VLSI Digital System Design Project ECEN 5263 : VLSI Tiled Convolution Neural Network (CNN) Architecture v1.0.1

James E. Stine
Electrical and Computer Engineering Department
Oklahoma State University
Stillwater, OK 74078, USA

The project in this course is designed to test your knowledge of the topics discussed in lecture and apply them to a real-world problem. The project will consist of a task and you should work individually and following the submission guidelines. Under no circumstances will copying be allowed on this project and any piece of software, hardware, or any other item that is copied may result in a failing grade for this class. However, I do encourage that discussion should be exhibited between all individuals to understand the problem. If you need any clarification on what constitute an illegal action, please feel free to contact me.

In this project, you will be doing something important in the area of deep learning. Deep learning is a relative old field with a new flavor or spin on it. It is basically computation that is done via some repetitve action to repeat what is done in nature. Specifically, this project will deal with something called a tiled convolution neural network [1]. A wonderful video describing some of these items comes from the following URL: https://www.youtube.com/watch?v=u6aEYuemt0M.

The modern Convolutional Neural Networks are based on work from the research paper [2]. In this paper, the authors propose a neural architecture called LeNet 5 used for recognizing hand-written digits and words. The name is inspired from signal and systems theory in processing signals using the convolution operator or:

$$y[n] = x[n] * h[n] = \sum_k x[k] \cdot h[n-k]$$

Although convolution is extremely useful it can be computationally complex for its implementation in hardware.

It is neither my wish or desire to make something for a project that does not have some redeeming educational value or is something that is not feasible. This area is rich in intellectual material well beyond the scope of the class and my hope is to appeal to some sense of science and engineering. Also, my hope is encourage and motivate you that you can achieve something if you work hard and attempt something that can lead to advancement in this field. I hope that I can provide some sense of purpose as well as provide some ambiguity that will allow you to embellish and/or innovate within this project.

My hope for this project is to create a convolution neural network (CNN) in hardware, specifically targeted for an Application Specific Integrated Circuit (ASIC) implementation. However, I would like to leave things open and if you wish to use what we have learned this semester to integrate within a Field Programmable Gate Array (FPGA), you are welcome to target this option. ASICs are far faster than FPGAs and have better applicability towards this area, but FPGAs are just as viable and may have some embedded processing power that ASICs do not. This is illustrated by the some of the FPGA boards that are produced by Digilent/Xilinx and Altera [1].

I. CNN Architecture

For this project, I would like you to use the modules you used for your previous homeworks. That is, I would like to have a `float16` version of this CNN architecture implemented. If you could not get your `float16` adder or multiplier to work, you are welcome to use the ones I have posted to D2L.

Hardware can be as complicated or simple as you like it. I would like you to implement this architecture with complexity and speed in mind. Therefore, to make sure you implement this unit with this concept, please try to minimize the amount of hardware. Significant deductions will be made if you utilize too much hardware, as will be explained later. I am open to multiple ideas, so I am not going to hold you one specific architecture. Try to get the project working and then move from there if you wish to make it more robust or optimized. I can work with students with projects that are sufficiently developed for conference publications and/or along with your graduate adviser, if needed, as well.

A. Convolution Neural Network Computation

The main part of your unit should compute the convolution neural network. This works from a basic matrix multiplication point of view. You should have a basic matrix that gets multiplied by several filter banks and then added together. The basic equation is computed as:

$$\sum_i w_i \cdot x_i + b$$

[1] I have some Digilent Zybo Zynq-7000 boards you are welcome to borrow if you are so inclined to try to implement for a real-time application

Most CNN architectures have several computational elements associated with it. These involve the convolution layer, pooling layer, and fully-connected layer to form a full 3-step architecture. A good website that describes this process is available at: http://cs231n.github.io/convolutional-networks/. A popular form of CNN is utilized within this project. It is sometimes called a ConvNet and is based on the work from [3].

To make this project have some tangible matrix to work with, we will be using 256x256 images from a Video database [4]. I will provide a sample image, but you are welcome to use any of the images at the referenced website. To make things easier, you should use a black and white image to make the matrix multiplication easier (i.e., limit the size). You are welcome to try an implementation in color, but you will need to perform the CNN three times to account for its color content (i.e., Red, Green and Blue values). This image will be input into your architecture through your testbench. MATLAB code is given later in this document to show how to create your testbench. However, you are welcome to use C, Java or Python to accomplish the same task.

Each architecture should take your image defined by its size and process it accordingly with values that define the CNN. These are typically called hyperparameters. These are, as follows:

- Number of filters, K
- Spatial Extent, F
- Stride, S
- Zero Padding, P

for a given matrix size given by $W_1 \times H_1 \times D_1$. Since you will be asked to minimally use black and images [2], the volume will only be defined by its Width and Height and not depth. For color images, the value of Depth is typically 3. For this project, I would like you to use the following minimal parameters. You are welcome to augment the values, if you wish:

- $W_1 = 256$, $H_1 = 256$, $D_1 = 1$
- $K = 1$, $F = 2$, $S = 1$, $P = 0$

The values of F indicate the spatial extent of the convolution layer neurons. These are typically set as filters based on a specific elementary function (e.g., most-often something called a sigmoid function). For your filters (i.e., values of w_i and b_i), please use values that can be $[-1, 1]$ or values between -1 and $+1$. You are welcome to use something easy to get things working – that is, use all 1s to guarantee easy debugging.

Using the formulas from the website or just simple matrix algebra, it should be easy to see the output volume or size of your final output is:

- $W_2 = (W_1 - F + 2 \cdot P)/S + 1 = (256 - 2 + 2 \cdot 0)/1 + 1 = 255$
- $H_2 = (H_1 - F + 2 \cdot P)/S + 1 = 255$
- $D_2 = K = 1$

To cut down on the processing, it might be advisable to change the parameters to cut down the processing time. For example, changing your Stride to $S = 2$ would cut your processing in half.

[2] the clock.tiff example is given on D2L for you to use

You are also welcome to augment the project by making the values of the weights, filters or spatial extent more realistic. There are several MATALB examples that are useful to understand more realistic ideas with deep learning [3]. This project is meant to be simplified to get you to understand the power in creating hardware. There has been many enhancements to MATLAB over the years to get deep learning architectures implemented computationally quickly by using Graphical Processor Units (GPUs). Again, you are welcome to add some of these for your project.

To make sure you do not have too many blocks and incur extra area, please limit your number `float16` units to F units. That is, you cannot use more than F units of `float16` multipliers or adder/subtracters. If you want to modify this, please discuss this with me for your justification. I am willing to accomodate changes if they are strategically and intelligently chosen. My main goal for adding this constraint to avoid 256 units and have unrealistic area and delay results. To make this work, you should use registers to store intermediate results.

Typically the weights are stored in memory, so please assume this is a behavior-level Verilog file that will not be included in your area/delay. You can use easily create a simple Verilog memory model to accomodate these values.

II. Basic Baseline Project

As stated earlier, I would like you to use `float16` as your computation units. Please try to create a testbench that takes as input each 8-bit pixel and then convert to its representable `float16` version. Your testbench should feed the image and your unit should output your new output matrix accordingly. The problem will be to understand how the computation works, implement the Verilog according to your understanding, and demonstrate that this works.

I would recommend storing your input matrix into registers and then using these registers to do each computation cycle. It will probably take several cycles to process the image and you should design your unit to be as simple as possible. Use of `float16` is a benefit in that each filter can be a representable `float16` output. You are welcome to reconstruct the image back to a `uint8` datapath accordingly.

A. Image Testbench

You should take a sample image from [4] or the provided image on D2L and use some way of converting each pixel to its representable 8-bit value. The easiest way to do this is through MATLAB as MATLAB has been adapted to handle images efficiently and effortlessly. To get your image into MATLAB, the following code should work:

```
clc; clear;
% Read the sample image
clock = imread('clock.tiff');
```

[3] See Google link for MATALB on Top 7 ways to get started with deep learning and MATLAB

Once you read in the image, you can easily write this out to a text file by using the following code assuming you are writting this to a file called `vector1.txt`:

```
vec1 = fopen('vector1.txt', 'wt');
fprintf(vec1, '%d,', clock);
fclose(vec1)
```

Any file you write, should always end in closing it. Please be aware that if you do not close a file once you open it, you could theoreically cause your computer to crash as it will create a pointer that goes on endlessly. That is, you will just use all your memory up and your computer may forget where your Operating System is. So, please be careful with this process if you are not familiar with it. However, if you use the above snippet of code in sequence as its written, it should be safe.

If you wish to display an output image, you can reverse the process by using another MATLAB function:

```
A1 = importdata('vector1.txt');
figure; imshow(uint8(A1))
```

III. Project Details

There are several project details that I have created to make this project less stressful and more fun (at least, I hope so). If you have any suggestions or comments, they are always welcome and I can modify things accordingly.

1) There is no final in this class, therefore, the project is due by May 14, 2017 no later than midnight. Try to show me you have demonstrated what you learned in this class by showcasing anything we included within this class. I am open to any modifications or new features you want to add.
2) I would like you to show me the easiest and most efficient way what you accomplished. I used to always require a report, but I understand some might be better at handing in a poster or performing a demo. Therefore, I will accept a demo, a poster, a report, summary sheets, youTube videos, and so on. But, something that gives me a complete picture of what you did and what worked. Again, I want you to have fun with this project as I think it is something that has good scientific value and can be used on your CV for demonstrating what you did in graduate school.
3) MATLAB has some really neat examples on deep learning using convolutional neural networks. Feel free to use any of these for extra credit to augment your project. However, many of these examples require GPUs from NVIDIA that enable their CUDA hardware. The website listed previously, http://cs231n.github.io/convolutional-networks/ has some Python routines using a numerical package called `numpy` that works well. There are also some other MATLAB items that work with your non-CUDA CPUs, if needed [5].
4) If you decide to utilize a demo, make sure you guide me through the process effortlessly, so that I understand what you did. Regardless of any method utilized for this project, you should still hand in your Verilog code and any associated software you utilized.
5) If you decide to use a poster or report, please try to document your project the best you can. I have added information on D2L to help you with these two options.
6) If you would like to use any hardware, as mentioned earlier, please let me know and I can let you borrow the hardware.
7) You should perform, at the very least, synthesis of your design with proper loading. Please try to give me total hardware (area) and its delay per cycle. Also, give me a complete picture of how long it takes to process your image.
8) Some images at [4] might be larger than 256x256 and may alter your hardware requirements. At the very least, you should use 256x256 as your image size. If you use somethign else, besides `clock.tiff`, it may add extra processing time and/or resources.
9) You can use FreePDK45 standard-cell library or Synopsys library used for the demo for your synthesis. The operating conditions, loading, and any other details are at your choosing. Please do not include the weights in your synthesis - assume this is not part of your synthesis.
10) I would also like you to have power numbers from synthesis. You can use the Verilog code I gave in lecture to get a VCD file and then bring this through Synopsys. I will explain in class how to analyze this more in depth. However, the VCD should be sufficient to process your energy, at least preliminarily. Better estimations are given with Electronic Design Automation (EDA) tools like Synopsys PrimeTime and HSPICE.

References

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., pp. 1097–1105. Curran Associates, Inc., 2012, http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convo\lutional-neural-networks.pdf.

[4] "Video Test Sequence Database," http://sipi.usc.edu/database/?volume=misc.

[5] A. Vedaldi and K. Lenc, "Matconvnet – convolutional neural networks for MATLAB," in *Proceeding of the ACM Int. Conf. on Multimedia*, 2015, Available at http://www.vlfeat.org/matconvnet/.