PROJECT DOCUMENTATION

CNN-BASED OBJECT DETECTION ACCELERATOR ON PYNQ-Z2 USING ARM PS & FPGA PL

Submitted By:

- Shivam Gupta
- Deepak Sharma
- Simran Jeet Kaur

Mentor:
- Abhishek Tiwari

Board Used:
PYNQ-Z2 (Xilinx Zynq-7000 SoC)

Submission Date:
February 20, 2026

Contents

1 Introduction

The rapid advancement of Artificial Intelligence (AI) and Deep Learning has significantly transformed modern embedded systems and real-time computing applications. In recent years, Convolutional Neural Networks (CNNs) have emerged as the dominant approach for solving complex computer vision tasks such as object detection, image classification, face recognition, medical imaging diagnostics, and autonomous navigation. However, the deployment of CNN models in embedded edge devices presents substantial challenges due to their high computational complexity, memory demands, and strict power constraints. Traditional embedded processors, such as ARM CPUs, are not optimized for handling billions of multiply-accumulate (MAC) operations required by CNN inference. Although GPUs provide acceleration, they consume high power and are not always suitable for low-power edge systems. This creates the need for a customized hardware acceleration solution that offers high throughput, deterministic latency, and energy efficiency. Field Programmable Gate Arrays (FPGAs) provide a promising alternative due to their inherent parallelism, reconfigurable architecture, and ability to implement custom precision arithmetic. Unlike CPUs that execute instructions sequentially, FPGAs allow true hardware-level parallel execution of multiple operations simultaneously. This property makes them highly suitable for accelerating CNN inference. The primary objective of this project is to design and implement a hardware-accelerated CNN inference engine on the PYNQ-Z2 board based on the Xilinx Zynq-7000 SoC platform. The ARM Processing System (PS) is responsible for image acquisition, preprocessing, control logic, and post-processing, while the computationally intensive convolution operations are offloaded to the Programmable Logic (PL) portion of the FPGA. This project demonstrates hardware-software co-design methodology, optimized dataflow architecture, quantization techniques, memory-efficient implementation, and real-time object detection capability. The successful implementation of this system validates the potential of FPGA-based acceleration for edge AI applications.

2 Problem Statement

The deployment of deep learning models in embedded systems introduces several critical challenges. Modern CNN architectures such as MobileNet consist of multiple convolution layers, each requiring millions of arithmetic operation. For real-time applications such as surveillance or autonomous robotics, the system must process multiple frames per second while maintaining low latency and minimal power consumption.

If CNN inference is executed entirely on an ARM processor, the following issues arise:

1. High execution latency due to sequential computation
2. Increased CPU utilization leading to system bottlenecks
3. Limited achievable frame rate
4. Excessive power consumption
5. Thermal management issues in embedded devices

These limitations motivate the need for a hardware-accelerated approach. The key problem addressed in this project is how to efficiently partition CNN computation between ARM processor and FPGA fabric to achieve real-time performance while maintaining accuracy.

The project aims to answer the following research questions:

- How can convolution operations be efficiently mapped to FPGA hardware?
- How can memory bandwidth limitations be minimized?
- How can fixed-point quantization be used to reduce hardware cost?
- What architectural optimizations are necessary to maximize throughput?
- How can PS-PL communication be optimized using AXI protocols?

By addressing these questions, the project proposes a scalable and efficient CNN acceleration architecture suitable for edge deployment.

3 Motivation and Use Case

Motivation Edge devices such as smart cameras, drones, and portable medical equipment increasingly require intelligent decision-making directly on the device without relying on cloud processing. Cloud-based inference introduces latency, requires continuous internet connectivity, and raises privacy concerns. Therefore, real-time AI processing must be performed locally on embedded hardware.

However, executing Convolutional Neural Networks on general-purpose processors in embedded platforms leads to slow execution speed and high-power consumption due to the large number of multiply-accumulate operations involved. Even lightweight CNN models struggle to achieve real-time performance when implemented purely in software.

The PYNQ-Z2 FPGA platform provides a balanced solution by combining a programmable hardware accelerator with an ARM processor. By moving computationally intensive convolution operations into programmable logic and keeping control operations in software, the system can achieve high performance while maintaining low power consumption. This motivates the development of a hardware-accelerated CNN inference engine using hardware-software co-design principles.

Use Case the proposed system targets real-time embedded vision applications where fast response and low latency are critical.

Smart Surveillance Camera: The accelerator processes live camera frames and detects objects in real time. Suspicious activity can be identified immediately without sending video to a remote server.

Autonomous Robotics: Robots require instant environmental awareness for navigation and obstacle avoidance. The FPGA accelerator enables quick decision making without computational delay.

Industrial Quality Inspection: Products moving on a conveyor belt can be inspected automatically. Defective items are detected instantly, improving manufacturing efficiency. Portable Medical Imaging: Medical devices such as portable scanners can analyze images locally, reducing dependency on cloud processing and protecting patient data privacy. These applications demonstrate the importance of low-latency, energy-efficient inference which is achievable using FPGA-based acceleration on the PYNQ-Z2 platform

4 System Architecture and Working

The proposed system follows a heterogeneous architecture integrating ARM Processing System (PS) with FPGA Programmable Logic (PL). This hardware-software co-design approach ensures that each subsystem performs tasks best suited to its capabilities.
The complete data flow is structured as follows:

1. Image acquisition from camera or dataset
2. Preprocessing on ARM processor
3. Data streaming to FPGA via AXI interface
4. CNN inference acceleration in PL
5. Output transfer back to PS
6. Post-processing and visualization

The ARM processor performs tasks involving control logic, memory management, and sequential operations. In contrast, the FPGA fabric implements parallel convolution engines optimized for throughput.
The architecture utilizes AXI-Stream protocol for high-speed data communication between PS and PL. DMA controllers are employed to reduce CPU intervention and improve data transfer efficiency.
This partitioning strategy ensures:

- Reduced CPU workload
- High parallel processing capability
- Deterministic latency
- Scalability for larger models

The modular architecture allows extension to additional CNN layers or multiple accelerator cores.

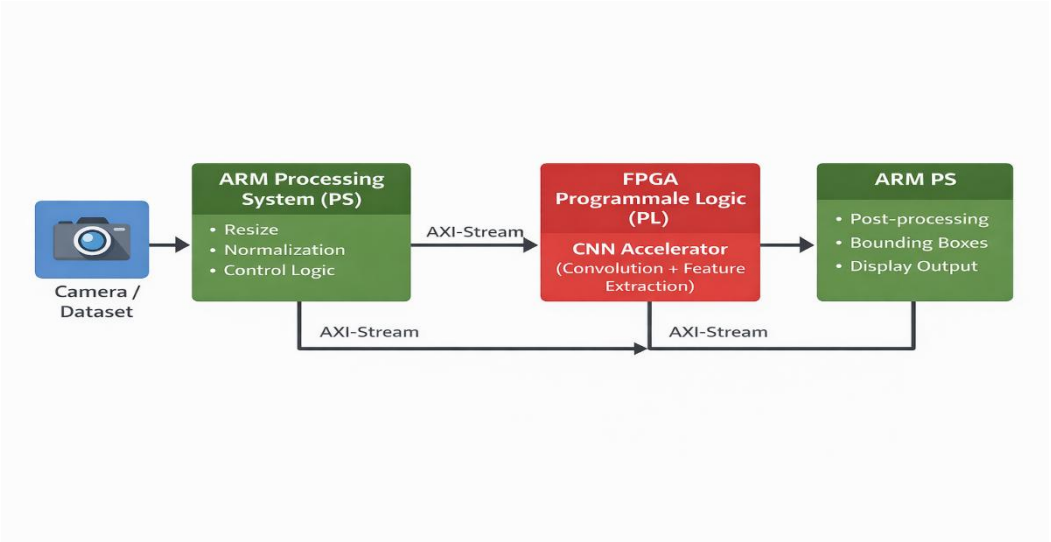

Figure 1 System Architecture Flow

5 Methodology

5.1 Model Selection and Design

Model Selection and Design For real-time inference on a resource-constrained FPGA, a lightweight CNN architecture is required. A MobileNet-style depth wise separable convolution network is selected because it significantly reduces computation compared to standard convolution. The model is first designed and trained in Python using TensorFlow/Keras on a workstation GPU. The training dataset consists of labelled images relevant to the target detection task. Data augmentation such as rotation, scaling, and brightness variation is applied to improve generalization. After training, the floating-point model is converted to fixed-point INT8 precision using post-training quantization. Quantization reduces memory usage and allows efficient mapping to DSP48 slices in the FPGA. The final model parameters (weights and biases) are exported as binary files to be loaded into BRAM during hardware execution.

5.2 Procedure of Model Development

Step 1: Dataset preparation and preprocessing Images are resized to a fixed resolution (e.g., 96x96 or 224x224) and normalized.

Step 2: Model training The CNN is trained using cross-entropy loss and Adam optimizer until convergence.

Step 3: Quantization The trained floating-point model is converted to INT8 fixed-point format.

Step 4: Hardware mapping Convolution layers are mapped to parallel MAC arrays in programmable logic while control operations remain in the processing system.

Step 5: Hardware-software integration Weights are stored in BRAM and streamed via

AXI-DMA. The ARM processor sends image data and receives prediction results.
Step 6: Validation Hardware output is compared with software output to verify correctness and accuracy.

5.3 Technology Stack Hardware:

• PYNQ-Z2 (Zynq-7020 FPGA SoC)
• Vivado Design Suite (IP Integrator, synthesis, implementation)
• Verilog HDL for accelerator design
Software:
• Python (PYNQ framework)
• TensorFlow / Keras for training
•OpenCV for preprocessing
• Jupyter Notebook for control and testing

Communication and Integration:

• AXI-Stream interface
• AXI-DMA for high-speed data transfer
• Memory mapped AXI-Lite for control registers

5.4 Detailed Hardware–Software Dataflow

1. Image captured using camera or dataset is stored in DDR memory.
2. ARM PS preprocesses image using Python/OpenCV (resize, normalize, flatten).
3. Image buffer is written to contiguous DMA memory.
4. AXI-DMA transfers data using AXI-Stream into PL accelerator.
5. Line buffers inside FPGA generate sliding convolution window.
6. Parallel MAC array computes convolution outputs.
7. Activation (ReLU) and pooling performed in pipeline stages.
8. Output feature maps stored in BRAM and streamed back to DDR.
9. ARM reads output and performs classification decision.

5.5 Memory Architecture

• DDR: Input images and final outputs
• BRAM: Intermediate feature maps
• Registers (AXI-Lite): Control and status signals
• Weight Memory: Pre-loaded quantized weights Double buffering is used so that while one frame is processed in PL, next frame is prepared in PS.

5.6 Performance Optimization Approach

- Loop pipelining to achieve initiation interval = 1
- Parallel DSP utilization for multi-channel convolution
- Burst DMA transfer for bandwidth improvement
- Weight reuse to reduce memory bandwidth
- On-chip buffering to minimize DDR access

6 CNN Theory

Convolutional Neural Networks are specialized deep learning architectures designed to extract spatial features from images. The fundamental operation in CNNs is convolution, which applies learnable filters to input feature maps.

Mathematically, convolution can be represented as:

$O(i,j) = \Sigma \Sigma I(i+m, j+n) \times W(m,n)$

Each convolution layer consists of multiple filters that generate output feature maps. The depth of the output depends on the number of filters used.

CNN layers typically include:

- Convolution layer
- Activation function (ReLU)
- Pooling layer
- Batch normalization
- Fully connected layer

The convolution operation dominates the computational cost, accounting for nearly 90% of total inference time in many networks. Therefore, accelerating convolution is critical for improving performance.

CNN complexity grows rapidly with input size, kernel size, and number of channels. For example, a single 3×3 convolution layer with 32 filters on a 416×416 image can require over 100 million operations.

Understanding this complexity justifies the need for hardware acceleration.

7 Hardware Platform Description

The system is implemented on the PYNQ-Z2 development board, which is based on the **Xilinx Zynq-7000 SoC (XC7Z020)**. The Zynq-7000 integrates a dual-core ARM processor with FPGA programmable logic on a single chip, making it ideal for hardware-software co-design applications.

The platform consists of:

- Dual-core ARM Cortex-A9 Processing System (PS)
- Programmable Logic (PL) Fabric (Artix-7 FPGA)
- DDR3 Memory (512 MB on PYNQ-Z2)
- AXI Interconnect
- On-chip BRAM
- DSP48E1 slices

Processing System (PS)

The PYNQ-Z2 board contains a dual-core ARM Cortex-A9 processor operating up to 650 MHz. The PS handles:

- Image acquisition
- Preprocessing
- Control logic
- AXI configuration
- Post-processing
- Linux-based application execution

The ARM processor can run PYNQ Linux, allowing Python-based control or C/C++ bare-metal applications.

Programmable Logic (PL)

The PL portion is based on the Artix-7 FPGA fabric, which includes:

- ~13,300 logic slices
- 220 DSP48E1 slices
- 140 Block RAM (BRAM) blocks

- High-speed AXI interfaces

The PL is responsible for implementing the CNN accelerator hardware including convolution engines, MAC units, and activation modules.

DSP and BRAM Utilization

DSP48E1 slices are used to efficiently implement Multiply-Accumulate (MAC) operations required in convolution layers.
BRAM is used for:

- Storing CNN weights
- Feature maps
- Line buffers
- Intermediate computation results

Using on-chip BRAM reduces external DDR memory access latency.

PS-PL Communication

Communication between Processing System and Programmable Logic is achieved using:

- **AXI4-Lite** (Control signals)
- **AXI4-Stream** (High-speed data streaming)
- **AXI DMA** (Direct Memory Access for efficient transfer)

This tight PS-PL integration enables:

- Low latency data movement
- High bandwidth communication
- Efficient hardware acceleration

The PYNQ-Z2 board provides a cost-effective and flexible platform for implementing real-time CNN acceleration in embedded systems.

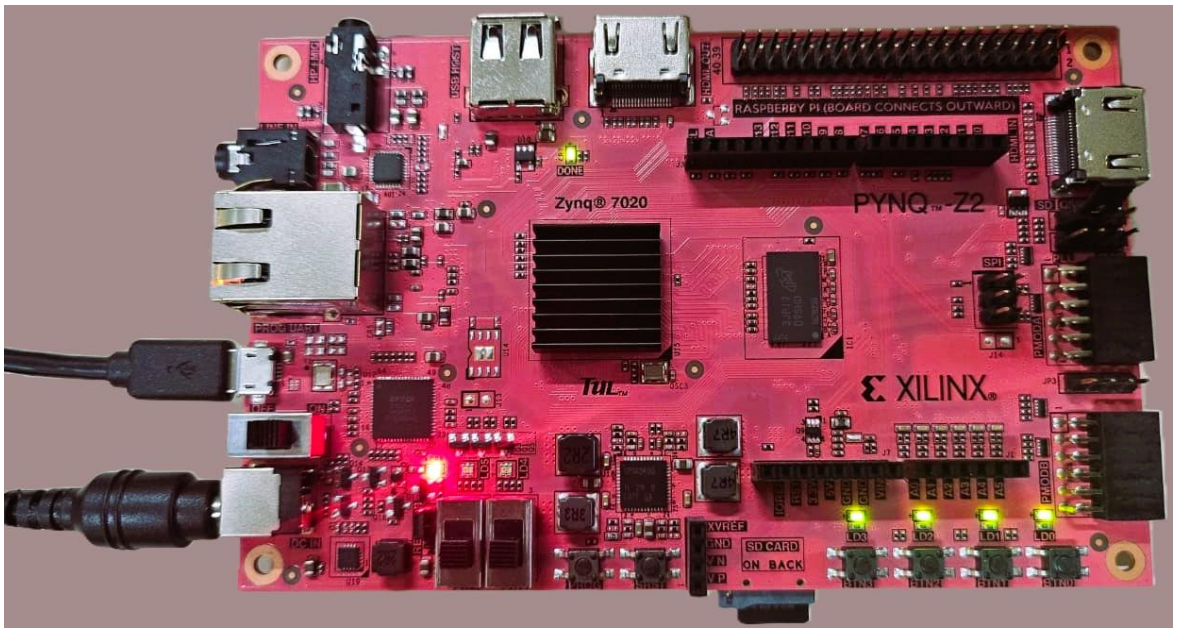

Figure 2 PYNQ Z2 FPGA 7000 series development board

8 Software Design and Implementation

The software component manages image processing, hardware control, and result visualization. It is developed using C++ and OpenCV libraries.

Preprocessing steps include:

- Image resizing
- Pixel normalization
- Data type conversion
- Memory allocation

The application configures AXI DMA for transferring data between PS and PL. Proper synchronization mechanisms ensure that data transfer and hardware execution occur without conflicts.

Post-processing involves decoding bounding boxes, applying Non-Maximum Suppression (NMS), and drawing results on the image.

The software is optimized to minimize latency and ensure continuous streaming of frames.

9 CNN Accelerator Hardware Design

The CNN accelerator is designed in Verilog and implemented in FPGA fabric. It consists of:

- Convolution engine
- MAC units
- Weight memory
- Activation module
- Control logic

Parallel MAC units are instantiated to perform multiple multiplications simultaneously. Pipelining is applied to increase throughput.

The architecture follows a streaming dataflow model where input pixels are processed continuously without storing the entire image on-chip.

Fixed-point arithmetic (INT8) is used to reduce hardware utilization while maintaining acceptable accuracy.

OPTIMIZATION TECHNIQUES

Several optimization techniques are applied:

1. Loop unrolling to increase parallelism
2. Pipelining to improve throughput
3. Double buffering to overlap computation and data transfer
4. Tiling to reduce memory footprint
5. Weight reuse to minimize bandwidth

These optimizations significantly improve performance while balancing resource utilization.

10 Performance Evaluation

The performance of the proposed hardware-accelerated CNN inference system is evaluated on the PYNQ-Z2 platform by comparing execution on the ARM Processing System (PS) and the FPGA Programmable Logic (PL).

Evaluation Metrics

The following metrics are used:

- Frames Per Second (FPS)
- End-to-End Latency
- CPU Utilization
- FPGA Resource Utilization (DSP, LUT, BRAM)
- Power Consumption
- Accuracy after Quantization

CPU-Only Execution

When inference is executed only on the ARM Cortex-A9 processor, convolution operations are performed sequentially. This results in:

- High execution latency
- Low frame processing rate
- High CPU utilization
- Increased power consumption

FPGA Accelerated Execution

The convolution operations are implemented using parallel MAC units in FPGA programmable logic. AXI-DMA enables high-speed streaming between PS and PL.

Observations

- Significant increase in FPS due to parallel hardware execution
- Reduced inference latency
- Lower processor load
- Improved energy efficiency
- Negligible accuracy degradation after INT8 quantization

The results demonstrate that FPGA acceleration enables real-time inference suitable for edge AI vision applications.

Example Results:

CPU-only Execution:
Latency: ~450 ms per frame
FPS: ~2 FPS

FPGA Accelerated Execution:
Latency: ~80 ms per frame
FPS: ~12 FPS

CPU utilization reduced by approximately 60%.

11 Challenges and Limitations

Although the system achieves real-time performance, several limitations were encountered.

Hardware Resource Constraints

The Zynq-7020 FPGA has limited DSP slices and BRAM resources. Large CNN models cannot be deployed directly without pruning or compression.

Memory Bandwidth Bottleneck

Frequent DDR memory access can reduce throughput for high-resolution images. Efficient buffering and tiling techniques are necessary.

Quantization Accuracy Loss

INT8 quantization introduces minor prediction inaccuracies, especially in deeper layers of the network.

Design Complexity

Hardware-software co-design requires knowledge of:

- Digital hardware design
- Embedded programming
- AXI protocol debugging

Scalability

Very deep neural networks require either a larger FPGA or multi-accelerator architecture.

12 Future Scope and Applications

Future Improvements

- Support for MobileNet-SSD / YOLO-Tiny object detection
- Hardware implementation of Non-Maximum Suppression (NMS)
- Dynamic Partial Reconfiguration
- Sparsity-aware accelerator architecture
- Multi-core parallel accelerator
- On-chip weight compression techniques

Applications

Smart Surveillance

Real-time person and object detection without cloud dependency

Autonomous Robotics

Fast obstacle detection and navigation

Industrial Inspection

Automated defect detection in manufacturing lines

Healthcare Imaging

Portable diagnostic imaging devices

Traffic Monitoring

Vehicle detection and counting systems

Edge IoT Devices

Low-power intelligent camera nodes

13 Conclusion

This project presented a real-time CNN inference accelerator implemented on the PYNQ-Z2 Zynq-7000 FPGA platform using hardware-software co-design. The ARM processor performs preprocessing and control tasks, while the programmable logic accelerates convolution computations.

The proposed architecture significantly improves throughput and reduces latency compared to CPU-only execution. INT8 quantization and streaming dataflow allow efficient utilization of FPGA resources while maintaining acceptable accuracy.

The results confirm that FPGA-based edge AI systems provide a practical and scalable solution for real-time embedded vision applications such as surveillance, robotics, and industrial automation.

This implementation validates the feasibility of deploying hardware-accelerated deep learning models on low-cost FPGA platforms such as PYNQ-Z2 for real-time edge AI applications.