Handwritten Digit Classification System Powered by the FPGA

Lucas Ferreira
Lina Tinnerberg

March 5, 2024

Contents

1 Project Preparation **1**
1.1 Getting familiarized with CNNs 1

2 Setting Up Your Project Environment **2**
2.1 Installing Python and Creating a Virtual Environment 2
2.2 Training the MNIST Detection Model 2
2.3 Testing Model Inference 3

3 Workflow **3**
3.1 Behavioral Model 3
3.2 Hardware accelerator 4
3.3 Collaboration and Support Guidelines 4

1 Project Preparation

1.1 Getting familiarized with CNNs

Convolutional Neural Networks Explained (CNN Visualized)
https://www.youtube.com/watch?v=pj9-rr1wDhM

How Convolutional Neural Networks Work - CNN's #1
https://www.youtube.com/watch?v=UzfRnZqPbCI

What is pooling? - CNN's #3
https://www.youtube.com/watch?v=KKmCnwGzSv8

Fully Connected Layer - Disregard the training part
https://youtube.com/shorts/kQl45ophSVQ?si=Xk0yW382xDYX6AcI

2 Setting Up Your Project Environment

2.1 Installing Python and Creating a Virtual Environment

Step 1: Start by creating a dedicated folder for your project to keep everything organized.

```
mkdir mnistDetection
cd mnistDetection
```

Step 2: Use ‘virtualenv‘ to create an isolated Python environment. This allows you to manage dependencies specific to this project without affecting other projects or your system’s Python setup.

```
pip install virtualenv
virtualenv mnistDetectionEnv
```

Step 3: Activate your virtual environment. Once activated, any Python or pip commands will use this environment’s settings and packages.

```
source mnistDetectionEnv/bin/activate
```

Your terminal prompt will change to show the environment name, confirming it’s active. Use ‘pip list‘ to see installed packages, and ‘deactivate‘ to exit the environment. If you close the terminal the environment will automatically be be deactivated and you will have to activat it again when starting the work next time. ne

Step 4: With mnistDetectionEnv activated, update pip to its latest version and install TensorFlow and Matplotlib, essential libraries for machine learning and data visualization.

```
pip install --upgrade pip
pip install tensorflow matplotlib
```

2.2 Training the MNIST Detection Model

Step 1: Go through the code, and run the provided Python script to train your model. This process involves using a dataset to teach your model how to recognize and classify images.

```
python trainMnistDetector.py
```

Step 2: This creates a ‘flite-models‘ directory with your trained model, ready for use, which contains your saved trained and quantized model.

Visualizing the Model with Netron

Step 1: Install Netron, a tool that lets you visually explore neural network models. This helps understand the model’s structure and data flow, its parameters (i.e weights and biases), and its quantization parameters (scale, zero-point, data-types, etc).

```
pip install netron
netron tflite_models/mnist_model_quant8.tflite
```

Step 2: Now visit 'localhost:8080' in your browser to view your model. Explore your model by clicking in the different layers. There you can find all parameters to gain insights into how your model works.

Note that filters in Netron are presented as <input channels x number of kernel rows, number of kernel columns, output channels>. For instance, if an image <1x100x100x1>undergoes Conv2D with a <100x3x3x1>filter, means that the same input image is convolved with 100 different 3x3 kernels, resulting in a <1x98x98x100>output feature map.

Important: When writing your behavioral model, note that the weights in Netron are the real convolution weights (not cross-correlation)! So in order to obtain the same results as Conv2D in Tensorflow, remember to mirror horizontally and flip vertically the kernel weights, i.e the convolution kernel <1x3x3x1>element $K_{3,3}$, is equivalent to the first row, first column element of a cross-correlation filter.

Also note that the dimension of each feature map is 98x98, as a valid convolution disregard what happens at the edge of the input image. Here, $Img_{rows} - Kernel_{rows} + 1 = FMap_{rows}$ or $100 - 3 + 1 = 98$.

Step 3: Read carefully the paper "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference" by J. Benoit.

2.3 Testing Model Inference

Step 1: Test your model's inference capability by running another script. This demonstrates how the model makes predictions on new data.

```
python visualizeInferenceResults.py
```

3 Workflow

Now you can start the process of implementing your trained and quantized network in the FPGA, focusing in obtaining the maximum performance the board can handle.

3.1 Behavioral Model

Our first suggestion is that you start by making a Python or Matlab behavioural model that produces the same inference results as the provided algorithm. By doing so, you will a deeper understanding of what operations have to be done in the FPGA during inference, and also be able to verify the dataflow. The

behavioural model should make the exact same calculations as the hardware accelerator that you will later create, which means it should be bit accurate.

To test your behavior model you need to load the data from the mnist data set. An example of how that is done is shown on row 15 to 17 in the file visualizeInferenceResults.py.

3.2 Hardware accelerator

When your behavioral produces the same results as the provided algorithm (also called golden model), start to think about the schedule of operations in the FPGA, how many accelerators per layer, how parallelism can be exploited over the different layers, and if you need to buffer any data to match the throughput between layers. Read the paper "A high performance FPGA-based accelerator for large-scale convolutional neural networks" by Huimin Li to learn some common methods on how to schedule and paralyze your accelerator.

When designing your hardware accelerator it's important to take the big picture into consideration and consider the interface for your design so it's compatible with other necessary components. A seminar that goes into more detail about this will be given.

3.3 Collaboration and Support Guidelines

We suggest that you create a discord group between yourselves, to share problems and technical solutions across the different teams! Help each other.

You are all grown-ups now, this project is an excellent chance to learn a lot - Do not copy code.

Do not expect that us TAs are going to solve the cryptic error that you encounter at tool X, or a segmentation error in your code.

Embrace the learning process and apply your knowledge to successfully complete the project. Experimentation and collaboration are key to overcoming challenges.