FPGA-based Embedded CNN for Rock–Paper–Scissors Image Recognition

Chengwen Chu (cc5397), Chengcheng Xu (cx2355), Linxioa Wu (lw3227), Harvey Lu (hl3999), Mingyuan Zheng (mz3143)

1. Motivation and Objectives

The goal of this project is to implement a small convolutional neural network (CNN) on an embedded FPGA platform to classify hand gestures into “rock,” “paper,” and “scissors.” After receiving an image, the system performs front-end capture and preprocessing, converts the image to a 32×32 single-channel greyscale tensor, and sends it to a CNN accelerator implemented on the FPGA; the final classification result is then shown on the board’s output interface.

The learning objectives are:

- Understand the complete deployment flow of a CNN image classifier on an embedded platform (training, quantization, inference).
- Gain hands-on experience with FPGA-based hardware accelerators, memory/data-path design, and hardware–software co-design for image processing applications.

2. System Specification and CNN Architecture

2.1. Input and Output Specification

- Input image: rock–paper–scissors hand gesture loaded from an offline dataset, preprocessed to 32×32×1 greyscale.
- Classes: 3 classes (Rock, Paper, Scissors).
- Output: a length-3 probability vector; the class with the maximum probability is selected as the prediction.

2.2. CNN Network Structure

- Conv1:
  - ♦ Kernel size 3×3, filters = 4, stride = 1, padding = valid.
  - ♦ Input: 32×32×1, output feature map: 30×30×4

- MaxPool1:
  - ♦ 2×2 max pooling, stride = 2, output: 15×15×4

- Conv2:
  - ♦ Kernel size 3×3, filters = 8, stride = 1, padding = valid.
  - ♦ Output: 13×13×8

- MaxPool2:
  - ♦ 2×2 max pooling, stride = 2, output: 6×6×8

- Flatten: 6×6×8 → 288
- Dense (Fully Connected): 288 → 3, followed by softmax.

The network has about 1,203 parameters and roughly 81,936 MAC operations per inference, which is small enough to fit on a mid-range FPGA using fixed-point arithmetic.

3. Hardware Architecture Overview

- The overall system is divided into the following modules:

3.1. Image Loading and Preprocessing

- Load rock–paper–scissors images directly from on-board or external storage (e.g., SD card, flash, or pre-stored memory) instead of using a live camera.
- Perform resize to 32×32, RGB-to-greyscale conversion (if needed), normalization, and data formatting, then store the preprocessed image in on-chip RAM.

3.2. CNN Accelerator

- Implement the two convolution layers and two max-pooling layers with a pipelined architecture using fixed-point multiply–accumulate units and line buffers.
- Load trained weights from offline training, stored in BRAM/ROM.

3.3. Control Unit (FSM or soft-core CPU)

- Control the sequence of image loading, preprocessing, CNN start/done signals, and result read-out.
- Handle interaction with user interfaces such as buttons, LEDs, seven-segment displays, or VGA.

3.4. Display and User Interface

- Display the current predicted gesture class and confidence on the FPGA board output devices.

Dataflow

Step 1. The user provides an image of a “rock,” “paper,” or “scissors” hand gesture.
Step 2. The preprocessing module converts it to a 32×32 greyscale image and stores it in the input buffer.
Step 3. The control unit triggers the CNN accelerator, which performs Conv1 → Pool1 → Conv2 →. Pool2 → Flatten → Dense.
Step 4. The CNN outputs three scores; the control unit selects the maximum and maps it to the. corresponding R/P/S label or icon.
Step 5. The prediction is shown on the FPGA outputs, and the system waits for the next image.

4. Hardware/Software Tasks and Tools

4.1. Software side

- Use Python with TensorFlow or PyTorch on a PC to train the 32×32 greyscale RPS CNN model.
- Quantize the trained model (e.g., 8-bit fixed point) and export the weights in a format suitable for FPGA memory initialization.

4.2. Hardware side

- Use Verilog/SystemVerilog to implement the CNN core, memory interfaces, and control FSM on the FPGA.
- Use the vendor toolchain (e.g., Vivado/Quartus) for synthesis, place-and-route, timing closure, and resource/power estimation.

4.3. Verification

- Build RTL testbenches that feed offline test images into the CNN hardware model and compare FPGA outputs with the software reference model.
- On the physical FPGA, evaluate classification accuracy and latency using multiple real-time gesture images.

5. Schedule

Week 1: Finalize project topic, collect RPS image dataset, write proposal.

Week 2: Complete CNN model experimentation and training on PC; fix 32×32×1 architecture and. hyperparameters.

Week 3: Design full FPGA system architecture and interfaces (preprocessing, CNN core, control unit).

Week 4: Implement and verify Conv1 + Pool1 hardware, integrate with preprocessing datapath.

Week 5: Implement Conv2 + Pool2 + Dense, complete full CNN pipeline and functional simulation.

Week 6: Download design to FPGA board, perform real-time tests, measure latency, clock frequency, resource usage, and accuracy.

Week 7: Consolidate results, write final report, and prepare demo.