EE 271: Final Project Inference Engine on FPGA

Jay Pankaj Patel

April 3, 2025

Contents

1 Building Blocks

For our inference implementation, I am considering using Convolutional Neural Networks (CNNs). They are an algorithm used to recognize patterns in data. I believe the target data should be handwritten digits from 0–9. Neural networks are built using a collection of neurons organized into layers, each with their own weights and biases. The building blocks of CNNs are the following:

- **Tensor** – can be thought of as an n-dimensional matrix. We need to decide how big to make our tensor. I was considering making it smaller, like three dimensions, because our goal is to perform simple classification.
- **Neuron** – is a mathematical function that takes multiple inputs and gives a single output.
- **Layer** – is simply defined as a collection of neurons.
- **Kernel Weights and Biases** – unique to each neuron and are determined during the training phase of our network. We will train the model on a computer and extract the weights and biases to the FPGA for inference.
- **Differentiable Score Function** – represented as class scores on the output layer.

These are the high-level components needed. Using this knowledge, let's go over the proposed design.

2 Design

Our neural network is going to be as simple as possible because the emphasis of this project will be optimizing the operations required for the model to run. Hence, we are using the "Hello World" equivalent of Machine Learning. Our architecture is as follows.

2.1 Architecture

Our design will take a **28×28 grayscale image** as input.

After:

- A **convolution layer** will be used to apply 8 kernels to the input, generating **8 feature maps of size 26×26**, which are then passed to a max-pooling layer.
- A **max-pooling layer** will reduce the **spatial dimensions** of each feature map from **26×26 to 13×13**, maintaining 8 feature maps, and will pass the results to the softmax layer.
- For training, a **softmax layer** will process the **flattened feature maps** and produce **10 probability values**, each representing the likelihood of the image belonging to one of the 10 classes (digits 0–9).

- For the inference and implementation on hardware, a **hardmax layer** will be used as implementing softmax on hardware is resource intensive and will increase development time.

Here is a visual representation of these processes.

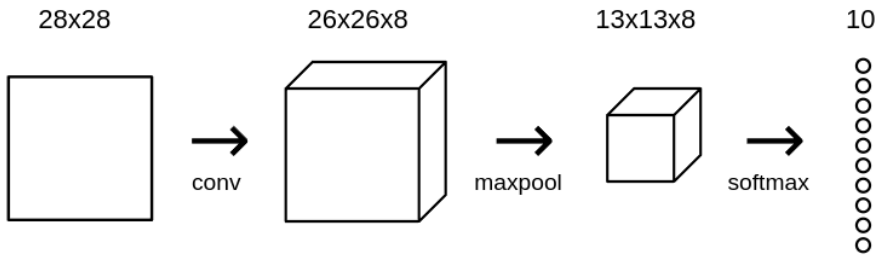

Figure 1: Visual representation of the CNN layers that will implemented as training in software

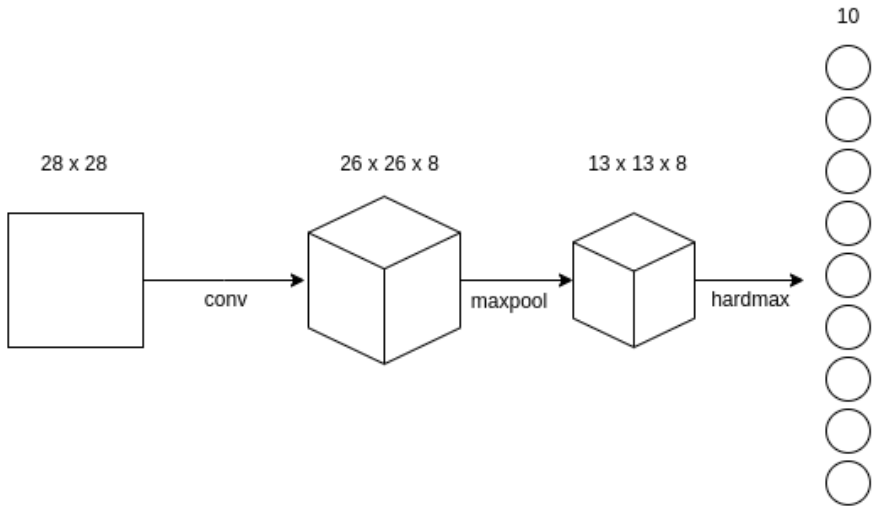

Figure 2: Visual representation of the CNN layers that will implemented in hardware

2.2 Model

A Python golden model has been created and should be used to validate the FPGA implementation. The Python model is not optimized for hardware, and operations should be adjusted for efficient computation on the FPGA. The model has been pre-trained in Python, and the trained weights and biases will be ported to the FPGA for inference. FPGA-specific design considerations will be discussed after detailing the theory behind each layer.

2.3 Convolutional Layer Design

2.3.1 Background

- **What is a feature?**
  A feature is an individual measurable property that a model uses to make a prediction or classification.
- **What is a feature map?**
  A feature map is the output produced when a convolutional kernel extracts relevant features from an input. In this field, the 'filter' is referred to as a *kernel*. Hence, I will be using "kernel" from this point forward.
- **What is convolution?**
  Convolution, in a mathematical sense, is an operation performed on two functions that produces a third function. The mathematical operation is defined as the integral of the two functions after one of them is reflected about the y-axis and shifted. Mathematically:

2.4 Equations

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

Graphically:

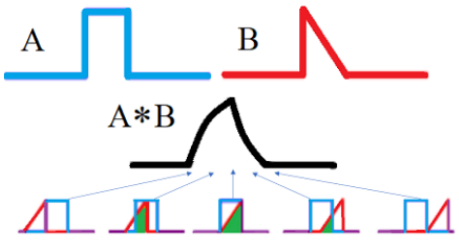

Figure 3: Convolution of two functions A (red) and B (blue) producing a third function describing the overlap (green)

Since we are dealing with discrete systems, convolution is defined using summation instead of an integral:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]\, g[n-m]$$

In CNNs, the most computationally expensive operation is the convolution layer. The convolution of the input feature map with the kernel results in an output feature map:

$$\boxed{y_j' = f\left(\sum_{i \in M_j} x_i^{l-1} * k_{ij}^l + b_j^l\right)}$$

Where:

- y_j^l : Feature map of the jth convolution kernel result of the lth layer.
- M_j : Represents the selection of the previous input feature map by the current convolution.
- x_i^{l-1}: Represents the previous input feature map.
- k_{ij}^l: Represents the ith weighting coefficient of the jth convolution of the lth layer.
- b_j^l : Represents the bias parameter of the jth convolution kernel of the lth layer.
- f : The nonlinear activation function.

2.5 Pooling Layer Design

2.5.1 Background

- **What is pooling?**
  Pooling is a technique used to downsample feature maps by aggregating adjacent values.
- **Why pooling?**
  Neighboring pixels in images tend to have similar values, leading to redundant information. Pooling reduces this redundancy while retaining important features.
- **How is pooling done?**
  Pooling is implemented using a max, min, or average value comparison over a small region.

2.6 Pooling Equation

A 3×3 convolution computes each output pixel using a weighted sum of a 3×3 region from the input:

$$\boxed{Y(i,j) = \sum_{m=0}^{2} \sum_{n=0}^{2} X(i+m, j+n) \cdot W(m,n)}$$

Where:

- $X(i,j)$ is the input feature map (28×28).
- $W(m,n)$ is the 3×3 convolution kernel (filter).
- $Y(i,j)$ is the output feature map after convolution.

Valid padding means no padding is added, so the output size is:

$$(28 - 3 + 1) \times (28 - 3 + 1) = 26 \times 26$$

And we repeat this 8 times because we have 8 kernels. Hence our final output will be 26 x 26 x 8.

2.7 Softmax Layer Design

2.7.1 Background

- **What is an activation function?** An activation function allows our model to achieve non-linearity as most things in real life are not a linear equation. By doing so, it allows our model to achieve higher accuracies.
- **What is softmax?** Softmax is an activation function that converts an arbitrary set of numbers into a probability distribution.
- **Why Softmax?** Softmax is useful when you have a multi-class model as we do here, (0-9 are different classes).

2.8 Softmax Equation

Mathematically, given an input vector $x = [x_1, x_2, ..., x_n]$, the Softmax function is defined as:

$$\boxed{s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}}$$

where:

- x_i represents the raw input (logit) for class i.

- e^{x_i} represents exponentiation of the input values.
- The denominator is the **sum of exponentials** over all input values.
- The function ensures that all outputs sum to 1.

2.8.1 Sample Calculation

1. Compute exponentials:

$$e^{-1} = 0.368, \quad e^{0} = 1, \quad e^{3} = 20.09 \quad e^{5} = 148.41$$

2. Compute denominator (sum of exponentials):

$$0.368 + 1 + 20.09 + 148.41 = 169.87$$

3. Compute Softmax probabilities:

$$\begin{aligned} s(-1) &= \frac{0.368}{169.87} = 0.002 \\ s(0) &= \frac{1}{169.87} = 0.006 \\ s(3) &= \frac{20.09}{169.87} = 0.118 \\ s(5) &= \frac{148.41}{169.87} = 0.874 \end{aligned}$$

Thus, the final probability distribution is:

$$[0.002, 0.006, 0.118, 0.874]$$

This means the model is **87.4% confident in class 5** and assigns lower probabilities to other classes.

2.9 Hardmax Layer Design

2.9.1 Background

- **What is hardmax?** Hardmax is a function that converts a set of values into a one-hot encoded vector, where only the index with the highest value is set to 1 and all others are set to 0.
- **Why Hardmax?** Hardmax is useful during inference when we want to make a final decision and select the most likely class. It is deterministic and discrete, which makes it appropriate for classification output.

2.10 Hardmax Equation

Given an input vector $x = [x_1, x_2, ..., x_n]$, the Hardmax function outputs a vector $h = [h_1, h_2, ..., h_n]$, where:

$$h_i = \begin{cases} 1 & \text{if } i = \arg\max_j x_j \\ 0 & \text{otherwise} \end{cases}$$

where:

- x_i represents the raw input (logit) for class i.
- $\arg\max_j x_j$ returns the index of the maximum value in the input vector.
- The result is a one-hot vector with a 1 at the index of the largest value.

2.10.1 Sample Calculation

1. Input vector:
$$x = [-1,\ 0,\ 3,\ 5]$$
2. Find index of maximum value:
$$\arg\max(x) = 3 \quad (\text{since } x_4 = 5 \text{ is largest})$$
3. Convert to one-hot encoding:
$$h = [0,\ 0,\ 0,\ 1]$$

Thus, the final hardmax output is:

$$[0,\ 0,\ 0,\ 1]$$

This means the model predicts class 3 with full confidence and disregards the other classes completely.