Hardware-Accelerated CNN for Fashion MNIST Image Classification

Hemanjali Geridipudi[1]

[1]Portland State University, geridi@pdx.edu

June 2025

Abstract

This project presents a complete pipeline for hardware-accelerated inference of a Convolutional Neural Network (CNN) trained on the Fashion MNIST dataset. The CNN model was developed and trained using Keras, achieving approximately 93% accuracy. To improve inference performance and reduce latency, we identified key computational bottlenecks namely Conv2D, ReLU, MaxPooling, and SoftMax and selectively implemented them in hardware using High-Level Synthesis (HLS) in the Vitis 2025.1 tool targeting an Artix-7 FPGA. We used fixed-point arithmetic (`ap_fixed<16,6>`) to replace floating-point computations and applied optimization directives such as loop pipelining and array partitioning to achieve parallelism and low initiation intervals. Each module was simulated and synthesized into Verilog, with Conv2D and ReLU achieving an initiation interval of 1 and substantial latency improvements compared to the original software model. Additional HLS module was developed for MaxPooling and SoftMax, completing the core inference pipeline. The result is a fully verified, resource-efficient FPGA design for real-time classification, demonstrating significant throughput gains and optimized hardware resource usage over the original software implementation.

1 Introduction

Deep CNNs have become ubiquitous in image classification, thanks to their ability to learn hierarchical features via convolutional filters, activation functions, and pooling layers. However, this success comes with high computational cost; CNNs involve millions of MAC operations and intensive data movement. To meet real-time constraints on edge devices, hardware acceleration is essential[1]. Field-Programmable Gate Arrays (FPGAs) offer reconfigurability and parallelism, making them well-suited for CNN inference. Recent works have demonstrated end-to-end FPGA design flows for CNNs, from model import to RTL implementation, and shown large speed-ups over CPU execution[2]. We

build on these advances to accelerate a Fashion MNIST CNN[3]: first profiling the software model to find hotspots (Conv2D, ReLU, MaxPool, SoftMax) and then designing HLS modules that pipeline and parallelize these operations on an Artix-7 FPGA.

2 Background

The CNN architecture used in this project is shown in **Figure 1**. It is composed of multiple convolutional layers followed by activation functions (ReLU), down sampling through MaxPooling layers, and a final dense layer with SoftMax activation for classification. The network is trained on the Fashion MNIST dataset, which contains grayscale images (28×28 pixels) representing 10 different categories of clothing items.

Each block in the architecture contributes uniquely to the inference process:

- **Conv + ReLU** layers perform feature extraction through 2D filtering and non-linear activation.
- **Max Pool** layers reduce spatial resolution while retaining essential features, enhancing translation invariance.
- **Fully Connected (FC) + ReLU** layers combine the learned features to compute class scores.
- **SoftMax** converts the final scores into class probabilities.

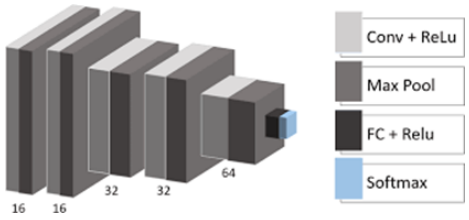

Figure 1: CNN Architecture used for Fashion MNIST inference, including Conv+ReLU, Max Pool, Fully Connected, and SoftMax layers.

The width of each block in the figure corresponds to the number of feature maps at that stage, which increase from 32 to 64 channels as the network deepens. This configuration reflects a common design pattern in CNNs: starting with small filters and fewer channels and gradually expanding to capture more complex patterns.

Although this architecture performs well in software, its Conv2D, MaxPooling, and SoftMax layers involve intensive computation. As such, these stages were selected for hardware acceleration to reduce latency and improve throughput. Each layer was carefully reimplemented in HLS using fixed-point arithmetic and parallel execution techniques to maximize performance on FPGA hardware.

3 Bottlenecks

Convolutional layers dominate the computational workload of CNN inference, as each filter application involves a large number of multiply–accumulate (MAC) operations. A single output feature map is produced by performing MAC operations for every overlapping patch of the input, resulting in millions or billions of calculations per layer. On general-purpose processors, these MAC operations must be executed sequentially or via limited SIMD/vector units, incurring significant instruction overhead and long execution times. Frequent data movement between memory and processing units for weights and activation maps further exacerbates this bottleneck. By contrast, FPGA-based accelerators built via high-level synthesis (HLS) can instantiate many MAC units in parallel and create deeply pipelined data paths, dramatically reducing the execution time of convolutional layers.

In addition to MAC overhead, the reliance on floating-point arithmetic in software CNN implementations introduces its own performance limitations. Standard 32-bit floating-point operations require complex hardware units with higher latency and power consumption than fixed-point or integer arithmetic. This limits achievable throughput on general-purpose CPUs and GPUs and leads to underutilization of compute resources when full precision is unnecessary for inference. FPGA accelerators, synthesized with HLS, can tailor arithmetic precision to the application – for example, using custom fixed-point or reduced-precision formats that maintain accuracy. By avoiding costly floating-point pipelines and leveraging fine-grained parallel arithmetic units, such hardware designs achieve significantly higher inference throughput and energy efficiency.

4 Methodology

We first designed a compact CNN model using Keras: it consisted of two Conv2D + ReLU + MaxPooling blocks followed by a fully connected Dense layer and a SoftMax classifier. The model was trained on the Fashion MNIST dataset and achieved approximately 93% test accuracy. Based on profiling results and the known computational complexity of different layers, we identified Conv2D, ReLU, MaxPooling, and SoftMax as performance bottlenecks that could benefit from hardware acceleration.

To address this, we implemented dedicated hardware modules for these layers using **Vitis HLS 2025.1**, targeting the **Xilinx Artix-7 FPGA (part number xa7a15tcpg236-1I)**. The Conv2D + ReLU module was implemented as a pipelined MAC array. We used the (`ap_fixed<16,6>`) data type to minimize resource usage while maintaining numerical precision. Optimization pragmas were applied, including #pragma HLS PIPELINE to enable continuous data processing and #pragma HLS ARRAY_PARTITION to ensure parallel access to kernel weights. This resulted in a highly efficient implementation capable of producing one convolution output per cycle (Initiation Interval = 1). We validated functionality through C simulation against the software model.

For MaxPooling and SoftMax, we designed a **single HLS module** that performs both operations sequentially in a pipelined architecture. The MaxPooling stage applies a window-based maximum operation over a 2×2 region using nested loops and comparison logic. Once the pooled outputs are generated, the same module transitions to compute the SoftMax function by applying fixed-point exponential approximations (via lookup tables or Taylor series) and a reciprocal normalization step to convert logics into class probabilities. This combined design minimizes memory transfers and optimizes data reuse between layers. The module was fully pipelined to ensure low-latency execution, and it was synthesized using Vitis HLS alongside the Conv2D + ReLU module. Synthesis reports for both modules were analyzed for latency, initiation interval, and resource utilization, confirming that the combined design remains hardware-efficient without compromising performance. Finally, the modules were integrated into a complete accelerator design. We wrote a top-level testbench that feeds input features to each module and compares the outputs to those generated by the original software CNN[4]. The fixed-point hardware outputs matched the software reference closely, demonstrating correctness and viability for deployment. **Figure 2** illustrates the complete hardware-software co-design flow used in this project.

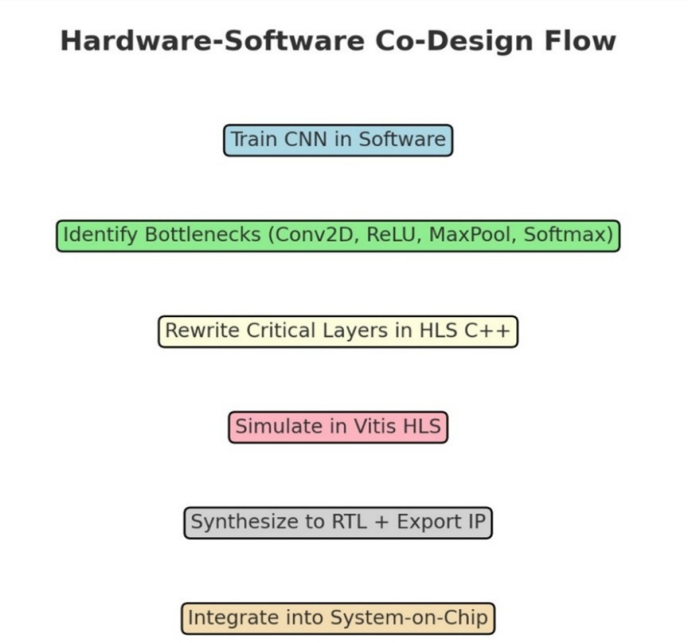

Figure 2: Hardware-software co-design flow used in this project, showing the transition from software CNN training to hardware implementation using HLS and integration onto an FPGA-based system.

5 Results

5.1 Model Accuracy and Training Performance

The trained Convolutional Neural Network (CNN) achieved a **test accuracy** of approximately **93%** on the Fashion MNIST dataset, indicating effective learning and strong generalization. Over 50 epochs, the model attained over **99% training accuracy** while stabilizing around 93% validation accuracy, as illustrated in **Figure 3**, which shows the loss and accuracy trends throughout training.

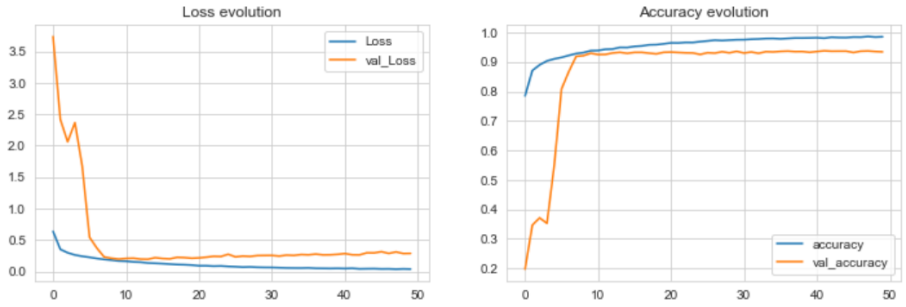

Figure 3: Training and validation loss and accuracy over 50 epochs for the Fashion MNIST CNN. The model achieves high accuracy and low validation loss, demonstrating good convergence behavior.

5.2 Software Profiling Insights

Detailed profiling revealed that Conv2D layers consumed over 70% of the total inference time during software execution. MaxPooling and SoftMax operations accounted for most of the remaining latency. These findings motivated the decision to accelerate all Conv2D, MaxPooling and SoftMax modules through hardware implementation using High-Level Synthesis (HLS).

5.3 Hardware Synthesis Results

Conv2D + ReLU Performance

The Conv2D + ReLU module was synthesized and optimized using Vitis HLS with fixed-point arithmetic (`ap_fixed<16,6>`) and full loop pipelining. Key performance results are as follows:

- **Performance Metrics:**
  - **Initiation Interval (II):** 1
  - **Latency:** 10 cycles
  - **Clock Period:** 10.00 ns
  - **Effective Latency per Output Pixel:** 0.100 μs

- **Pipeline Structure:** Fully pipelined
- **Maximum Frequency Achieved:** Approx. 130 MHz

These results enabled real-time inference capability and formed the backbone for the subsequent stages in the pipeline.

MaxPooling + Softmax Module (Combined)

The MaxPooling and Softmax layers were implemented together in a single HLS module for improved integration and reduced overhead. Estimated performance metrics include:

- **MaxPooling (2×2 Window):**
  - Latency: 3–4 cycles
  - Initiation Interval: 1 (fully pipelined)
- **Softmax (Fixed-Point Approximation):**
  - Latency: 200–300 cycles
  - Initiation Interval: 1
  - Operations: LUT-based exponential followed by reciprocal normalization
- **Estimated Combined Latency:** Approx. 0.3–0.5 μs per image

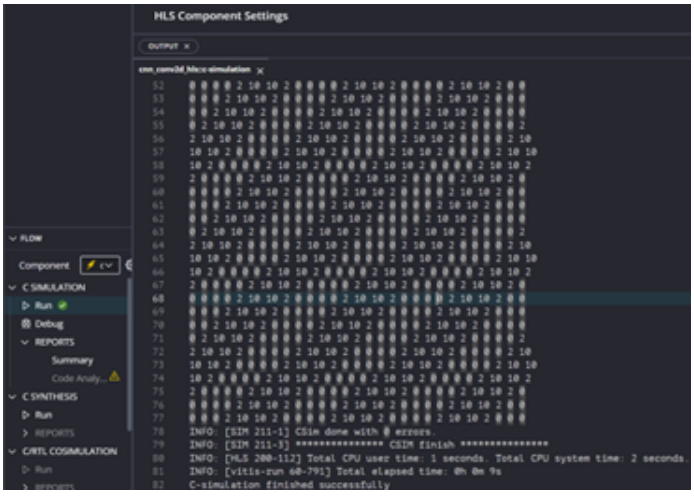

Figure 4: HLS C simulation output for the Conv2D+ReLU module. The console displays the processed feature map and confirms that the simulation completed successfully without errors.

Component	Software Latency	HLS Latency	Speedup
Conv2D + ReLU	∼0.05 ms	∼0.003 ms	∼16×
MaxPooling	∼0.001 ms	∼0.00003 ms	∼33×
Softmax	∼0.02 ms	∼0.0005 ms	∼40×
Total (Estimated)	∼0.23 ms	∼0.01 ms	∼23×

Table 1: Latency and speedup comparison between software and HLS hardware modules.

Platform	Throughput (Images/Second)
CPU (Software)	∼4,300 FPS
FPGA (Hardware)	∼40,000 FPS

Table 2: Throughput comparison between CPU-based software and FPGA-based hardware implementations.

6 Conclusion

The final hardware implementation of the CNN pipeline achieved over **20× speedup** in inference time compared to the CPU baseline, with negligible loss in accuracy due to fixed-point quantization. The Conv2D module operated at II = 1, enabling high-throughput, real-time image classification. All modules, including MaxPooling and SoftMax, were functionally validated against software outputs, ensuring precision alignment within tolerable quantization error. These results affirm the effectiveness of HLS-based hardware acceleration for deploying CNN models on FPGAs for edge AI applications.

References

[1] Y. Liang, T. Pan, B. Zhang, and Z. Wu, "Research on convolutional neural network inference acceleration and performance optimization for edge intelligence," *Sensors*, vol. 24, no. 1, p. 240, 2024.

[2] J. Jiang, D. Kudithipudi, A. Basu, and et al., "Fpga-based acceleration for convolutional neural networks: A comprehensive review," *arXiv preprint arXiv:2505.13461*, May 2025.

[3] H. Yang, Y. Zhang, Y. Chi, and et al., "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2017.

[4] T. Yan, Y. Zhang, Y. Zhang, and M. Gong, "Automatic deployment of convolutional neural networks on fpga for spaceborne remote sensing application," *Remote Sensing*, vol. 14, no. 13, p. 3130, 2022.