Facial Recognition using hardware accelerated CNN

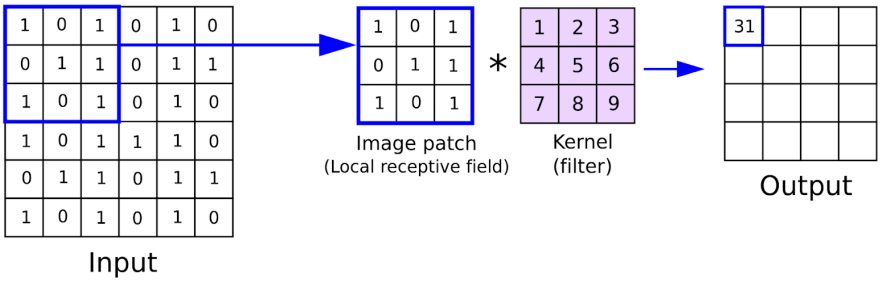
Ryan Kennedy (rdk2132), Felix Hanau (fjh2116), Daniel Cooke (dwc2122), Richard Mouradian (rom2110), Liam Bishop (lb3306)

Overview:

The idea for our project is to make a Convolutional Neural Network accelerator for a Computer Vision Model which will perform facial recognition. Computer Vision models which rely on CNN's are very taxing on a processor because of the constant heavy computations needed to process images. To alleviate this we will get the FPGA to perform very fast and parallelized computations on pixel matrices. We will use the software as a controller for the CNN which will take video input from a camera (peripheral) and feed it to the CNN algorithm on the hardware side. Upon completion of the final layer of computation we will get the output indicating the decision the model made. Finally we can set up a display to say whose face it recognized to the user.

CNN Model:

Convolutional Neural Networks (CNN) are a form of Artificial Neural Network that we will be using for image recognition. A CNN is composed of many connected neurons with weights and biases. Each neuron receives input data and performs operations on it using the weights and biases then outputs the transformed data to the next layer. The CNN model we are building is composed of four types of layers: convolution, activation, pooling, and fully connected. These functions are combined to form single layers, then the layers are stacked to form the CNN.



[2]

The convolution function is the core function of the CNN and is used to extract features from the input images. The convolution works by taking a small filter or kernel (usually 3x3 or 5x5) and sliding over the input. At each pixel, the filter values are multiplied by the corresponding pixel in the neighborhood and summed to create a feature map. Generic kernels exist such as the Sobel filter for detecting vertical and horizontal edges, but our CNN will use unique kernels trained to recognize features specific to our dataset. For pixels on the edge of the image where the filter would hang over the image can be zero-padded by adding zeros to the edge of the image. The other option would be to ignore the edge pixels and shrink the image after convolution. Our model makes use of convolution in two layers and uses 5x5 kernels along with no zero padding to decrease memory usage.

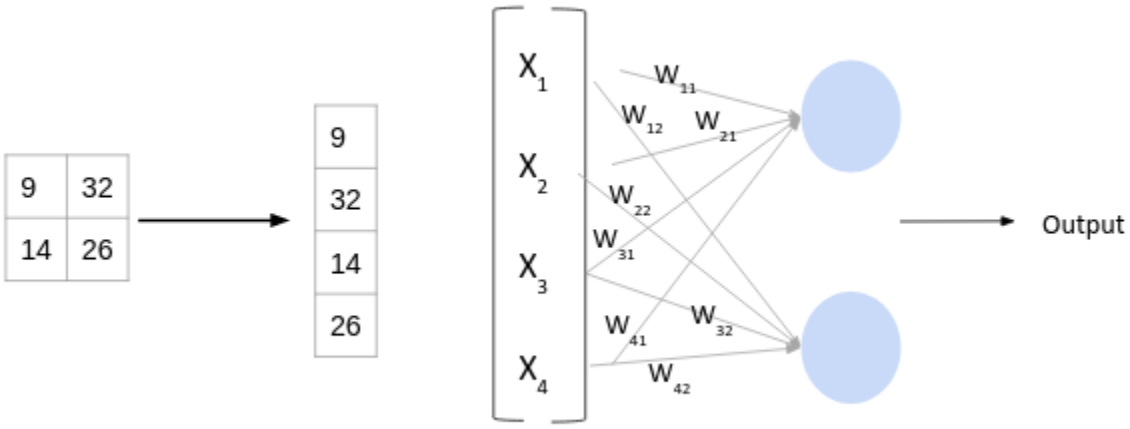
The activation layer introduces non-linearity into the CNN model and is used to decide if the neuron will "fire" or not. This effect is achieved through the use of an activation function. A popular activation function is ReLU which is simply f(x) = max(0,x), it eliminates negative numbers and turns them to zero. Another activation function is sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Because this function uses the exponential function, for the hardware implementation we will use a segmented fitting optimization as shown in [3].

Interval	Fitting function
[0, 2 048]	f(x)=1004x/4096+2050
[2 049,10 035]	f(x)=-184x^2/4096/4096+1167x/4096+2012
[10 036, 16 384]	f(x)=-71x^2/4096/4096+615x/4096+2695
[13 385, 22 938]	f(x)=-17x^2/4096/4096+198x/4096+3500
[20 939, 30 720]	f(x)=6x/4096+4050
[30 721, 32 767]	f(x)=4096

The fully connected layer takes as input the flattened matrix of the previous layer's output (ie. 12x4x4 gets flattened to 192x1) and fed as input. The inner workings of this layer take each of these 192 neurons and connect them to each and every neuron in the next layer. The size of this next layer in the fully connected layer can be changed to meet the desired output possibilities as the user wishes. For a next layer of size 10 we have 1920 connections each with a corresponding weight as well as a bias for each neuron in the next layer (10 in size, 1 for each neuron). To illustrate this refer to this image below [4].



The pooling layer averages the input values via a 2x2 kernel and outputs the result to the next layer. By averaging the input, this layer downsamples the input image to decrease its resolution.

We use bias in the convolution layer, but not in the fully connected layer. As the activation function, we use ReLU in the convolution layer and sigmoid in the fully connected layer.

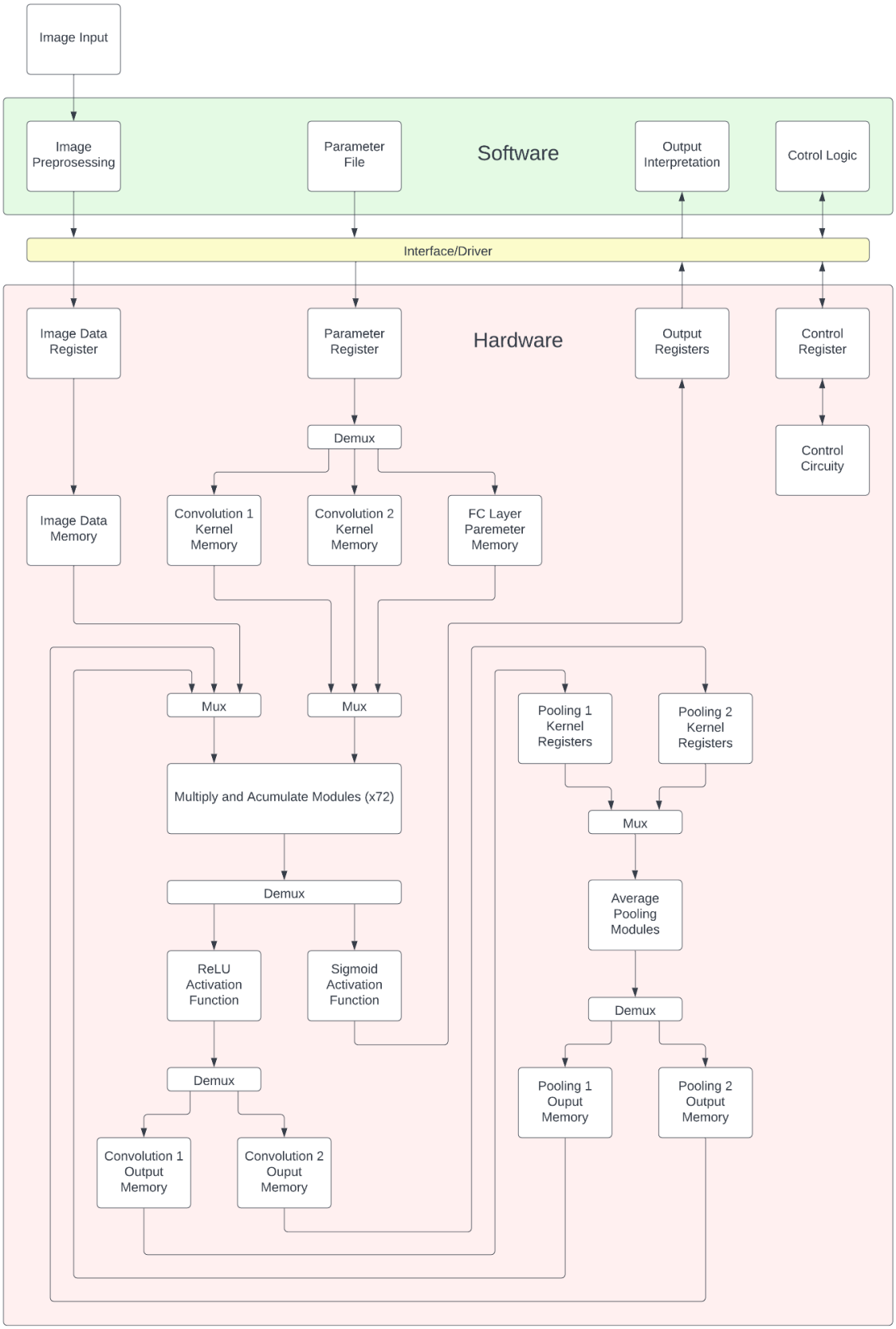
The input to the CNN is a grayscale 28x28 image.

Layer Number	Type	Kernel Size	Output Size	Parameters
1	Convolution	6 5x5 (150)	6x24x24 (3456)	156
2	Avg Pooling	6 2x2 (24)	6x12x12 (864)	None
3	Convolution	6 groups of 12 5x5 (1800)	72x8x8 (4608)	10872
4	Avg Pooling	12 2x2 (48)	72x4x4 (1152)	None
5	Fully Connected	None	50	57600

Software/Hardware Interface:

Even though the accelerator is quite complex, the software/hardware interface is surprisingly simple with 2 input registers for image data and parameters and 50 output registers for the resulting classification vector.The image data will be preprocessed with downsampling and conversion to 16-bit fixed point prior to being sent to the FPGA. The input image data register will be a 16-bit register and the input parameter register will be a 16-bit register. The data will be sent to these registers in serial and will be loaded into the FPGA's memory. After the accelerator completes, the 50 output values that are stored in 16-bit registers on the FPGA can be sent back to the software. Controlling the accelerator will be accomplished by an 8-bit control register. The first bit when set to '1' will indicate to the accelerator that image data should be loaded into memory. The second bit if set to '1' will indicate to the accelerator that parameter data should be loaded into memory. The third bit when set to '1' will start the accelerator. Finally, the fourth bit will indicate to the software when the accelerator is finished. The remaining 4 bits will be reserved for the inclusion of more control signals if needed.

Block Diagram:



Resources:

Layer	Data (Bits)	Weights + Bias (Bits)	Memory Needed(Bits)
Input	28x28x16	0	12544
1	6x24x24x16	6x5x5x16 + 6x16	57792
2	6x12x12x16	0	13824
3	72x8x8x16	72x6x5x5x16 + 72x16	247680
4	72x4x4x16	0	18432
5	50x16	57600x16 + 50x16	923200
		Total:	1273472

As it can be seen from the table above our estimation for memory utilization is 1273472 bits or about 1300 Kb which is capable of fitting within the 4450 kb of embedded memory of the FPGA. In addition to memory another potential bottleneck is the utilization of the DSP blocks. For this accelerator 72 multiply and accumulate modules will be used to parallelize the computations, especially in the convolution layers. Each multiply and accumulate module will utilize a DSP block in a 16x16 multiplier adder mode. This will give us a remainder of 15 DSP blocks that can be used for other functions needed such as proper memory addressing, the best fit curves for the sigmoid function, and other mathematical functions.

Citations:

1. D. Shan, G. Cong and W. Lu, "A CNN Accelerator on FPGA with a Flexible Structure," 2020 5th International Conference on Computational Intelligence and Applications (ICCIA), 2020, pp. 211-216, doi: 10.1109/ICCIA49625.2020.00047.
2. V. Lendave, “What is a convolutional layer?,” *Analytics India Magazine*, 18-Jun-2021. [Online]. Available: https://analyticsindiamag.com/what-is-a-convolutional-layer/. [Accessed: 29-Mar-2022].
3. Yuxi Zhang, Hanying Yang and Yaotian Zhang, "Implementation of sigmoid function based on FPGA", Computer Engineering and Application, vol. 52, no. S1, pp. 501-504, 2016.
4. *Introduction to neural network: Convolutional Neural Network*. Analytics Vidhya. (2020, May 8). Retrieved March 29, 2022, from https://www.analyticsvidhya.com/blog/2020/02/mathematics-behind-convolutional-neural-network/