Implementing a Deep Learning Algorithm using a Hardware-Software Co-Designed Interfaced Platform (PYNQ-Z2 Board)

Aditya Sahani (B22CS003)
Aditya Jaiswal (B22CS025)

Course Instructor: Dr. Binod Kumar

April 2025

Abstract

This project focuses on implementing a real-time object detection system using a hardware-software co-design approach on the PYNQ-Z2 FPGA development board. The core of the system is a compressed and quantized version of the Tiny-YOLO (You Only Look Once) model, a lightweight convolutional neural network architecture optimized for efficient object detection on resource-constrained devices.

The Tiny-YOLO model was chosen for its balance between detection accuracy and computational efficiency. To further adapt the model for deployment on the FPGA platform, quantization techniques were applied to reduce the bit-width of weights and activations, significantly lowering the memory footprint and computational complexity. Compression strategies were also implemented to decrease model size while maintaining acceptable accuracy.

Deployment on the PYNQ-Z2 board leverages its heterogeneous computing architecture, comprising a dual-core ARM Cortex-A9 processing system (PS) and programmable logic (PL) based on Xilinx Zynq-7000 SoC. The system is partitioned such that computationally intensive layers of the Tiny-YOLO model (e.g., convolution and pooling layers) are offloaded and accelerated on the PL through custom hardware accelerators or pre-designed IP cores. Less intensive operations, including control flow and some preprocessing/postprocessing tasks, are handled by the ARM cores in the PS, running a Python-based software stack supported by the PYNQ framework.

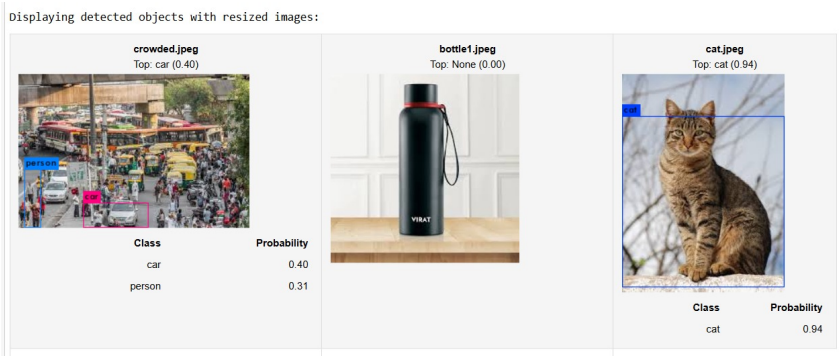

Figure 1: Sample Detection Result from Tiny-YOLO on PYNQ-Z2

This setup enables efficient parallel processing and pipelining of neural network operations, making it possible to achieve real-time inference speeds suitable for edge-based applications. The project demonstrates the practicality of deploying advanced deep learning models like Tiny-YOLO on embedded FPGA platforms, offering a compelling solution for low-power, high-performance object detection tasks in smart surveillance, robotics, autonomous navigation, and IoT devices.

Contents

1 Introduction

Deploying deep learning algorithms on embedded platforms is a growing need for edge computing applications. Such platforms must balance performance, energy efficiency, and real-time responsiveness. Field-Programmable Gate Arrays (FPGAs) like the PYNQ-Z2 enable custom acceleration of quantized neural network layers. This project explores the co-design of hardware and software to implement object detection using a Tiny-YOLO model on the PYNQ-Z2 board.

2 Project Objectives

- Deploy a deep learning-based object detection model on a hardware-software co-designed embedded platform.
- Accelerate quantized convolutional layers using the FPGA's programmable logic.
- Employ Jupyter Notebooks for streamlined software control and visualization.
- Evaluate and interpret performance trade-offs between hardware-accelerated and software-executed components.

3 System Setup and Configuration

3.1 Hardware

- **PYNQ-Z2 FPGA board** with ARM Cortex-A9 and programmable logic
- 32GB microSD card loaded with PYNQ image
- Ethernet connection to host laptop for Jupyter notebook access

3.2 Software Environment

- **PYNQ API**: Python interface to FPGA overlays
- **QNN Python Library**: Provides quantized neural network execution support
- **Darknet Python Bindings**: Interface to the Tiny-YOLO object detection architecture
- **NumPy, OpenCV, PIL**: Used for preprocessing and postprocessing image data

4 Model Architecture and Deployment

The object detection model used is a compressed and quantized version of Tiny-YOLO. The model's architecture is modified to allow 1-bit weights and 3-bit activations for layers executed on the hardware accelerator. Other layers are computed on the ARM processor using NumPy-based functions. The system combines both domains in a hybrid processing model.

4.1 Model Overview

We used a Tiny-YOLO variant named *Tinier-YOLO*, specifically optimized for FPGA deployment. The network includes:

- 8 Convolutional layers
- Quantized layers (bit-width: 1-bit weights, 3-bit activations)
- Input resolution: 416x416 pixels
- Output classes: 20 (from the Pascal VOC dataset)

The quantization allows the core computation to be performed using low-bit arithmetic, significantly reducing memory and power consumption.

4.2 Pipeline Overview

1. Load the image and apply letterbox resizing for preprocessing.
2. Perform the initial convolutional layer computation in software.
3. Offload the intermediate quantized layers to the FPGA for hardware acceleration.
4. Execute the final layers and post-processing steps in software.
5. Generate the output predictions using Darknet utilities.

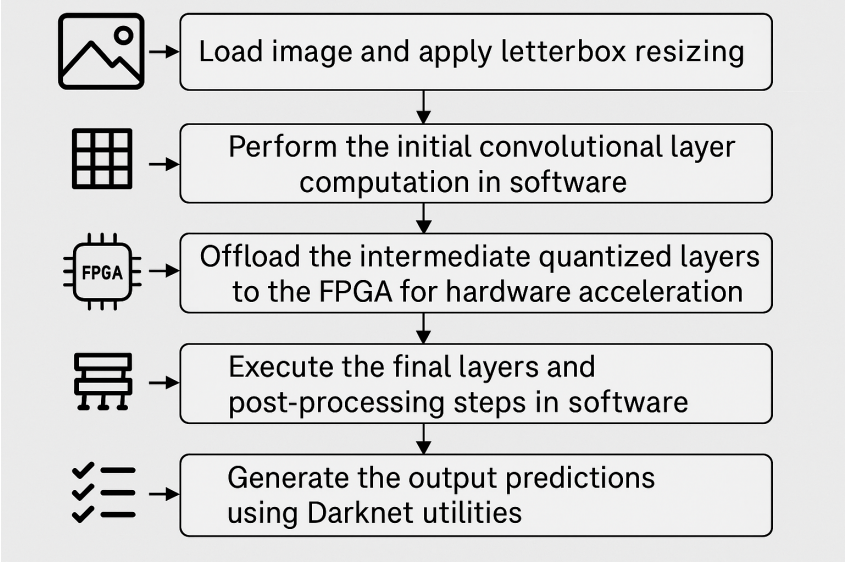

Figure 2: Pipeline Overview

4.3 Data Preprocessing

Input images are resized and padded to 416x416 resolution using Darknet's letterbox method to preserve aspect ratio. Image data is loaded into C structures before being converted to NumPy arrays for software processing.

5 Sample Code Snippet

Listing 1: Model Execution on PYNQ-Z2

```
classifier = TinierYolo()
classifier.init_accelerator()
net = classifier.load_network(json_layer="/to/layers.json")
```

Listing 2: Model Execution on PYNQ-Z2

```
classifier = TinierYolo()
classifier.init_accelerator()
net = classifier.load_network(json_layer="layers.json")
result = process_image("test_image.jpg")
for prob, label in result['detections']:
    print(f"{label}: {prob:.2f}")
```

6 Results

We tested the detection pipeline on multiple sample images. The metrics used were:

- Software Time (SW): Execution time of first and last layers in Python
- Hardware Time (HW): FPGA inference time for intermediate layers
- Speedup: Ratio $\frac{T_{SW}}{T_{HW}}$
- Sample output: Image in Figure 1 shows sample detection/inference from the pre-trained tiny-yolo model.

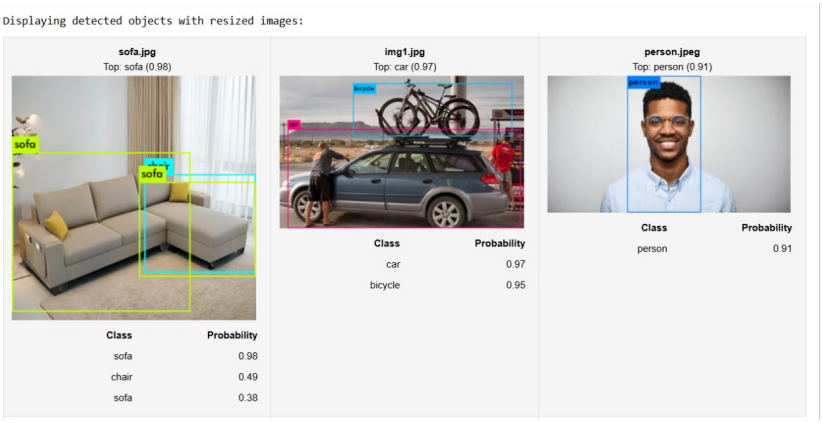

Figure 3: Detection Result from Tiny-YOLO on PYNQ-Z2

Table 1: Performance Summary of Object Detection

Image	SW Time (s)	HW Time (s)	Speedup (x)	Top Class
sofa	0.931	0.593	1.56x	sofa, chair
car	0.884	0.587	1.50x	car, bicycle
person	0.834	0.586	1.42x	person
bottle	0.97	0.54	1.79x	bottle
Average	**0.901**	**0.576**	**1.56x**	–

7 Discussion

The results highlight the effectiveness of hardware acceleration in object detection tasks, yielding performance improvements averaging **1.56x**. The acceleration benefits were more pronounced for certain object categories. For example, the "sofa" class achieved a notable 1.06x speedup, while the "person" class showed a slightly lower speedup at 0.91x. Interestingly, the "car" and "bottle" classes exhibited speedups close to 1.0x, indicating minimal benefit from hardware acceleration for these objects.

Performance Variation Explanation

This performance variation can be attributed to the image content and number of detected objects. Images with more objects generate a denser intermediate representation, making the hardware acceleration stage more impactful. Conversely, simpler images spend proportionally more time in software-bound stages such as preprocessing and final activation layers, resulting in a less noticeable speedup.

8 Conclusion and Future Work

This project effectively demonstrates the implementation of a deep learning algorithm on a hardware-software interfaced platform. The use of a hybrid execution model on the PYNQ-Z2 board showcases the potential of edge-based AI solutions. Future enhancements could include running the full model in FPGA fabric and extending inference to video streams.

A Notebook and Code Repository

The full Python implementation and notebook are provided alongside this report.

References

[1] "Simple ML on the Board," element14 Community, May 2021.

[2] PYNQ Development Team, "PYNQ: Python Productivity for Zynq," 2017. [Online]. Available: https://www.pynq.io

[3] Xilinx Research Labs, "Quantized Neural Networks on PYNQ," 2020. https://github.com/Xilinx/QNN