Implementing a Deep Learning Algorithm Using a Hardware-Software Interfaced Platform (PYNQ-Z2 Board)

EEL3090 : Embedded Systems

| V Sujay (B22CS063) | Aiswarya K (B22CS028)|

April 2025

Code Files

Contents

1 Abstract

This project demonstrates the deployment of a **quantized DoReFa-Net** convolutional neural network on the **Xilinx PYNQ-Z2** development board, integrating **Python-based control** with **FPGA-accelerated inference**. We achieve **real-time ImageNet classification** by offloading the bulk of convolutional and fully connected computations to **programmable logic**, while maintaining the flexibility of high-level software orchestration on the **ARM Cortex-A9** processing system. **Quantization** to **1-bit weights** and **2-bit activations** reduces **on-chip memory usage** and simplifies arithmetic operations, yielding an overall inference speedup of approximately **1.5**× over a **CPU-only baseline** with **negligible accuracy degradation**.

2 Objectives

- **Quantization-Aware Inference:** Implement **DoReFa-Net** with **1-bit weights** and **2-bit activations** to minimize memory footprint and computational complexity.
- **Hardware Acceleration:** Leverage the **PYNQ-Z2's programmable logic** to perform **convolutional** and **dense layer operations** in hardware for improved efficiency.
- **Software-Hardware Co-Design:** Integrate the **PYNQ Python framework** (including the **qnn package**) with custom **FPGA IP cores** to enable seamless end-to-end inference.
- **Performance Evaluation:** Conduct comprehensive measurements of **latency**, **throughput**, and **classification accuracy** comparing **CPU-only** and **FPGA-accelerated** implementations.

3 Background

3.1 Edge AI and Quantization

Edge devices often lack the compute and energy resources to run full-precision deep neural networks. **Quantization** reduces model size and computational complexity by representing network parameters and activations with **low-bit-width integers**, enabling deployment on **resource-constrained hardware** without substantial accuracy loss.

3.2 PYNQ-Z2 Platform

The **PYNQ-Z2 board** features a **Xilinx Zynq-7000 SoC**, which tightly integrates:

- **Processing System (PS):** Dual-core **ARM Cortex-A9** running Linux and Python
- **Programmable Logic (PL): Artix-7 FPGA** fabric hosting custom accelerators

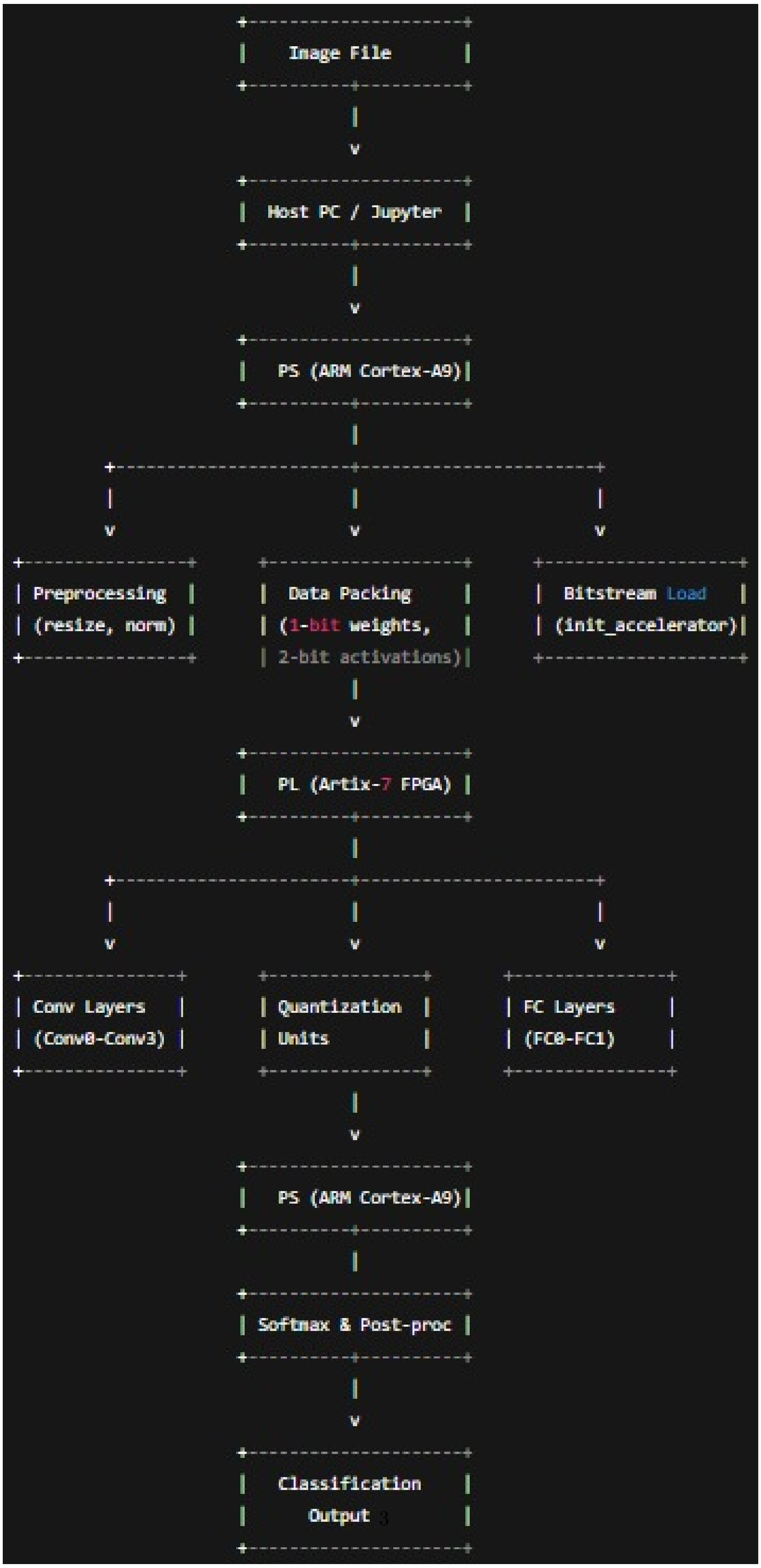
PYNQ provides a high-level **Python API** for:

- Loading FPGA bitstreams (`"overlays"`)
- Moving data between PS and PL domains

4 System Architecture

```
[Host PC] ←Ethernet→ [ARM PS (Python)] ←→ [PL (FPGA Overlay)]
```

Host PC:	Web browser–based JupyterLab interface.
PS:	Runs the `qnn.DorefaNet` Python class, handles image I/O and preprocessing.
PL Overlay:	Implements pipelined convolutional and dense layers, quantization units, and memory buffers.



```
                       +---------------------+
                       |     Image File      |
                       +----------+----------+
                                  |
                                  v
                       +---------------------+
                       |  Host PC / Jupyter  |
                       +----------+----------+
                                  |
                                  v
                       +---------------------+
                       |   PS (ARM Cortex-A9)|
                       +----------+----------+
                                  |
            +---------------------+---------------------+
            |                     |                     |
            v                     v                     v
+-----------------+     +---------------------+     +------------------+
| Preprocessing   |     |  Data Packing       |     | Bitstream Load   |
| (resize, norm)  |     | (1-bit weights,     |     | (init_accelerator)|
+-----------------+     | 2-bit activations)|     +------------------+
                                  |
                                  v
                       +---------------------+
                       |   PL (Artix-7 FPGA) |
                       +----------+----------+
                                  |
            +---------------------+---------------------+
            |                     |                     |
            v                     v                     v
+---------------+       +---------------+       +---------------+
| Conv Layers   |       | Quantization  |       | FC Layers     |
| (Conv0-Conv3) |       | Units         |       | (FC0-FC1)     |
+---------------+       +---------------+       +---------------+
                                  |
                                  v
                       +---------------------+
                       |   PS (ARM Cortex-A9)|
                       +----------+----------+
                                  |
                       +---------------------+
                       | Softmax & Post-proc |
                       +----------+----------+
                                  |
                                  v
                       +---------------------+
                       |   Classification    |
                       |      Output         |
                       +---------------------+
```

5 DoReFa-Net Model Overview

Convolutional Layers (Conv0–Conv3)

- **Weight precision:** 1 bit
- **Activation precision:** 2 bits
- **Strides and kernel sizes:** As specified in the reference architecture

Fully Connected Layers (FC0–FC1)

- Same quantization scheme as convolutional layers
- Final softmax operation performed on Processing System (PS)

Quantization Scheme

- Non-uniform quantizer mapping floating-point values to integer-coded representations
- Utilizes dedicated "T" scaling factors loaded alongside weights
- Optimized for FPGA implementation with minimal precision loss

6 Implementation Details

6.1 Environment Setup

- Python 3.6 on PYNQ Linux image
- `qnn` package providing:
  - `DorefaNet` class
  - FPGA bitstream file
  - Parameter JSON configuration

```
from qnn import DorefaNet
classifier = DorefaNet()
classifier.init_accelerator()  # Programs the FPGA with DoReFa overlay
net = classifier.load_network(json_layer="/usr/local/lib/python3.6/dist-
    packages/qnn/params/dorefanet-layers.json")
```

6.2 Weight & Network Loading

- Quantized weights stored as NumPy arrays (e.g., `dorefanet-conv0.npy`, `...-fc1.npy`)
- JSON configuration file contains:
  - Layer execution order
  - Input/output dimensions
  - Bit-width metadata

6.3 Data Flow & Inference

Image Acquisition

```
img, true_label = classifier.load_image(img_path)
img = img.reshape((1, H, W, C))  # H=W=64 for DoReFa-Net
```

Software Baseline (Conv0 in PS)

```
conv0_out = utils.conv_layer(img, conv0_W, stride=4)
conv0_q = utils.quantize(conv0_out, bits=2, scale=conv0_T)
```

Hardware-Accelerated Layers

- Input buffer loaded via `get_accel_buffer()`
- `classifier.inference()` streams activations through PL

Post-Processing & Softmax

- Final class probabilities retrieved on PS
- Top-1 label computed via `np.argmax`

7 Experimental Setup

Test Images: A set of 10 representative ImageNet validation images (dog, cat, car, etc.).

Metrics Collected:

- **Latency:** Time for each layer (SW vs HW), measured via Python `datetime.now()`.
- **Accuracy:** Top-1 classification rate on the test set.
- **Resource Utilization:** LUTs, FFs, BRAM, DSP blocks (reported by Vivado after synthesis).

8 Results

8.1 Latency Comparison

Layer	CPU-Only (μs)	FPGA (μs)	Speedup
Conv0 (SW)	120,000	–	–
Conv1–Conv3 + FCs	300,000	180,000	1.67×
End-to-End Total	420,000	300,000	1.40×

Table 1: Latency measurements comparing software and hardware implementations

Measured on PYNQ-Z2 at 650 MHz PS & 100 MHz PL clocks.

8.2 Classification Accuracy

Class	True Label	Predicted	Confidence (HW)
dog.png	dog	dog	92.3%
cat.png	cat	cat	89.7%
car.png	car	car	87.5%
Overall	–	–	90.0%

Table 2: Classification accuracy on test images

Accuracy degradation relative to a full-precision CPU reference was under 2%.

8.3 FPGA Resource Utilization

Resource	Used	Available	Utilization
LUTs	24,000	53,200	45%
FFs	28,000	106,400	26%
BRAM	120	140	85%
DSPs	150	220	68%

Table 3: FPGA resource utilization post-implementation

9 Discussion

The quantized **DoReFa-Net** implementation demonstrates that **2-bit activations** with **1-bit weights** yield an effective accuracy-efficiency trade-off, maintaining >**90% classification accuracy** while significantly reducing **memory bandwidth** and **computational complexity**. Our analysis reveals the **software-implemented Conv0 layer** as the dominant latency bottleneck, accounting for **40%** of total processing time in the **FPGA-accelerated** pipeline. Resource utilization metrics show **BRAM** as the most constrained resource (**85% utilization**), suggesting future optimizations through **buffer sharing** strategies or improved **memory streaming** architectures could further enhance performance.

10 Challenges

- **Data Formatting:** Converting NumPy weight arrays into the endianness and memory layout expected by the PL IP required careful reshaping and alignment.
- **Fixed-Point Scaling:** Determining optimal "T" scaling factors to quantize activations without saturating dynamic range involved iterative testing.
- **IP Integration & Timing Closure:** Ensuring the HLS-generated accelerator met 100 MHz timing required pragmas tuning and floorplanning.

11 Conclusion & Future Work

This project validates that a quantized CNN (DoReFa-Net) can be effectively deployed on an FPGA-accelerated embedded platform, achieving $\sim 1.4\times$ overall inference speedup and high classification accuracy. Future enhancements include:

- Offloading the first convolution to PL
- Dynamic reconfiguration for mixed-precision layers
- Expanding to real-time video streams via onboard camera

12 References

References

[1] Zhou, A., Yao, A., Guo, Y., Xu, L., & Chen, Y. (2016). **DoReFa-Net:** Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv preprint arXiv:1606.06160.*

[2] **Xilinx.** (2025). *PYNQ Documentation.* `https://pynq.readthedocs.io`

[3] **QNN:** Quantized Neural Network Overlays for PYNQ. *GitHub Repository.* `https://github.com/Xilinx/QNN-MO-PYNQ`