of Object Detection Accelerator Based on Vitis-AI," in *2021 11th International Conference on Information Science and Technology (ICIST)*. IEEE, 2021, pp. 571–577.

[71] T. Aarrestad, V. Loncar, N. Ghielmetti, M. Pierini, S. Summers, J. Ngadiuba, C. Petersson, H. Linander, Y. Iiyama, G. Di Guglielmo *et al.*, "Fast Convolutional Neural Networks on FPGAs with hls4ml," *Machine Learning: Science and Technology*, vol. 2, no. 4, p. 045015, 2021.

[72] V. Rybalkin, A. Pappalardo, M. M. Ghaffar, G. Gambardella, N. Wehn, and M. Blott, "FINN-L: Library Extensions and Design Trade-off Analysis for Variable Precision LSTM Networks on FPGAs," in *2018 28th international conference on field programmable logic and applications (FPL)*. IEEE, 2018, pp. 89–897.

[73] J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang, "FPGA HLS Today: Successes, Challenges, and Opportunities," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 4, pp. 1–42, 2022.

[74] G. T. Tufa, F. A. Andargie, and A. Bijalwan, "Acceleration of Deep Neural Network Training Using Field Programmable Gate Arrays," *Computational Intelligence and Neuroscience*, vol. 2022, no. 1, p. 8387364, 2022.

[75] S. Zhang, O. Wallscheid, and M. Porrmann, "Machine Learning for The Control and Monitoring of Electric Machine Drives: Advances and Trends," *IEEE Open Journal of Industry Applications*, vol. 4, pp. 188–214, 2023.

[76] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating CNN Inference on FPGAs: A Survey," *arXiv preprint arXiv:1806.01683*, 2018.

[77] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," *Advances in neural information processing systems*, vol. 25, 2012.

[78] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with Convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[79] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[80] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[81] J. Li, Y. Liang, Z. Yang, and X. Li, "An Efficient Convolutional Neural Network Accelerator Design on FPGA Using the Layer-to-Layer Unified Input Winograd Architecture," *Electronics*, vol. 14, no. 6, p. 1182, 2025.

[82] A. Haijoub, A. Hatim, M. Arioua, S. Hammia, A. Eloualkadi, and A. Guerrero-González, "Implementing Convolutional eural Networks on FPGA: A Survey and Research," in *ITM Web of Conferences*, vol. 52. EDP Sciences, 2023, p. 02004.

[83] A. Abdelfattah, A. Haidar, S. Tomov, and J. Dongarra, "Performance, Design, and Autotuning of Batched GEMM for GPUs," in *High Performance Computing: 31st International Conference, ISC High Performance 2016, Frankfurt, Germany, June 19-23, 2016, Proceedings*. Springer, 2016, pp. 21–38.

[84] T. Abtahi, C. Shea, A. Kulkarni, and T. Mohsenin, "Accelerating Convolutional Neural Network with FFT on Embedded Hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 9, pp. 1737–1749, 2018.

[85] J. Zhang, F. Franchetti, and T. M. Low, "High Performance Zero-memory Overhead Direct Convolutions," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5776–5785.

[86] Y. Zhou, M. Yang, C. Guo, J. Leng, Y. Liang, Q. Chen, M. Guo, and Y. Zhu, "Characterizing and Demystifying The Implicit Convolution Algorithm on Commercial Matrix-multiplication Accelerators," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2021, pp. 214–225.

[87] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2017, pp. 152–159.

[88] J. Yang, M. Wen, J. Shen, Y. Cao, M. Tang, R. Yang, J. Fei, and C. Zhang, "Bp-im2col: Implicit im2col supporting AI Backpropagation on Systolic Arrays," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 2022, pp. 415–418.

[89] H. Zeng, R. Chen, and V. K. Prasanna, "Optimizing frequency domain implementation of cnns on fpgas," *University of Southern California, Tech. Rep*, 2017.

[90] M. Zhang, L. Li, H. Wang, Y. Liu, H. Qin, and W. Zhao, "Optimized Compression for Implementing Convolutional Neural Networks on FPGA," *Electronics*, vol. 8, no. 3, p. 295, 2019.

[91] S. Li, Y. Luo, K. Sun, N. Yadav, and K. K. Choi, "A Novel FPGA Accelerator Design for Teal-time and Ultra-low Power Deep Convolutional Neural Networks Compared with titan X GPU," *IEEE Access*, vol. 8, pp. 105 455–105 471, 2020.

[92] M. P. Véstias, R. P. Duarte, J. T. De Sousa, and H. C. Neto, "A Configurable Architecture for Running Hybrid Convolutional Neural Networks in Low-density FPGAs," *IEEE Access*, vol. 8, pp. 107 229–107 243, 2020.

[93] J. Wang, Q. Lou, X. Zhang, C. Zhu, Y. Lin, and D. Chen, "Design Flow of Accelerating hybrid Extremely Low Bit-width Neural Network in Embedded FPGA," in *2018 28th international conference on field programmable logic and applications (FPL)*. IEEE, 2018, pp. 163–1636.

[94] S. Neelam and A. Amalin Prince, "VCONV: A Convolutional Neural Network Accelerator for FPGAs," *Electronics*, vol. 14, no. 4, p. 657, 2025.

[95] B. Hunter, S. Hindocha, and R. W. Lee, "The Role of Artificial Intelligence in Early Cancer Diagnosis," *Cancers*, vol. 14, no. 6, p. 1524, 2022.

[96] C. Weidenbaum and M. K. Gibson, "Approach to Localized Squamous Cell Cancer of The Esophagus," *Current Treatment Options in Oncology*, vol. 23, no. 10, pp. 1370–1387, 2022.

[97] M. Shi and G.-q. Zhai, "Models for Predicting Early Death in Patients with Stage IV Esophageal Cancer: a Surveillance, Epidemiology, and End Results–Based Cohort Study," *Cancer Control*, vol. 29, p. 10732748211072976, 2022.

[98] H.-E. Kim, H. H. Kim, B.-K. Han, K. H. Kim, K. Han, H. Nam, E. H. Lee, and E.-K. Kim, "Changes in Cancer Detection and False-Positive Recall in Mammography Using Artificial Intelligence: a Retrospective, Multireader Study," *The Lancet Digital Health*, vol. 2, no. 3, pp. e138–e148, 2020.

[99] R. C. Mayo, D. Kent, L. C. Sen, M. Kapoor, J. W. Leung, and A. T. Watanabe, "Reduction of False-positive Markings on Mammograms: a Retrospective Comparison Study Using an Artificial Intelligence-based CAD," *Journal of digital imaging*, vol. 32, pp. 618–624, 2019.

[100] H. Lin, R. Li, Z. Liu, J. Chen, Y. Yang, H. Chen, Z. Lin, W. Lai, E. Long, X. Wu *et al.*, "Diagnostic Efficacy and Therapeutic Decision-making Capacity of an Artificial Intelligence Platform for Childhood Cataracts in Eye Clinics: A Multicentre Randomized Controlled Trial," *EClinicalMedicine*, vol. 9, pp. 52–59, 2019.

[101] J. Jiang, Q. Xie, Z. Cheng, J. Cai, T. Xia, H. Yang, B. Yang, H. Peng, X. Bai, M. Yan *et al.*, "AI Based Colorectal Disease Detection using Real-time Screening Colonoscopy," *Precision Clinical Medicine*, vol. 4, no. 2, pp. 109–118, 2021.

[102] Y. Xu, T. M. Khan, Y. Song, and E. Meijering, "Edge Deep Learning in Computer Vision and Medical Diagnostics: a Comprehensive Survey," *Artificial Intelligence Review*, vol. 58, no. 3, pp. 1–78, 2025.

[103] X. Gao and B. Braden, "Artificial Intelligence in Endoscopy: The Challenges and Future Directions," *Artificial Intelligence in Gastrointestinal Endoscopy*, vol. 2, no. 4, pp. 117–126, 2021.

[104] Z. He, K. Zhang, N. Zhao, Y. Wang, W. Hou, Q. Meng, C. Li, J. Chen, and J. Li, "Deep Learning for Real-time Detection of Nasopharyngeal Carcinoma During Nasopharyngeal Endoscopy," *Iscience*, vol. 26, no. 10, 2023.

[105] W. Cao, W. Shen, Z. Zhang, and J. Qin, "Privacy-preserving Healthcare Monitoring for IoT Devices Under Edge Computing," *Computers & Security*, vol. 134, p. 103464, 2023.

[106] A. Singh and K. Chatterjee, "Securing Smart Healthcare System with Edge Computing," *Computers & Security*, vol. 108, p. 102353, 2021.

[107] A. M. Alwakeel, "An Overview of Fog Computing and Edge Computing Security and Privacy Issues," *Sensors*, vol. 21, no. 24, p. 8226, 2021.

[108] E. Morgan, I. Soerjomataram, H. Rumgay, H. G. Coleman, A. P. Thrift, J. Vignat, M. Laversanne, J. Ferlay, and M. Arnold, "The global Landscape of Esophageal Squamous Cell Carcinoma and Esophageal Adenocarcinoma Incidence and Mortality in 2020 and Projections to 2040: New Estimates from GLOBOCAN 2020," *Gastroenterology*, vol. 163, no. 3, pp. 649–658, 2022.

[109] N. C. Institute, " Upper Endoscopy," 2025, https://www.cancer.gov/publications/dictionaries/cancer-terms/def/upper-endoscopy.

[110] İ. Arıkan, T. Ayav, A. Ç. Seçkin, and F. Soygazi, "Estrus Detection and Dairy Cow Identification with Cascade Deep Learning for augmented Reality-ready Livestock Farming," *Sensors*, vol. 23, no. 24, p. 9795, 2023.

[111] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[112] L. Chi, B. Jiang, and Y. Mu, "Fast Fourier Convolution," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4479–4488, 2020.

[113] A. Anderson, A. Vasudevan, C. Keane, and D. Gregg, "High-performance Low-memory Lowering: GEMM-based Algorithms for DNN Convolution," in *2020 ieee 32nd international symposium on computer architecture and high performance computing (sbac-pad)*. IEEE, 2020, pp. 99–106.

[114] Y. Huang, J. Shen, Z. Wang, M. Wen, and C. Zhang, "A High-efficiency FPGA-based Accelerator for Convolutional Neural Networks Using Winograd Algorithm," in *Journal of Physics: Conference Series*, vol. 1026, no. 1. IOP Publishing, 2018, p. 012019.

[115] M. M. Ghaffar, C. Sudarshan, C. Weis, M. Jung, and N. Wehn, "A Low Power in-DRAM Architecture for Quantized CNNs Using Fast Winograd Convolutions," in *Proceedings of the International Symposium on Memory Systems*, 2020, pp. 158–168.

[116] A. Karbachevsky, C. Baskin, E. Zheltonozhskii, Y. Yermolin, F. Gabbay, A. M. Bronstein, and A. Mendelson, "Early-stage Neural Network Hardware Performance Analysis," *Sustainability*, vol. 13, no. 2, p. 717, 2021.

[117] Y. Lin, Y. Zhang, and X. Yang, "A Low Memory Requirement Mobilenets Accelerator Based on FPGA for Auxiliary Medical Tasks," *Bioengineering*, vol. 10, no. 1, p. 28, 2022.

[118] Kaggle, " Real Oesophageal Cancer Datasets," 2025, https://www.kaggle.com/chopinforest/esophageal-endoscopy-images.

[119] J. Wang, W. Tong, and X. Zhi, "Model Parallelism Optimization for CNN FPGA Accelerator," *Algorithms*, vol. 16, no. 2, p. 110, 2023.

[120] N. Zayed, N. Tawfik, M. M. Mahmoud, A. Fawzy, Y.-I. Cho, and M. S. Abdallah, "Accelerating Deep Learning-Based Morphological Biometric Recognition with Field-Programmable Gate Arrays," *AI*, vol. 6, no. 1, p. 8, 2025.