Pattern Recognition using Convolutional Neural Networks in Python

Zion Eric O. Chan*, Mac Excel S. Fallar†, Patrick Julian M. Ramos‡, and Miguel Karlo D. Sese§
Electronics and Communications Department
De La Salle University - Manila
Email: *zion_chan@dlsu.edu.ph, †mac_fallar@dlsu.edu.ph, ‡patrick_ramos@dlsu.edu.ph, §miguel_sese@dlsu.edu.ph

***Abstract*—This paper discusses the implementation of a convolutional neural network for pattern recognition through the use of Python. It aims to aid the researchers to understand the concepts of convolutional neural network through building it from scratch. The neural network can learn four types of patterns with a 4x4 resolution.**

I. Introduction

The design considerations primarily focuses on analyzing the convolutional neural network (CNN) in Python in a way that the core concepts are extracted in order to understand the architecture from how it was built from scratch. This is a necessary step to understand how the CNN works without the usage of libraries that skips the core concepts on how a CNN is actually made from scratch. By obtaining this knowledge, it is now possible to create the FPGA equivalent of the CNN implementation in Python in a more convenient way by translating the Python code into VHDL to create the hardware model.

Convolutional neural network (CNN) is under the classification of a deep neural network. Convolutional neural networks uses the concept of neural networks but assumes that the inputs fed are images. Properties of these images are recorded into filters that capture the features of the images after the whole training process. A normal neural network would have a basic construction composed of simply an input layer, a hidden layer, and an immediate output layer, while on the other hand, a deep neural network is composed of underlying multiple hidden layers for providing a better output accuracy.

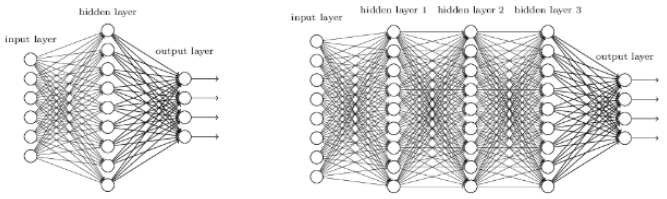

Fig. 1. Neural Network vs Convolutional Neural Network

II. Architecture

A small-scaled pattern recognizer architecture is implemented to further study the behavior of convolutional neural networks. The architecture simply involves an input layer, a convolution layer, a fully connected layer, and an output layer as shown on figure 2.

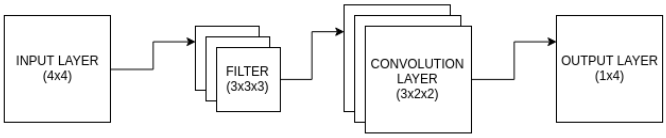

Fig. 2. Pattern Recognition Scale Architecture

III. Flowchart

A. Recognition

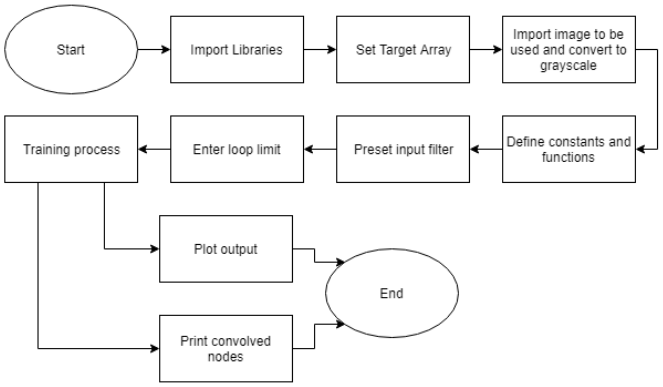

Fig. 3. Recognition Flowchart

As shown on figure 3, the recognition flow starts my simply importing libraries for plotting images, and other functions needed to work on. The setting of target array is for what trigger array is to be used, then after that is the importing of image to be converted into grayscale. The constants are to be defined right after that step, which includes the learning rate to be used, the iterations to be performed, and other sizes that will matter on the process. Next is the definition of the functions to be used such as, softmax, rectified linear unit, sigmoid, error calculation, activation, CSV, and forward pass. The training process begins right after that, then finally plotting the output and print the convolved nodes.

B. Training

As stated on figure 3, the steps for the training part is mostly repetitive, but first, specify the input training image. After that, the next step is to generate random input filter and input fully connected weights. Then output random filter weight and fully connected values to CSV. After that is the main loop for the training which contains the forward pass, the convolution part, the sigmoid activation, the flattening of sigmoid, the implementation of the fully connected layer, the softmax activation, the error calculation, the backpropagation which is to be degraded in the latter, then lastly is the generating of maximum/minimum/average values into CSV. After loop is done, proceed to output.

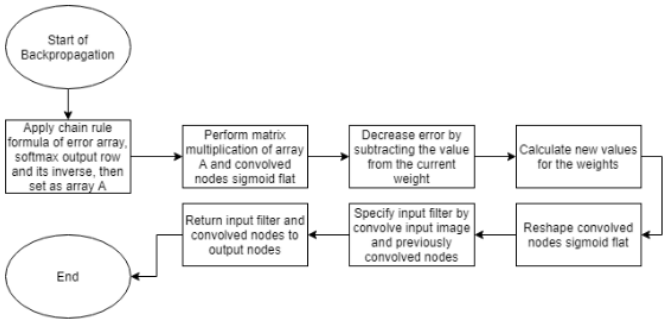

Fig. 4. Backpropagation Flowchart

According to Mazur [1], the backpropagation is the important feature in convolutional neural network because this is where the accuracy relies on. As stated on figure 4, by converting the formula in the chain rule to a python code, and perform the loop on error decrease the calculation of new values, you will have a good set of weights for your output. After applying the chain rule, the matrix multiplication will take part then together with that is the use of the learning rate. The next part is the calculation of new values for the weights and the reshaping of the convolved nodes that has been sigmoided and flattened. Then after that is specifying the input filter by convolving the input image and the previously convolved nodes. Lastly, is returning the input filter and the convolved nodes to output nodes.

IV. Python Dependencies

Python version 3.6.4 will be used to implement the convolutional neural network. The libraries Numpy, Pillow, Matplotlib, and Scipy will be used to manipulate the data in each operations. These libraries have specific functions that are helpful in understanding the nature of convolutional neural networks in a scientific approach.

A. NumPy

According to the Scipy Organization [2], Numpy is a package that is focused on scientific computing through the use of manipulating multidimensional array objects. It is a part of Scipy. Numpy focuses more on operations such as mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transformations, basic linear algebra, basic statistical operations, random simulation and other functions regarding array manipulation. This library will be heavily used for array manipulation of each layers and weights in the convolutional neural network.

B. Pillow

Python Imaging Library, or also known as Pillow, gives Python image processing capabilities [3]. The library supports a plethora of file formats which also have an efficient internal representation to be able to process images in a powerful manner. The package can create image archives, do batch processing, and convert file formats seamlessly. It currently supports PhotoImage and BitmapImage interfaces that can be used with PythonWin and other Windows-based toolkits. In this research, this library is used to extract the pixel data from the input images and put into arrays to be processed and fed into the neural network.

C. Matplotlib

Matplotlib is a library that focuses on plotting different types of graphs, images, and data features [4]. The Pyplot feature of the Matplotlib library will be focus of the design. According to Matplotlibs documentation [5], Pyplot is a function that works like the plot functions of MATLAB. The Matplotlib library will be used to observe the step-by-step manipulation of the nodes of each layer, the filter weights, and the fully connected layer weights.

D. SciPy

Scipy is a library that implements a collection of mathematical algorithms and functions. It provides higher-level commands for manipulating and visualizing data [6]. The scipy library will be used for the mathematical functions such as convolving in the convolutional neural network.

E. Pandas

According to its official documentation [7], Pandas is a package that provides useful data structures for file manipulation, file representation, data manipulation, and data representation. In the system implementation, pandas is used to manipulate numpy arrays into CSV files for storing each iteration of the training process of the convolutional neural network. The minimum value, maximum value, average value, and output values of the filter weights and fully connected layer weights are stored into the CSV files for further reference for the hardware modeling of the FPGA.

V. Weights

A. Initial Weights

The number of initial weights are dependent on the number of filters, its matrix size, and the corresponding stride and padding of the convolution process through the input image data. The value of the weights are initialized at random between 0 and 1. Despite having random values initially, the backpropagation of the neural network would create the optimal filters after training.

B. Final Weights

The number of final weights is the same as the number of the weights initialized. The weights of the neural network is updated and determined by the backpropagation process of the neural network through each iteration. The values of the final weights would enable the neural network to be able to recognize the distinct features of a pattern. Figure 5 shows a sample final filter weight plotted PyPlot.

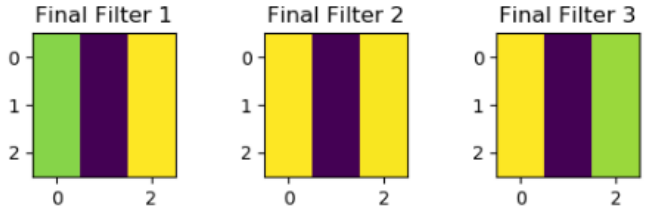

Fig. 5. Sample Final Filter Weights in PyPlot

VI. Input Layer

A 4x4 input image of black and white pixels are accepted into the input layer. Figure 6 is a sample 4x4 input image represented in Pyplot.

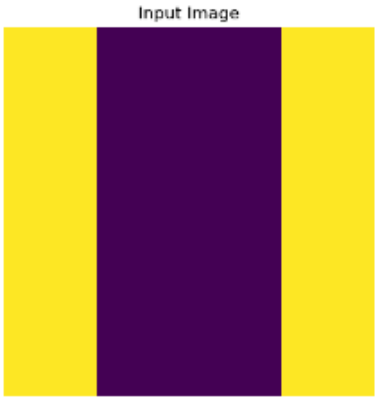

Fig. 6. Sample Input Image

The yellow areas of the image have a value of 255 while the purple areas have a value of 0. These values are later normalized into values that will range from 0 to 1. Normalizing these values will help reduce that numerical value of the filter weights, as well as the corresponding nodes.

VII. Convolution Layer

After getting the pixel data from the input image, the gathered data is convolved onto each filter to find similar features of the input image and the specified filter.

This is done by running a specific number of filters through the image depending on the specified stride and padding to produce an output corresponding to the similarities of the input image and the filter. The convolution operation will not use any padding and will use a stride of 1. Figure 7 shows a clearer understanding of the convolution operation in a 3x3 input with a 2x2 filter.

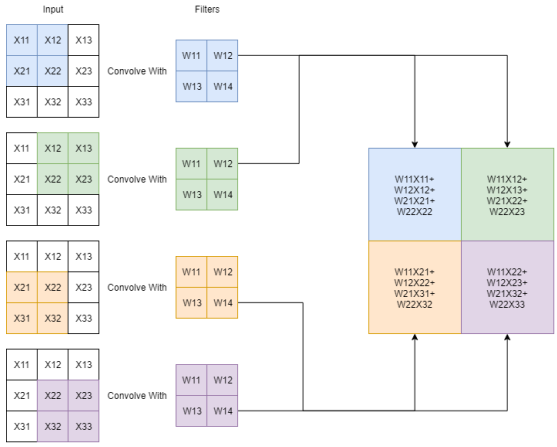

Fig. 7. Forward Pass Convolution Representation

From the input image pixel data and the filter data, the dot product is calculated between them using the formula in equation 3. which is also known as the convolution. This process will enable the neural network to identify the similar features of the input image and the filter convolved. The output is then fed into the max pooling layer to minimize the number of data for training.

$$\underset{a}{\rightarrow} * \underset{b}{\rightarrow} = a_1 b_1 + a_2 b_2 + a_3 b_3 \tag{1}$$

From a source by Agrawal [8], for the backpropagation of the convolution layer, the same procedure is done. However, the output of this procedure would result in getting the updated filters for the neural network. This is done by convolving the input image and the updated nodes in the convolution layer from the backpropagation process. Figure 8 shows this process in a clearer manner.

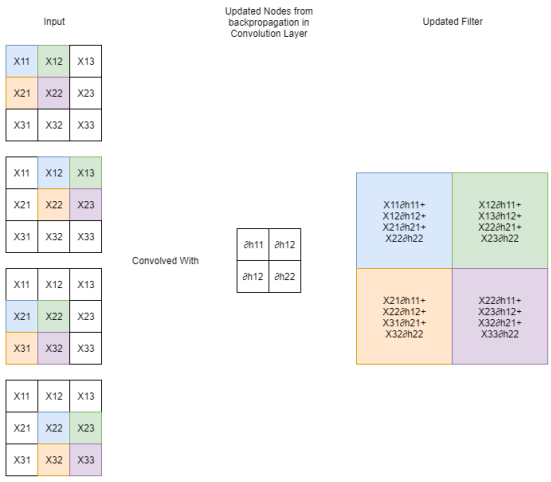

Fig. 8. Backpropagation Convolution Representation

VIII. Activation

The activation function that was used in this research is the sigmoid function. The sigmoid function limits the values from 0 to 1 depending of the degree of its original value. According to Han and Moraga [9], the sigmoid function removes the negative values and makes it close to zero while positive numbers are close to 1, however this function has a vanishing gradient problem and optimization problem. Another source by Walia [10], sigmoids saturates and kills gradients that may reduce accuracy of edge detected features. The output of the sigmoid function is not zero centered, and it makes gradient updates too far in different locations which makes optimization suffer.

A sigmoid function is a mathematical function having a sigmoid curve [11]. A sigmoid function is a real function that is differentiable and is bounded for all real input values and has a non negative derivative at each point. The sigmoid function refers to the logistic function in figure 9 and defined by formula 2.

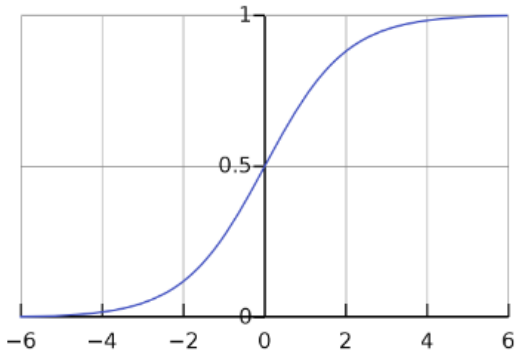

Fig. 9. Sigmoid Function

$$S(x) = \frac{1}{1+e^x} = \frac{e^x}{e^x+1} \tag{2}$$

IX. Fully Connected Layer

In a fully connected layer, each and every node or neuron in one layer is connected to every node or neuron in the other layer. The connection is the same as the common perceptron neural network that the architectures every neuron in a layer is connected to each neuron in the next layer until the output layer. The fully connected layer of the convolutional neural network consists of 48 distinct weights. These weights came from the 12 nodes from the convolution layer that are connected to the 4 nodes from the output layer.

As shown in equation 3, the nodes in the output layer are updated through getting the matrix product of the values of the activated nodes from the convolution layer and the values of the fully connected layer weights.

$$AB = [c_{ij}], c_{ij} = a_{i1}a_{1j} + a_{i2}a_{2j} + ... + a_{in}a_{nj} \tag{3}$$

After going through the forward pass, the output data is then calculated for errors depending on the target values corresponding to each number from 0-9 and the output result [1]. This is calculated by using the formula in equation 4.

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2 \tag{4}$$

Through the use of chain rule, the weights of the fully connected layer are translated from equation 5.

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial neto1} * \frac{\partial net_{o1}}{\partial w_5} \tag{5}$$

Expanding equation 5, the partial derivative of each entities are translated in equation 6.

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1} \tag{6}$$

X. Output Layer

A. Classification

Classification of the output value is determined by getting the position of the highest value in the output matrix. For example, if the position of highest value of the output matrix is at node 0, the recognized digit is 0. If it is at node 4, the recognized digit is 4. This pattern is also applied to the remaining digits to determine the recognized digit from the output. Figure 10 shows a diagram of the classification method.

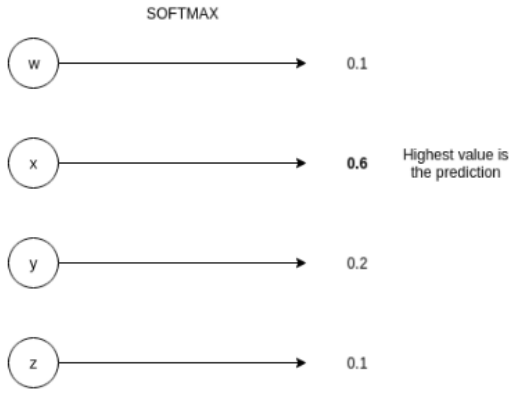

Fig. 10. Sample Classification Representation

B. Target Output

The target output is defined as a matrix which consists of 4 numbers. A value of 1 is initialized into the position of the desired output number for training and the other values of the matrix are initialized as 0. For example, if the desired digit to be trained is a 1, the value of node 1 of the matrix would be 1 and the other values in other positions would be 0. This pattern is also applied to the other digits. Figure 11 shows the target output diagram of each output node.

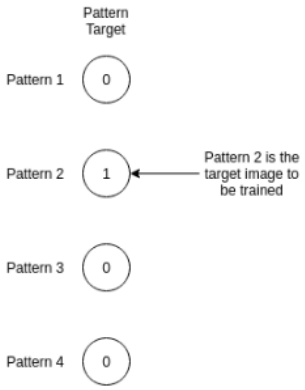

Fig. 11. Sample Target Output Representation of Pattern to be trained

XI. Results and Discussion

Figure 12 is the sample input image used. The input image will be trained for 1000 iterations. All of the filter weights are initiated to a value of 0.1 in each pixel.

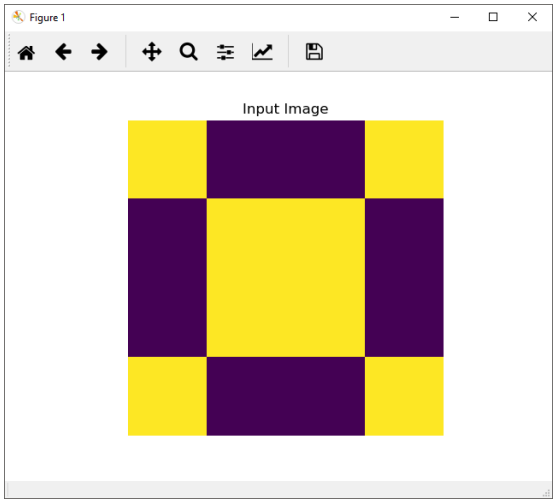

Fig. 12. Input 4x4 Black and White Image

Table I shows the CSV generated set of random initial fully-connected layer weights.

TABLE I
Initial Fully Connected Weights

	1	2	3	4
1	5.26E-01	1.84E-01	6.92E-01	5.37E-01
2	1.72E-01	9.24E-01	5.32E-02	8.79E-01
3	3.48E-02	3.42E-01	9.56E-01	7.50E-01
4	7.32E-02	9.43E-01	4.61E-02	8.84E-01
5	6.80E-01	1.17E-01	4.83E-01	4.42E-01
6	3.52E-01	9.69E-01	9.26E-01	5.00E-01
7	7.45E-02	4.41E-01	2.78E-01	9.28E-02
8	1.68E-01	1.78E-01	7.34E-01	7.24E-01
9	3.68E-02	8.37E-01	4.47E-01	7.30E-01
10	9.15E-02	9.13E-01	7.94E-01	2.26E-01
11	5.89E-01	3.49E-02	5.05E-01	2.05E-01
12	9.04E-02	8.26E-01	4.42E-01	1.29E-01

After the end of 1000 iterations, the final filter weights attained are shown in figure 13. The filters have captured the features of the input image.

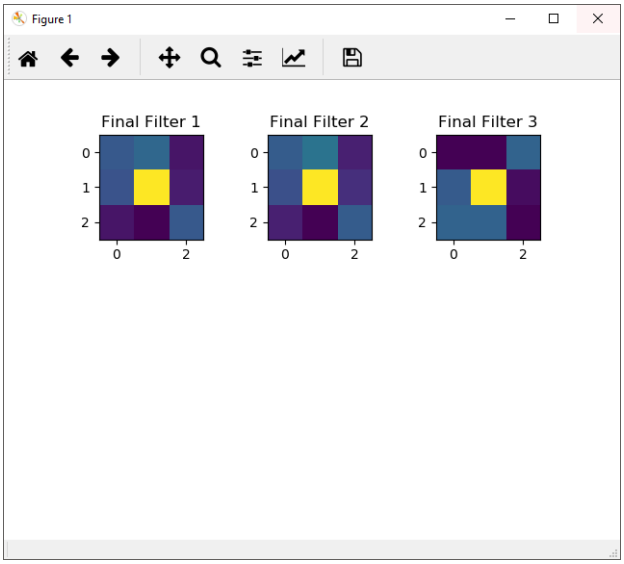

Fig. 13. Trained Filters After 1000 Iterations

As stated above, the initial, maximum, and minimum weights are saved for future use. Table II shows the trained fully connected weights after 1000 iterations.

TABLE II
Trained Fully Connected Weights After 1000 Iterations

	1	2	3	4
1	1.06E+00	-9.19E-02	4.46E-01	3.12E-01
2	7.02E-01	6.47E-01	-1.93E-01	6.53E-01
3	5.65E-01	6.60E-02	7.09E-01	5.24E-01
4	6.03E-01	6.67E-01	-2.00E-01	6.58E-01
5	1.21E+00	-1.57E-01	2.38E-01	2.18E-01
6	8.82E-01	6.93E-01	6.80E-01	2.75E-01
7	6.00E-01	1.68E-01	3.41E-02	-1.31E-01
8	6.96E-01	-9.66E-02	4.89E-01	5.00E-01
9	5.65E-01	5.61E-01	2.01E-01	5.05E-01
10	6.20E-01	6.38E-01	5.49E-01	1.19E-03
11	1.12E+00	-2.39E-01	2.61E-01	-1.79E-02
12	6.17E-01	5.51E-01	1.97E-01	-9.54E-02

The minimum and maximum weight values are also captured. The minimum filter weight values are shown in table III while the minimum fully connected weight values are shown in table IV

The maximum filter weight values are shown in table V while the maximum fully connected weight values are shown in table VI

TABLE III
MINIMUM FILTER WEIGHTS IN 1000 ITERATIONS

row	column	depth	value
0	0	0	0.1
0	0	1	0.1
0	0	2	0.1
0	1	0	0.1
0	1	1	0.1
0	1	2	0.1
0	2	0	0.1
0	2	1	0.1
0	2	2	0.1
1	0	0	0.1
1	0	1	0.1
1	0	2	0.1
1	1	0	0.1
1	1	1	0.1
1	1	2	0.1
1	2	0	0.1
1	2	1	0.1
1	2	2	0.1
2	0	0	0.1
2	0	1	0.1
2	0	2	0.1
2	1	0	0.1
2	1	1	0.1
2	1	2	0.1
2	2	0	0.1
2	2	1	0.1
2	2	2	0.1

TABLE IV
MINIMUM FULLY CONNECTED WEIGHTS IN 1000 ITERATIONS

	1	2	3	4
1	5.37E-01	-9.19E-02	4.46E-01	3.12E-01
2	1.83E-01	6.47E-01	-1.93E-01	6.53E-01
3	4.52E-02	6.60E-02	7.09E-01	5.24E-01
4	8.36E-02	6.67E-01	-2.00E-01	6.58E-01
5	6.90E-01	-1.57E-01	2.38E-01	2.18E-01
6	3.62E-01	6.93E-01	6.80E-01	2.75E-01
7	8.49E-02	1.68E-01	3.41E-02	-1.31E-01
8	1.78E-01	-9.66E-02	4.89E-01	5.00E-01
9	4.72E-02	5.61E-01	2.01E-01	5.05E-01
10	1.02E-01	6.38E-01	5.49E-01	1.19E-03
11	6.00E-01	-2.39E-01	2.61E-01	-1.79E-02
12	1.01E-01	5.51E-01	1.97E-01	-9.54E-02

TABLE V
MAXIMUM FILTER WEIGHTS IN 1000 ITERATIONS

row	column	depth	value
0	0	0	1.649771
0	0	1	1.889302
0	0	2	1.177499
0	1	0	1.61235
0	1	1	1.792264
0	1	2	1.66894
0	2	0	1.262554
0	2	1	1.47403
0	2	2	1.722817
1	0	0	1.74792
1	0	1	2.075665
1	0	2	1.279811
1	1	0	2.912326
1	1	1	3.363333
1	1	2	2.900316
1	2	0	1.204172
1	2	1	1.287668
1	2	2	1.719117
2	0	0	1.262554
2	0	1	1.47403
2	0	2	1.722817
2	1	0	1.299976
2	1	1	1.571069
2	1	2	1.231376
2	2	0	1.649771
2	2	1	1.889302
2	2	2	1.177499

TABLE VI
MAXIMUM FULLY CONNECTED WEIGHTS IN 1000 ITERATIONS

	1	2	3	4
1	1.06E+00	1.55E-01	6.71E-01	5.21E-01
2	7.02E-01	8.95E-01	3.25E-02	8.63E-01
3	5.65E-01	3.14E-01	9.35E-01	7.34E-01
4	6.03E-01	9.14E-01	2.53E-02	8.68E-01
5	1.21E+00	8.88E-02	4.62E-01	4.26E-01
6	8.82E-01	9.40E-01	9.05E-01	4.84E-01
7	6.00E-01	4.13E-01	2.57E-01	7.67E-02
8	6.96E-01	1.49E-01	7.13E-01	7.08E-01
9	5.65E-01	8.08E-01	4.26E-01	7.14E-01
10	6.20E-01	8.84E-01	7.73E-01	2.10E-01
11	1.12E+00	6.27E-03	4.85E-01	1.89E-01
12	6.17E-01	7.98E-01	4.21E-01	1.13E-01

XII. CONCLUSION

The implementation of the architecture consisted of different parts such as the input layer, the convolution layer, and the output layer. The filter weights are weights that helped to capture the features of the input image. The weights in the fully connected layer also help gather more features. The architecture of the convolutional neural network was implemented successfully. The final filters were able to capture the features of the input pattern. Each layer was also implemented through only using the matrix manipulation packages in Python as stated above. The minimalist scale helped in visualizing and understanding the concept of the convolutional neural network. This kind of architecture can be expanded in the future for other purposes regarding image processing. The study was able to focus on the mathematical part of the convolutional neural network. As seen above, the different derivations of the layer functions were implemented.

ACKNOWLEDGMENT

The completion of this project could not have been possible without the participation and assistance of many people whose names may not all be enumerated. Their contributions are sincerely appreciated. However the group would like to express their deep gratitude particularly to our thesis adviser, Engr. Roderick Y. Yap, for guiding and providing us important knowledge regarding the topic. The study would not be possible without his advise regarding the topic.

REFERENCES

[1] M. Mazur, "A step by step backpropagation example,"
[2] "What is numpy," 2017.
[3] "Pillow overview," 2018.
[4] Edureka, "What is matplotlib," 2014.
[5] J. H. D. D. E. F. M. Droettboom, "Pyplot tutorial," 2017.
[6] "Introduction to scipy," 2017.
[7] "Python data analysis library," 2017.

[8] M. Agrawal, “Back propagation in convolutional neural networks - intuition and code,” 2017.
[9] J. H. C. Morag;, “The influence of the sigmoid function parameters on the speed of backpropagation learning,” *From Natural to Artificial Neural Computation*, 1995.
[10] A. Walia, “Activation functions and its types-which is better?,” 2017.
[11] C. Han, Jun; Morag, *The influence of the sigmoid function parameters on the speed of backpropagation learning*. 1995.