EECE 4632: Spring 2022
Hardware-Software Codesign for FPGA-Based Systems
Preliminary Project Report

Oscar Kellner

Convolution neural networks (CNNs) are a type of neural network that excel in processing multidimensional data (e.g. images) and are a crucial component in computer vision applications. A CNN is composed of multiple layers, including convolutional layers, max pooling layers, and fully-connected layers. Within each layer, more complex and larger features of the input data are distinguished until at the end of the network where the network would make its best prediction as to what object it is observing [3]. This project intends to design a hardware implementation of part of a convolution neural network, specifically the convolution algorithm used within a convolutional layer (known here as a "convolution hardware accelerator").

The ultimate goal of this project, which is outside the scope of this class, is to be able to interface with an existing machine learning framework so that the accelerator will run if the parameters of the current convolution layer fit within the constraints of the hardware, or run a software function if it doesn't. Because this is a joint project with another class (EECE 4534: Microprocessor-Based Design) that emphasized interfacing with embedded systems, the Darknet framework was used as it was both open-source and written in C, making it convenient for developing on an embedded system. In the context of this project, the hardware had to be designed such that the organization of the input and output data was convenient for the Darknet framework to interpret. Sample data generated from the framework was also used for testing. For this class, the goal of this project is to synthesize the hardware, validate its functionality, optimize the design, and compare the performance to a similar software-based implementation.

The code used in High Level Synthesis had been overhauled and rewritten many times, finally settling for a more simpler design that will perform a single convolution for one channel and one kernel at a time and adding the results to a "partial sum" passed into the accelerator if it is processing multiple channels for a given filter. The typical "Conv2D" algorithm is used in this implementation, in that the value of a given pixel for an output map is determined by multiplying the input pixel and its surrounding neighbors within the kernel window by its weights stored within the window and accumulated together. In this implementation, an output value is streamed out as soon as it is calculated. The accelerator uses AXI-Stream and DMA interfaces for a greater efficiency in handling inputs and outputs. In order to conform to the nature of streaming in that data is only read once in a sequential order, a line buffer is implemented to retain input values that may appear again within the window [2].

Occasionally, there may be a convolution layer that has a kernel size of 1x1. When this occurs, there is no need for a line buffer as only one value is being multiplied and stored to the output map at a time in sequential order. Thus, a simple nested for loop is sufficient for a convolution involving a 1x1 kernel. It's worth noting that kernels with even dimensions (e.g. 2x2) are not compatible with this accelerator.

Debugging the code used within High Level Synthesis was challenging as it needed to conform to the way data was structured in the Darknet framework, as it was important to make integration back into the Darknet framework as simple as possible later in the project. The approach for debugging was modifying the original Darknet framework so that a different function (which would be used for the hardware design) would be called if the current convolution layer's parameters were sufficient for the accelerator. The new function would process the data in isolation, and return the values back into the framework. Keeping the function isolated from the rest of the framework (i.e. no libraries or functions shared from the source code of the framework) would allow the code to easily be transferred to Vivado HLS for synthesis.

After synthesizing the hardware **(with no HLS optimizations)**, a Jupyter Notebook was created to interact with the hardware through use of IPython and the PYNQ libraries. A basic test script was first written with help of the PYNQ documentation to ensure that data could be written to and read from the AXI-Stream and DMA interfaces [4]. Once the interfaces were confirmed to be working and that the outputs looked sensible, another script was created to emulate the first convolution layer of the Darknet framework's example configuration found within the tiny.cfg file. This was accomplished by creating 3 data files each containing a stream of float32 values for the layer's inputs, weights, and outputs, which were obtained by storing the values from a test run with an unmodified copy of the Darknet framework. The script would read the inputs and weights, run the accelerator, and compare the generated outputs with the outputs from the data file and count the number of incorrect values. After some careful debugging, the script was able to report no discrepancies between the generated output and the output from the data file. Generally, the hardware would complete the layer in around 12 seconds, with 1.75 seconds being used to load values into the input buffer. While there isn't a point of comparison with running Darknet on the PYNQ's PetaLinux, it's worth noting that Darknet will finish *inferencing* in less than one hundredth of the time taken on a Lenovo T580 laptop in Linux.

For a more visual demonstration of the accelerator, an additional test script was made to recreate the software implementation described within the submission of "Project Implementation 1," where a few kernels were used to convolute over a sample 128x128 RGB image depicting a flower. This script borrows resources from an educational article referenced at the end of this document [1]. This script also served to compare the performance between the convolution hardware accelerator and the software implementation of the Conv2D algorithm written in Python. The software implementation takes around 42.5 seconds to perform convolutions with all 6 filters, while the hardware can do the same in about 1.33 seconds. On average, the hardware can consistently perform more than 30 times faster than the software implementation. After each run, a grayscale image depicting the contents of each output map is saved within a folder. Interestingly, the hardware will produce "brighter" images than the software, however it's possible that this is a result of an unknown programming bug.

While it's difficult to reflect on how this project was approached given that the integration back to the Darknet framework isn't close to completion yet, I'm anticipating that it *will* be worth emphasizing compatibility with Darknet in the design of the accelerator, as needing to reorganize data in a different structure may cause too much overhead and negate the advantage of using a hardware accelerator rather than just designing the accelerator around the framework. That being said, I had wasted a substantial amount of time by hastily attempting to implement designs that tried to change as little as possible from the original Darknet source code. The problem with trying to do this is that Darknet's algorithm for forward convolution uses an algorithm known as "im2col," which is unfriendly for High Level Synthesis as it requires a large amount of area and does not easily allow for use of streaming (if at all).

Subsequent designs prior to the current implementation were slight improvements but were very limited. One design contained 4 input streams, 3 of which were reserved for input of individual channels. This allowed for the hardware to fully process a convolution for an entire filter at a time, but restricted to only processing convolution layers with 3 input channels. There were also occasionally some design errors when attempting to implement the AXI-Stream and DMA interfaces, such as forgetting to include a TLAST signal in order to indicate that the data had finished streaming. An additional AXI-Lite signal was added to determine the length of data being sent to the DMA for more control. All of these mistakes had consumed a lot of time that could have been spent exploring optimizations for the design, and many of these mistakes could have been avoided by closely studying the Darknet framework and spending more time planning out the hardware rather than hastily implementing half-working designs.

The next step of the project for this class is exploring options for optimizing the design, now that the overall design has been confirmed to be working properly. A 'design-space exploration' will be conducted in pursuit of finding the most optimal design for lowest area and lowest latency, with the results being presented in a Pareto optimization graph. The design with the lowest latency will be chosen for the final design, as in most cases it is preferred to have quicker inferencing in machine learning applications. If more time was allowed for the project, a potential next step would be attempting to redesign the algorithm so that sections of the input would be processed as chunks in parallel to increase performance. In other words, rather than convoluting through an entire 224x224 image at once, four 56x56 chunks would be processed in parallel and written to the same output buffer to potentially increase throughput.

To run the aforementioned test scripts, create a directory in the PYNQ's file system to hold the bitstream and hardware handoff files of the hardware design:

`'home/xilinx/pynq/overlays/fcl_accel/'`

Add the 'fcl_accel.bit' and 'fcl_accel.hwh' files within this subdirectory. You can run the test scripts (Jupyter Notebook files 'dma_flower.ipynb' and 'dma_darknet.ipynb') by executing each cell in the order that they appear. You may need to create the subdirectory 'flower_outputs/' in order to be able to save the output feature maps generated by 'dma_flower.ipynb'. Additionally, you will need the flower sample image 'flower128.jpg' and data files 'inputs.dat', 'weights.dat', and 'exp_output.dat' within the same directory as the Jupyter Notebooks for everything to work properly.

In addition to these test scripts, the code used to synthesize the current design using High Level Synthesis has also been included (named 'fcl.ccp' and 'fcl.h'). All necessary files are included in a ZIP archive alongside this report.

References:
[1]https://towardsdatascience.com/convolutional-layer-hacking-with-python-and-numpy-e5f64812ca0c
[2]https://basile.be/2019/03/18/a-tutorial-on-non-separable-2d-convolutions-in-vivado-hls/
[3]https://www.ibm.com/cloud/learn/convolutional-neural-networks
[4]https://pp4fpgas.readthedocs.io/en/latest/axidma2.html