Neural Networks on a FPGA

Degree programme: BSc in Computer Science | Specialisation: IT-Security
Thesis advisor: Prof. Dr. Bernhard Anrig, Prof. Andreas Habegger
Expert: Dr. Federico Flueckiger

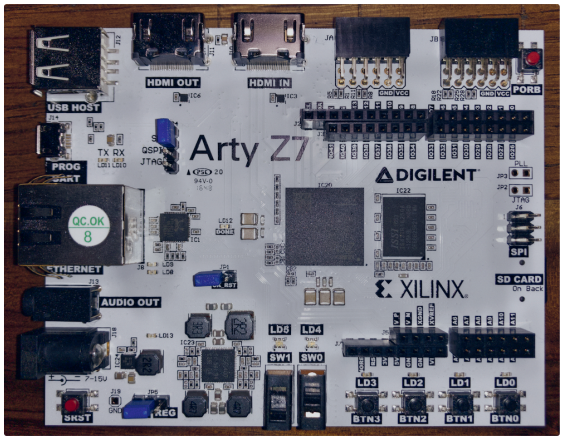
In today's world, neural networks are ubiquitous. They are used from voice recognition to driving autonomous vehicles. One of their main drawbacks is that they are very computationally expensive, which limits their practicality in mobile and embedded applications. In this work, we explore one approach to mitigate this flaw: Running a quantized neural network on a FPGA.

Introduction

A field-programmable gate array (FPGA) is an integrated circuit (IC) that can be programmed to represent any digital circuit, as long as the design does not exceed the resources of the specific device. The description of such a circuit is done in a hardware description language (HDL). To do this for something as complex as a neural network, a top down approach was chosen. We first evaluated a reasonable neural network architecture using Python, then implemented it in plain C and finally ported the inference part of the network to the FPGA using Verilog.

Evaluation of the Network Architecture

To find a reasonable network architecture to implement, we tested convolutional neural networks (CNN) and fully-connected neural networks of various sizes. The networks were trained and evaluated using the MNIST dataset, which consists of 70'000 handwritten digits (0 to 9). As a result, a fully-connected network architecture with 2 hidden layers (64 nodes per layer) was chosen. We chose a fully-connected network since it offered reasonable accuracy (around 94%) while being easier to implement than a CNN.



The Arty Z7-20 development board.

CPU Implementation

We then implemented this architecture in plain C using only POSIX libraries. To train the network we mostly used floating point values, since this is required for stochastic gradient descent to work. The inference part of the network was implemented twice. The first implementation used regular float values while the second used only integer values. For this quantized inference to work, we designed a conversion function to transform the learned parameters of the neural network (weights and biases) into integers of various sizes (8 – 32 bits). This has two advantages: We save space when storing the parameters of the network (e.g. the weights use 4 times less storage) and it reduces the computational requirements of inference, since an integer multiplication is much simpler than a floating point one. Most importantly, quantized inference is in our case only slightly less accurate than normal inference (less than 0.1%).

FPGA Implementation

After testing and verifying the quantized inference, we implemented it on the FPGA. We started by designing a block diagram, writing the corresponding Verilog code, simulating and testing the design in the Vivado Simulator. To get the parameters of the trained network into the design, we used a helper function in the C code, which exported them in a format usable on the FPGA. The inference part was then packed as an IP core and wrapped into a bigger design containing components like a DMA Controller and various AXI Interfaces. These parts are needed, to actually run the inference on our target hardware platform, the Digilent Arty Z7-20. This enabled us to read the image data from the DDR memory of the Arty and stream it into our inference core. We created a custom Linux distribution that uses a special device driver to make our DMA controller accessible as a device file. Finally we were able to write an application that interfaces with this device, reads the MNIST data from a file on the SD card and writes it into the DDR memory. The data is then streamed into our inference core.

Pascal Daniel Liniger