Approximate Neural Networks in FPGAs

Mentor(s): Peter Milder, Emre Salman
Alexander Ahandour, Mariano Bello, William Wong and Sabreen Mostafa

Stony Brook University
Electrical and Computer Engineering

Abstract

As CNNs become more widespread, the question of how to reduce their power consumption arises. One method of accomplishing this is to lower the voltage level they operate at. However, doing so will sacrifice some accuracy. The question then is how much inaccuracy can the CNN tolerate before it becomes unacceptable?

In this project, we mirror these inaccuracies by deliberately injecting errors into our CNN. In software, these types of computations cost a lot of time, so we implemented this system in hardware to reduce the time spent performing calculations. This project is modeled after the MNIST CNN model, which identifies a number based on a 28 pixel*28 pixel image.

Introduction

One way to reduce the amount of power consumed by a neural network is to reduce its operating voltage. However, the penalty for the reduction of the operating voltage level is the reduction in accuracy. In some applications, such as face filters in social media apps, the loss of accuracy may not be a huge concern, although in other applications such as medical imaging, accuracy loss can be critical. It is therefore of paramount importance that the user of such an application knows exactly how much power reduction will affect the accuracy of the CNN.

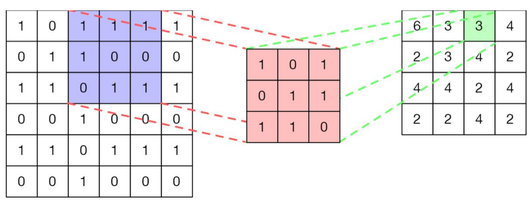
Because every convolutional layer and error injection is lengthy in terms of computation time in software, our goal was to implement a system in hardware that would shorten the amount of time needed to perform these operations. To do so, we decided to use an FPGA and design hardware modules in VHDL that would perform the task of the CLP and the Error Injector.

The Error Injector and CLP were designed as two separate modules. The Error Injector takes a 32 bit number and flips each of those bits according to a 24-bit number representing the error rate. The CLP takes an input image matrix supplied externally and convolves it with a user supplied weight matrix and stores the output in memory.

After building this design on the FPGA, a software driver written in C was used to dictate various parameters and control the start and end of hardware operations.

Background

An abstract representation of how a 2-D Convolution is performed



Data Types:

- Software: calculations done with floating point numbers.
- Hardware: calculations done with integer numbers.
- Conversion: multiply floating point by power of 2 e.g.:

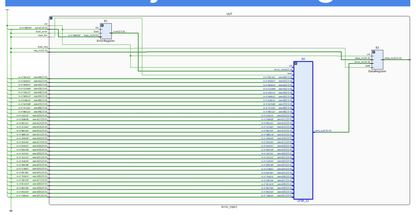
$$XX.XX * 2^1 = XXX.X \quad \text{(The decimal point has been shifted by 1 bit)}$$

- Truncate at new decimal position: Data type now an int
- In C, this is done by type casting a float to an int. By shifting the decimal point, any fractional bits shifted over get preserved as integer bits.

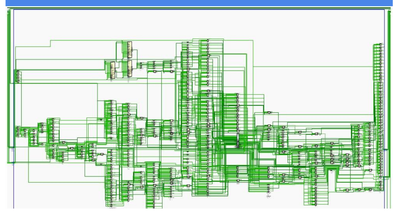
Pseudo-Random Number Generator

- Hardware: LFSR
  - Runs through all possible numbers before repeating based on a seed

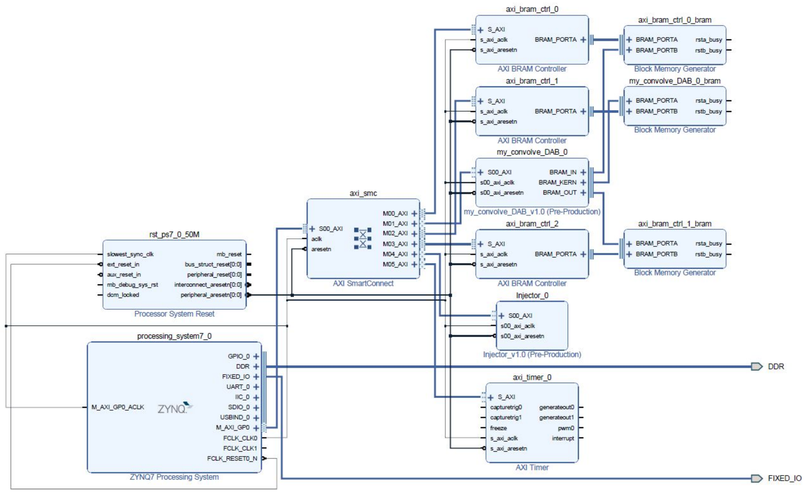
Error Injector Diagram



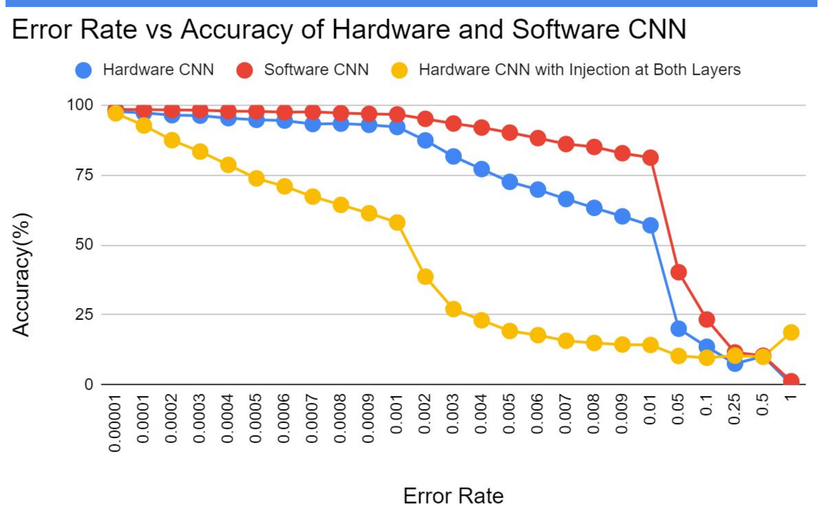
CLP Schematic



Block Diagram

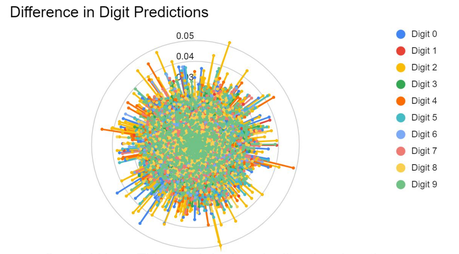


Results



Timing Data / **FC Layer Discrepancy**

Convolution Layer	Hardware Time	Software Time
Layer 1	56.559 s	109.161 s
Layer 2	73.293 s	165.480 s
Total Time	**129.851s**	**274.641 s**



Special Note: This graph looks a lot like the virus that doomed the Class of 2020

Conclusion

Overall, our project resulted in a device capable of evaluating neural networks using a Xilinx MiniZed board, as well as inject errors at the end of forward activation layers. First, model weights and biases are gathered from C header files in the project files. Next, test images are gathered from their own header file and are loaded into the CLP to be convolved. After each layer, error can be inserted at a given rate. Finally, the results are recorded and one can accurately model how a given network is impacted by error. This hardware and accompanying software framework is the result of our work this year.

We've all learned a lot about hardware design, control systems, Neural Networks, testing and verification, and working with FPGAs through this project. Although out design is functional and meets its target, there are ways we can improve our design, and working on this project has made us continuously think about how we can improve our design

Glossary

CNN: Convolutional Neural Network
CLP: Convolutional Layer Processor: Hardware designed to processes a convolution layer
FPGA: Field Programmable Gate Array: A device whose hardware configuration can be programmed
VHDL: A Hardware Description Language
LFSR: Linear Feedback shift register: A register configuration that allows random number generation based on careful placement of XOR gates
G.O.A.T.: Greatest of All Time: vis a vis Tony Olivo

Acknowledgements

First and foremost, we would like to thank our advisors **Professor Milder** and **Professor Salman** for mentoring, guiding, and fostering our growth for the whole year. We're sorry about the 2 AM slack messages, but extremely grateful for your wisdom, advice, and encouragement.

We would also like to thank **Tony Olivo** for letting us use his lab space and for being a great person in general. You're the G.O.A.T., Tony!

A special shoutout to **Professor Westerfeld** for checking in periodically to make sure my last brain cell remained intact

Our love goes out to the people on Stack Overflow asking the relevant questions 3 years ago

Thank you to all of our professors, friends, and fellow Class of 2020 seniors for a memorable and enjoyable 4 years and for helping us make it every step of the way. We love you all!

Team **A.N.N.E.X.**