Convolutional Neural Networks Implementation on FPGA (Verilog)

Spring 2020 | Logic-Design-2 | CMPN111
Three team members

Ahmed Mohamed Abdelazim
1170100

Mahmoud Ezzat Mahmoud
1170532

Nour Ayman Gaafar Zohry
1170408

Abstract— **In deep learning, a convolutional neural network, or CNN, is a type of neural network used to identify and analyze images. CNNs have proven to have very high accuracy in a wide range of activities as well as overall good performance. By using energy-efficient and effective FPGAs, CNNs' performance can be even further optimized. In this paper, we present the design and implementation of a convolutional neural network on FPGAs using Verilog. We discuss the key points in our projects and compare our project to previously created projects to see the advantages and disadvantages of similar implementations. By dividing the project into five parts, we could more easily group similar tasks together and organize the project to achieve the best outcome. Finally, we expect this CNN to adequately conduct its purpose in a variety of tasks and problems.**

Keywords—Field Programmable Gate Arrays, Convolutional Neural Network, Verilog

I. Introduction

Artificial neural networks are capable of expertly handling numerous tasks such as image recognition, machine translation, and video analysis. In fact, artificial neural networks are able to even surpass and outperform humans and other existing machine learning methods. Today, convolutional neural networks, or CNNs, are the involved in many neural network architectures due to the need for convolution, which allows for the extraction of features from input images.

CNNs, neural networks that include convolutional layers, are one of the most successful deep learning models, having shown excellent performance in several computer vision and machine learning problems. CNNs have numerous applications such as face recognition, text-to-speech synthesis, handwritten character recognition, image description generation, among many other uses as needed in the field.

One common usage of CNNs is image recognition; the CNN takes an input image and extracts the features of the image such as color or texture. After repeated trials, the CNN can identify the object in the image and classify it. For example, it can identify whether the animal in an image is a cat or a dog based on the fur's texture and the facial features such as the eyes, nose, and ears. Of course, CNNs aren't limited to identifying animals, but allow for the general recognition and identification of images for image categorization and future applications' use.

In order to improve the performance of our designed CNN, we used an FPGA-based design. The benefits implementing FPGAs into the CNN's design include higher computation speed, wide reconfigurability, and overall higher performance. Additionally, FPGAs are energy efficient and adaptable.

II. Selecting FPGA Implementations to Discuss

All of the following reports were published in IEEE Conferences and can be found in the references list at the end of this paper.

A. Floating Point or Fixed Point Convolution on FPGA

Caffeinated FPGAs: FPGA framework For Convolutional Neural Networks" by R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor and S. Areibi [1]

B. Floating Point Tanh Operation

" A Configurable FPGA Implementation of the Tanh Function Using DCT Interpolation" by A. M. Abdelsalam, J. M. P. Langlois and F. Cheriet [2]

C. Floating Point Addition/Multiplication/Division, ANN, SoftMax and Averagepool on FPGA

" VHDL generator for a high performance convolutional neural network FPGA-based accelerator" by M. K. Hamdan and D. T. Rover [3]

D. Whole Convolutional Neural Network Implementation on FPGA

" FPGA-Based Convolutional Neural Network Architecture with Reduced Parameter Requirements" by M. Hailesellasie, S. R. Hasan, F. Khalid, F. A. Wad and M. Shafique [4]

III. Literature Reviews

A. Floating Point or Fixed Point Convolution on FPGA

In my selected paper for the topic of convolution, "Caffeinated FPGAs: FPGA Framework For Convolutional Neural Networks," the writers present a "a modified version of the popular CNN framework Caffe, with FPGA support" with the intention of implementing FPGAs into CNNs in order to improve the functionality of CNNs and advance the field of machine learning in general.

The proposed design utilizes an FPGA-based Winograd convolution engine, which is powered by a convolution method named Winograd convolution in which the Winograd minimal filtering algorithm is deployed. This convolution method allows the CNN to spend less time in calculating floating point operations by decreasing the number of required floating point operations.

The common Winograd convolution engine, the convolution method without the addition of FPGAs, is represented by the function F(m×m, r×r) in which m×m refers to the output file's tile size and r×r refers to the filter size. By applying many calculations on the variables m and r, much more data can be obtained such as variables for filter transformation and input transformation. The more important variable to calculate is the final output of the Winograd algorithm by using the pre-computed transformed filter and the run-time transformed input data as shown in the following equation as documented and explained in the paper:

$$Y = F(m \times m, r \times r) = A^T[U \odot V]A$$

"Where:
Y is an m × m output tile;
⊙ is an element wise multiplication;
A is an (m + r − 1) × m transform matrix, defined by the Winograd algorithm."

B. Floating Point Tanh Operation

Activation functions are very important components in artificial neural networks in general as they control the outputs of nodes given an input or set of inputs. One of the main features of using activation functions is their nonlinearity; nonlinear activation functions allow networks to complete complicated tasks using only a small number of nodes and hence less resources.

The hyperbolic tangent activation function, also known as the tanh function, is considered to be one of the most commonly used nonlinear activation functions due to its high accuracy which leads to greater performance.

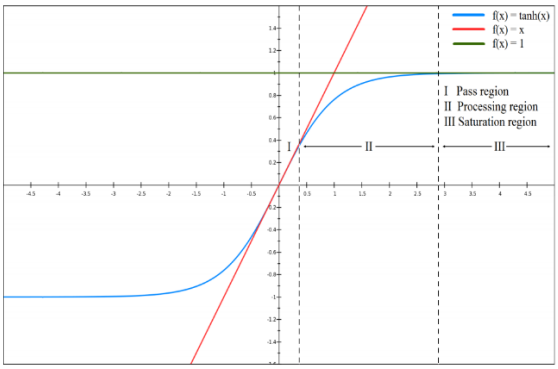

Figure 1: Hyperbolic Tangent Activation Function

As shown in Figure 1, the hyperbolic tangent activation function can be divided into three regions; Pass Region, Processing Region, and Saturation Region. In the Pass Region, the function behaves like an identity function, having the same values as the function f(x) = x. In the Saturation Region, the function's value is approximately 1. The function has a curve in the Processing Region that connects the function's values from Pass Region to Saturation Region.

The paper proposes a new high- accuracy approximation using the Discrete Cosine Transform Interpolation Filter, also known as DCTIF on FPGAs and by additionally using the hyperbolic tangent activation function. By using the hyperbolic tangent activation function, the DCTIF-utilizing Deep Neural Network (DNN) can have better performance through high accuracy and lower cost, so it is to the authors' benefits to use the hyperbolic tangent activation function in the proposed project.

The results of the upgraded DNN have been tested and analyzed and it has been confirmed that using the hyperbolic tangent activation function on FPGA using DCTIF yields greater results through higher accuracy for similar amounts of computational resources.

C. Floating Point Addition/Multiplication/Division, ANN, SoftMax and Averagepool on FPGA

The paper acknowledges CNN as an effective algorithm which can be used in many applications such as handwriting recognition and image classification, but since a CNN normally takes a very long development round on FPGAs, the paper proposes a tool to automatically write fast and modular VHDL code depending on the requested CNN model. The tool is titled VHDL Generator for A High Performance Convolutional Neural Network FPGA-Based Accelerator. This is demonstrated using two different CNN models; LeNet and AlexNet.

Among the many layers of the CNNs such as the convolutional layers and fully-connected layers are the pooling layers. This CNN uses spatial pooling, described as "a form of nonlinear subsampling that is utilized to reduce the feature dimensions as we go deeper in the network," or simply a way to reduce the size of the input so the size can be reduced as it continues the sequence of operations from layer to layer.

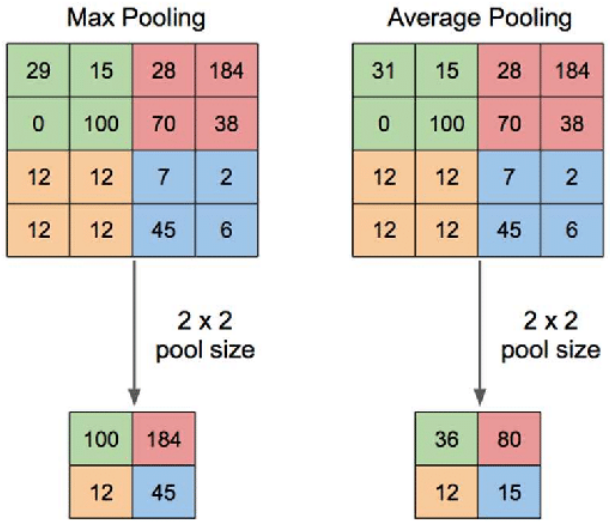

Figure 2: Max Pooling and Average Pooling

The two main methods of pooling are max pooling and average pooling. In max pooling, the maximum neuron values are passed to the next layer. Alternatively, in average pooling, the value passed to the next layer is the average of the neurons. The differences can be seen in Figure 2.

In the fully connected layer, the SoftMax classifier takes the output of the nonlinearity in the final fully connected layer. The SoftMax classifier performs its task and converts the output neurons to a probability in the range (0,1) so they can be classified into output classes, hence SoftMax also being called an Output Classifier.

D. *Whole Convolutional Neural Network Implementation on FPGA*

In this paper, the writers propose an approach for improving the speed of a convolutional neural network by using hardware instead of software and using fixed-point calculations instead of floating-point calculations. The convolutional neural network's speed can also be increased by reducing the network size and decreasing the size of saved weights alongside other small changes to the structure of the network.

To implement this approach, the DE0-Nano development board is used for several reasons. These reasons include that the board uses an Intel FPGA which is considered cost-effective, and that the board has 32 MB of RAM available which can be used to store the weights. Additionally, connecting additional hardware such as cameras is made easier using the board.

The CNN's architecture is adjusted to suit the application in which fixed-point numbers are used instead of floating-point numbers and hardware is used instead of software. These adjustments include:

1. Minimizing the number of fully connected layers;
2. Reducing the number of convolution layers;
3. Using simple activation function such as RELU instead of tanh or others;
4. Minimizing the number of heterogeneous layers.

By applying these changes, the number of stored weights is minimized. These changes are only done to the point of not degrading the output or operations. For example, reducing the number of convolution layers is done as much as possible without degrading the performance.

The project also proposes an adjustment to the architecture named the Low Weights Digit Detector or LWDD. The large fully connected layers and bias terms are removed and the "GlobalAvgPooling" layer is replaced with "GlobalMaxPooling." These adjustments assure that the efficiency of the layers is unchanged and that the network performance also stays the same.

This new proposed architecture design is successfully implemented on FPGA and the results demonstrate low hardware requirements. Input images are processed within milliseconds where the output shows the original image alongside the output image. It takes roughly 230,000 clock cycles to classify one image using this design.

IV. Discuss About the Selected Convolutional Neural Network(s) and it's Architecture

Our selected CNN is essentially a sequence of operations applied on an input matrix which represents the input image. This sequence of operations is performed using a number of processes including convolution, average pooling, and hyperbolic tangent and SoftMax activation.

One of the major operations performed in the CNN is convolution. The purpose of convolution is to extract features from the input image and classify said features, for example classifying what counts as a horizontal or vertical line. While lines are perhaps the simplest to identify, it's more difficult to classify what qualifies as a mouth or nose, or building or garden. It isn't limited to just structures or creatures but also corners and curves and edges, anything that may appear on an image.

This CNN uses the classic LeNet-5 architecture which consists of 6 layers excluding the input and the output layer. The first 5 layers of the 6 are followed by the hyperbolic tangent activation function whereas the 6th layer is followed by the SoftMax activation layer. After convolution takes place, the LeNet takes the output and applies average pooling to reduce the dimension of the resulted image and use the average of the output's features.

A quick summary of the layers has the following layers, also depicted in Figure 3:

1. Convolution: Input 32x32x1 grayscale image passes through the first convolutional layer and outputs as 28x28x6;
2. Average pooling: The result is reduced to 14x14x6;
3. Convolution: Using 16 feature maps of size 5x5, only 10 feature maps are connected to 6 of the previous layer's feature maps;
4. Average pooling: The result is reduced to 5x5x16;
5. Convolution: Using 120 feature maps of size 1x1, all 120 units are connected to all 5x5x16 nodes;
6. Fully connected layer with 84 units

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Figure 3: LeNet Layers

V. Discussion about the Developed Verilog Modules

The developed Verilog code includes several modules, each module having a testbench file with which to test the output of the written Verilog module. Unlike many existing designs for CNNs, the Verilog code for this CNN uses fixed point calculations instead of floating point calculations, which allows for faster calculations that are more simple to handle when writing the Verilog code.

The convolutional basic unit, which takes an image and defines its features by through element-wise multiplication of the filter matrix, takes an image and a filter as an input. Through repeated addition of the element-wise multiplication results, the convolution outputs the convolution result image.

For the tanh activation function, the function takes the matrix as an input and performs Taylor expansion approximation as shown in the following equation. Compared to the Coordinate Rotation Digital Computer algorithm, or CORDIC algorithm, tanh is faster in this application for the CNN.

$$\tanh x = x - \frac{1}{3}x^3 + \frac{2}{15}x^5 - \frac{17}{315}x^7 + \cdots$$

The SoftMax activation function functions similar to the tanh activation function in which it takes a matrix as an input and uses a similar equation to the tanh activation function which is the Taylor expansion shown in the following equation.

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} \cdots$$

Pooling is done in the average pooling layer. It takes a vector as the input and outputs the reduced version of the input image as an arithmetic average. As we are using fixed point calculations in this design, writing the code for it was easier than using floating point calculations as modules such as fp_mul and fp_div are simpler and hence averaging is easier and faster.

VI. Comparison Between The Selected Attempts And Our Developed CNN RTL Implementation

A. Floating Point or Fixed Point Convolution on FPGA

Unlike the implementation in the selected paper, our developed CNN RTL uses LeNet-5 convolution instead of the FPGA-based Winograd convolution engine mentioned. Furthermore, the Winograd method was mainly proposed to spend less time and energy on the calculations of floating point operations, whereas in our implementation for the CNN, we used simply fixed point numbers instead of floating point calculations due to external constraints.

B. Floating Point Tanh Operation

In our CNN project, the activation function is handled by a hyperbolic tangent activation function and uses the Taylor expansion approximation. In the paper, the Discrete Cosine Transform Interpolation Filter or DCTIF to achieve better performance in this specific application for the DNN.

C. Floating Point Addition/Multiplication/Division, ANN, SoftMax and Averagepool on FPGA

(Please note that although the requirement mentions floating point operations, our CNN project was implemented using fixed point operations since we were allowed to do so in our case due to the small number of members working on the project.)

In our CNN project, pooling is done by the average pooling layer, rather than maximum pooling, which accepts a vector of length 4x1 and outputs the arithmetic average of the input vector values as a single output. In the case of the selected paper, a max pooling layer was used in the AlexNet CNN model instead.

Both our CNN and the paper's CNN use SoftMax in some way, but there are still some differences. In our CNN, the SoftMax can result in outputs from the range 0 to 9 whereas the proposed CNN results in outputs in the range of 0 to 1. The features and usage of the two SoftMax implementations are still reasonably the same.

D. Whole Convolutional Neural Network Implementation on FPGA

There are some differences, ranging from small to fundamental, between our implemented CNN and the proposed CNN from the paper. The general features for our implemented CNN and the proposed CNN are roughly the same, but the major difference is in the platform. In our CNN design, we design a common LeNet model CNN. However, in the proposed CNN design, many software and hardware modifications are made in order to implement neural networks for mobile devices.

Some of these differences are in hardware as well. Our CNN project uses the PYNQ development board whereas the proposed CNN uses the DE0-Nano development board which makes the available RAM in both projects largely different.

There are still some differences between both projects. Both projects use fixed-point arithmetic instead of floating-point arithmetic which increases the speed of operations due to the fewer calculations and both projects also rely on hardware. Similarly, both projects use FPGAs to implement CNNs.

VII. Conclusion About What Is Unique In Our Work

In this CNN project, we use a simple yet effective convolution method in which the parallel convolutional layer is divided into two subtasks: parallel convolution with single filter and parallel convolution with many filters. The latter is an extension to the design in the former, where the parallel convolution with single filter accepts the whole image and produces the convolution process result.

The used activation function, the hyperbolic tangent activation function, uses Taylor expansion approximation in calculations. It is nonlinear in nature so it's possible to stack layers. The hyperbolic tangent activation function is used in almost every layer throughout the CNN which makes it a very important and effective component in the CNN design.

The SoftMax activation function takes exactly 10 values. The result represents the final classification which is represented through values from 0 to 9. Some other implementations have different ranges, but the range from 0 to 9 is suitable for this application of CNN in our case. Somewhat similar to the hyperbolic tangent activation function, the SoftMax uses Taylor expansion.

The average pooling layer is done by accepting a vector of length 4x1 and outputting a single output value. While others have used average pooling using floating point numbers or using different kinds of pooling such as maximum pooling, in this project we used fixed point numbers with average pooling.

Overall, by connecting all the layers together, we created a completely LeNet Convolutional Neural Network. The CNN takes an image as an input and goes through the many described operations such as convolution, activation functions, and pooling to finally output the resulting image. Using FPGAs makes the CNN operate quickly and faster than some existing neural network implementations.

References

[1] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor and S. Areibi, "Caffeinated FPGAs: FPGA framework For Convolutional Neural Networks," 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, 2016, pp. 265-268, doi: 10.1109/FPT.2016.7929549.

[2] A. M. Abdelsalam, J. M. P. Langlois and F. Cheriet, "A Configurable FPGA Implementation of the Tanh Function Using DCT Interpolation," 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, 2017, pp. 168-171, doi: 10.1109/FCCM.2017.12.

[3] M. K. Hamdan and D. T. Rover, "VHDL generator for a high performance convolutional neural network FPGA-based accelerator," 2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, 2017, pp. 1-6, doi: 10.1109/RECONFIG.2017.8279827.

[4] M. Hailesellasie, S. R. Hasan, F. Khalid, F. A. Wad and M. Shafique, "FPGA-Based Convolutional Neural Network Architecture with Reduced Parameter Requirements," 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, 2018, pp. 1-5, doi: 10.1109/ISCAS.2018.8351283.

[5] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing and X. Liang, "DRQ: Dynamic Region-based Quantization for," ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Shanghai Jiao Tong University, China, 2020.

[6] J. Wu, "A Survey of FPGA-based Accelerators for Convolutional Neural Networks," National Key Lab for Novel Software Technology, Nanjing University, China, 2017.

[7] Zhang, Chen & Li, Peng & Sun, Guangyu & Guan, Yijin & Xiao, Bingjun & Cong, Jason. (2015). Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. 161-170. 10.1145/2684746.2689060.

[8] M. Rizwan, "LeNet-5 – A Classic CNN Architecture," 30 9 2018. [Online]. Available: https://engmrk.com/lenet-5-a-classic-cnn-architecture/. [Accessed 6 6 2020].