Convolutional Nets on FPGA

Project Proposal

Ricson Cheng — `ricsonc`

Sree Vishant Prabhakaran — `sreevisp`

April 8, 2017

1 Summary

We are going to implement convolutional neural networks on an FPGA, which was previously explored by [2]. We plan to implement both training and inference steps on the FPGA. We will also explore tricks such as binarized neural nets **1** in our implementation.

2 Background

2.1 Field Programmable Gate Arrays

A field-programmable gate array is a programmable circuit board that allows the designer to implement code directly onto a set of hardware logic blocks on the array. FPGAs are typically programmed in a hardware description language in terms of logical gates and connections. They are generally harder to develop on as compared to a higher-level language such as C, but provide the benefit of being able to achieve maximum performance when implemented correctly. This is due to the fact that having a circuit block specifically designed to run a certain program ideally eliminates any unnecessary operations (such as extraneous loads from memory or otherwise wasted communication) during the execution of the program.

2.2 Convolutional Networks

Convolutional networks are a type of neural network which are very good at extracting features from image data.

A convolutional network takes as input a HxWx3 image (where H is height and W is width), and applies convolutions to the image, interleaved by activations in order to create nonlinearities in the output.

The output of each convolutional layer is a tensor in the form H'xW'xD, where H' and W' may be less than or equal to H and W, respectively, and D is the number of filters used in the convolutional layer.

The output of the last convolutional layer may also be passed through some fully connected layers in the network to obtain the final output.

3 Challenge

When implementing a convolutional network on FPGA, loop unrolling and loop tiling are both important factors affecting performance [2]. [2] explores the efficiency of unrolling and untiling loops to various degrees in the implementation of a convolution. They also explore loop pipelining, which starts on the next iteration of a loop before the current iteration is finished as an optimization.

One challenge of ours will be to make use of these optimization techniques. In addition, we must ensure that our code is amenable to these techniques – it is not possible to do loop pipelining unless the next iteration of the loop is at least partially independent from the current one.

An additional challenge is to make efficient use of memory accesses because memory bandwidth is limited. Doing this may involve data transformations and rearranging loops in order to reduce the number of accesses. This was also explored in [2].

Finally, another challenge in this project will be the actual implementation of a neural net on an FPGA. Neither of us has programmed FPGAs before, so we hope that this will be a good learning experience in working on and optimizing closer to hardware.

4 Resources

4.1 Hardware

We will need an FPGA, and have already talked to a TA about this.

4.2 Software

We will be writing all the code from scratch.

4.3 Previous Work

Design decisions when implementing convolutional net on FPGA has previously been explored by [2] (This was explained in an above section).

Another slightly unrelated paper, [1] proposes neural networks which only output a binary instead of a floating point value. We will attempt adapting this to our implementation since it may lead to better performance.

Finally [3] is reference text on many of the implementation details for putting (non-convolutional) neural networks on FPGA, and we will use it in order to guide our implementation as well.

5 Goals and Deliverables

5.1 Plan to Achieve

1. Implement a fully convolutional network on FPGA (This is our final goal, after all)
2. Optimize code until it is significantly faster (5x) than our baseline We expect this to be possible due to similar speedup factors in [2].
3. Achieve good classification accuracy on the MNIST dataset, at least 95%. This should be possible because the MNIST dataset is comparatively easy for convolutional nets to do well on. We may also try more challenging datasets if time allows, however this is not the focus of the project.
4. Adapt the work of [1] in our implementation. We expect this to speed up our implementation because single-bit computations may be much faster than dealing with 32-bit floating point numbers.
5. Analyze the speedup achieved due to different optimizations and data transformations

5.2 Hope to Achieve

1. Implement training on the FPGA. This may be significantly more difficult than implementing inference.
2. Implement fully connected and pooling layers. Many convolutional networks have both pooling and fully connected layers, however, they are not strictly necessary.

5.3 Evaluation

We will evaluate our implementation by comparing it to several benchmarks

1. Our baseline implementation
2. Other FPGA imeplentations (A table of many others is given in [2])

3. A GPU-implementation
4. A CPU-implementation

We will compare both compute speed and power efficiency. We expect that our implementation should have performance close to, or matching the results in [2]. We also expect that it should be much more power efficient than the CPU and possible GPU implementations.

5.4 Parallelism Competition

At the competition, we expect to show graphs which plot the speedup effects from different optimizations made to the network.

In addition, we also hope to show classification results from the network itself to demonstrate how accurate it is / how well it works.

6 Platform Choice

Our platform is the FPGA. This platform makes sense because FPGAs are well suited to tasks involving high parallelism, and doing large convolutions/matrix multiplications is one of those tasks.

Furthermore, compared to GPUs, which are the current dominant piece of hardware used to operate convolutional networks, FPGAs are far more power efficient.

On the other hand, FPGAs have drawbacks such as limited bandwidth, which is further explained in the section above (Challenges).

7 Schedule

7.1 Week 1 (April 9 - 15)

- Familiarize ourselves with the basics of programming on FPGA
- Identify a suitable neural network algorithm to implement
- Compute theoretical performance bounds to align expectations and goals

7.2 Week 2 (April 16 - 22)

- Develop baseline implementation using GPU
- Create test suite
- Begin initial implementation of neural net on FPGA

7.3 Week 3 (April 23 - 29)

- (The checkpoint is this week)
- Complete initial working FPGA NN implementation (if not already done)
- Compare results with baseline
- Identify areas for improvement and optimization

7.4 Week 4 (April 30 - May 6)

- Optimize FPGA network implementation
- Attempt features from ”Hope to Achieve” as time permits

7.5 Week 5 (May 7 - 12)

- Wrap up any remaining bugs/features/optimizations
- Prepare final presentation

References

[1] Courbariaux, Matthieu and Bengio, Yoshua, BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1, CoRR, 2016

[2] Zhang, Chen et al., Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks, FPGA, 2015

[3] Omondi, Amos and Rajapakse, Jagath, FPGA Implementations of Neural Networks, Springer, 2006