Handwritten Digit Image Recognition using Neural Network

A Project by

Reinelle Jan C. Bugnot
Francis Q. Laboguin
Nicholas Gabriel T. Ramirez

Submitted to

Luisito L. Agustin
Instructor

In Partial Fulfillment of the Requirements for the Course
ELC 152.1: Signal Processing

Department of Electronics, Computer and Communications Engineering
School of Science and Engineering
Loyola Schools
Ateneo de Manila University
Quezon City, Philippines

December 2018

Abstract

This project is an image processing program, developed using C++, capable of recognizing and identifying an image of a handwritten digit. A graphical user interface was developed using the wxWidget that will allow the user to upload and display the image to be recognized as well as the confidence level of prediction for all the digits between 0 to 9.. Afterwards, pre-processing techniques were implemented in order to clean the image and format it in a form that can be easily processed by the recognition algorithm. Concretely, this is done by applying (1) a gaussian filter in order to remove the unnecessary details of the image such as textures and contours, (2) a desaturation function that converts the colored image into its grayscale counterpart, and (3) an color inversion function that inverts the desaturated colors so as to make the background darkner in contrast to the written text. The digit recognition system is a 3-layered feedforward neural network machine learning algorithm utilizing a sigmoid activation function. This algorithm takes in as input an array of 400 values, each element representing the pixel brightness of a 20 x 20 image, and outputs 10 values representing the 10 digits, 0 ~ 9. The output neuron with the highest value will be the class prediction of the network. Overall, the system was able load images and perform all the necessary pre-processing and the neural network algorithm was able to accurately classify the input handwritten images.

Acknowledgements

We would like to send our deepest gratitude to our ELC 152.1 Professor, Dr. Luisito Agustin for teaching us digital signal processing in a fun and interactive manner. We would also like to thank him for the several opportunities to consult with him as we developed our project which really helped us. He is the prime witness of how much we struggled, and then later succeeded, all-throughout the development of this project. Without his availability during consultations and kind support, it would have been impossible for us to accomplish this project.

Lastly, we would like to thank our batchmates, particularly the members of TEAM LARO > ARAL Facebook group chat, for always being there to answer our C++ questions and boost our morale whenever we feel unconfident of our capacity to finish this project. Thank you for keeping our spirits up.

Table of Contents

1. Introduction

In the advent of smaller and more portable computers along with highly variant and complex fields of applications, the effectivity of using a keyboard and mouse as input interfaces have been relatively diluted as both physical and technical limitations were reached. To solve this, scientists and engineers looked back into the communication skill that man have developed for thousands of years; namely, speech and handwriting. This gave birth to the development of speech and handwriting recognition systems which found numerous applications in countless fields such as neuroscience, bio-medical engineering, and the development of smartphone applications. In light of this, our project will focus on developing a simple subset of a handwriting recognition system; that is, a handwritten digit recognizer using primarily the C++ programming language.

1.1. Objectives

This project aims to develop an assistive tool in recognizing simple text using machine learning. Specifically, this project aims to accomplish the following three (3) objectives: The first objective is to design a graphical user interface that can ask the user to input digital images through an upload button, and then display the input image in the GUI. The second objective is to pre-process the digital image via Gaussian Filter, Image Desaturation, and Color Inversion in order to prepare the image for handwriting recognition. Finally, the last objective is to develop a handwriting recognition software using machine learning, particularly, Neural Network.

1.2. Significance of the Study

The significance of this project can be seen in the context of the developing computer-human interface. With this project, this enables computers to recognize handwritten numbers input as images by the user. Hence, this serves as a step towards computers becoming smarter and more integrated directly in human society

1.3. Scope and Limitations

The scope of this project only revolves around developing a handwritten digit recognition software using feedforward neural networks and implemented using C++ along with a graphical user interface for image input. That being said, letters and other non-

numerical symbols are out of the scope of this project. Images of handwritten digits with convoluted features such as strikethroughs, noisy backdrops, and the like might not be accurately classified by the system since the network was trained to read well-written digits on a clean canvass. Also, our program only accepts 20 px by 20 px input images which is based on the MNIST Open Source database of 60,000 20x20 (or 28x28) images of handwritten digits. Advanced handwritten classifiers utilize resampling algorithm in order to apply the developed machine learning system onto higher resolution images. However, resampling is a complex filtering method that can already be considered a digital signal processing project on its own. Furthermore, the recognition algorithm used was a simple feedforward neural network algorithm. The architecture and implementation of the trained neural network prediction system was developed in C++, however, the weighted values of the network was previously trained using a process developed in the Octave 4.1.4 language and is thus excluded from the scope of this project.

2. Theoretical Background

2.1. The Digital Image

For this project, digital images are the dependent variable where these will be subjected to manipulation and adjustments. The digital image is made of picture elements called pixels. These pixels are arranged into a rectangular array or a matrix with a number of columns and rows. [1] It can be seen in Figure 1 the specific pixel within the image matrix, and it can be defined as coordinates at x and y.

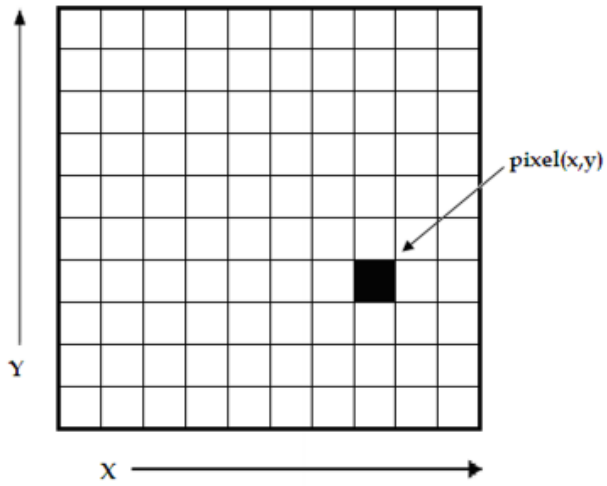

Fig. 1 Digital Image Pixel Array [1]

A parameter used for digital images is the image size. An image size is the number of pixels within a digital image [1]. Basically, this can be same as the matrix size of the pixels. To add, another parameter for digital images is the resolution. This indicates the spatial scale of image pixels [1]. Resolution can be used with a unit of pixels per inch (ppi). For example, an image of 300 x 300 pixels with a resolution of 30 pixels per inch has an image size of 10 x 10 inches.

The parameter that is vital in developing this project is the Red, Green Blue (RGB) value per pixel of the digital image. This parameter defines the digital image with its intensity, color and brightness. The various mixtures of these color intensities produces a color image. This is possible because the intensity values of digital images are defined by bits [1]. The standard digital photo uses an 8-bit range of values; RGB images use 8-bit intensity ranges for each color and black and white images have a single 8-bit intensity range [1]. In RGB values, each color per pixel has an intensity range from 0 to 255. For this project, the

RGB values of each pixel of the digital image will be manipulated in order to execute the handwriting recognition process.

2.2. Image Desaturation

Image Desaturation simply means to convert the colors present into shades of gray. In other words, to obtain the grayscale of the image. Moreover, the amount of how gray the color becomes is based on intensities of the red, green, and blue colors of the image [3].

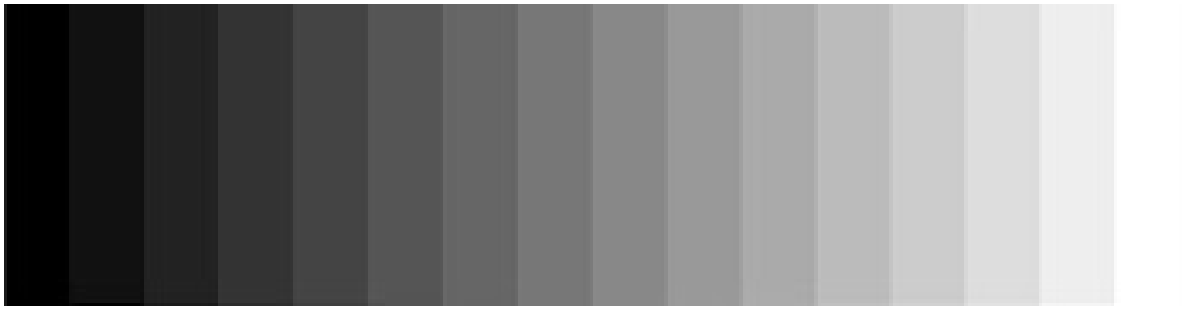

Fig. 2 Gaussian filter applied to an image [4]

Given this, in order to convert an image into grayscale, the following weights are used in order to dictate how gray the image will become. Hence, the following equation is used to convert the image to grayscale:

$$Gray = 0.2989R + 0.58700G + 0.1140B \quad (1)$$

2.3. Color Inversion

The concept of color inversion is used in the program given the limitation of the nueral network algorithm. It is used in order to invert the grayscale color of the image. That is, the white background of the image is turned into black while the black colored number is turned into white. The concept behind this is mirroring about the value 128. Since RGB values are from 0 to 255, to invert the color is simply to subtract 255 by the RGB value.

2.4. Gaussian Filter

Gaussian filters are an example of linear filters. Linear filters are tools used to manipulate images by replacing each pixel of the image by a linear combination of the pixel

and its surrounding neighbors. This is done by using a kernel that acts as the mask of the process. The kernel is convolved with each pixel of the image in order to apply the filter [2]

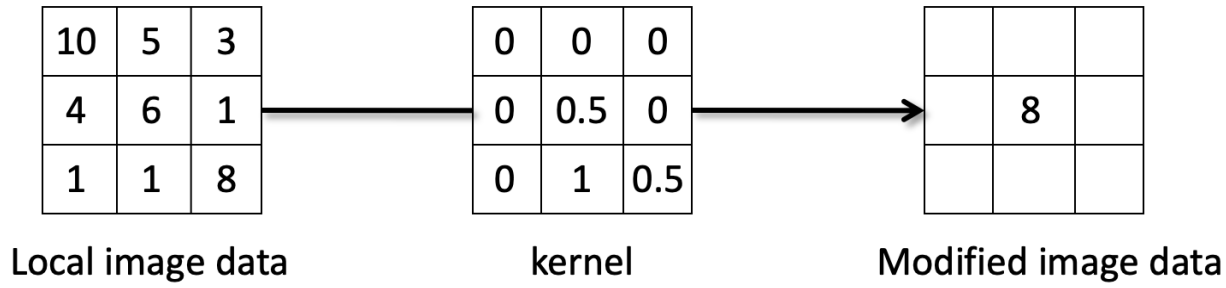

Fig. 3 Process of Linear Filtering [2]

Gaussian filters remove the high frequency components of an image. Hence, its effect is that the image becomes smoother and more blurred; its edges become less defined when this operation is performed. In reference to the study, this is required in order to reduce noise and unwanted segments of the image for the detection algorithm.

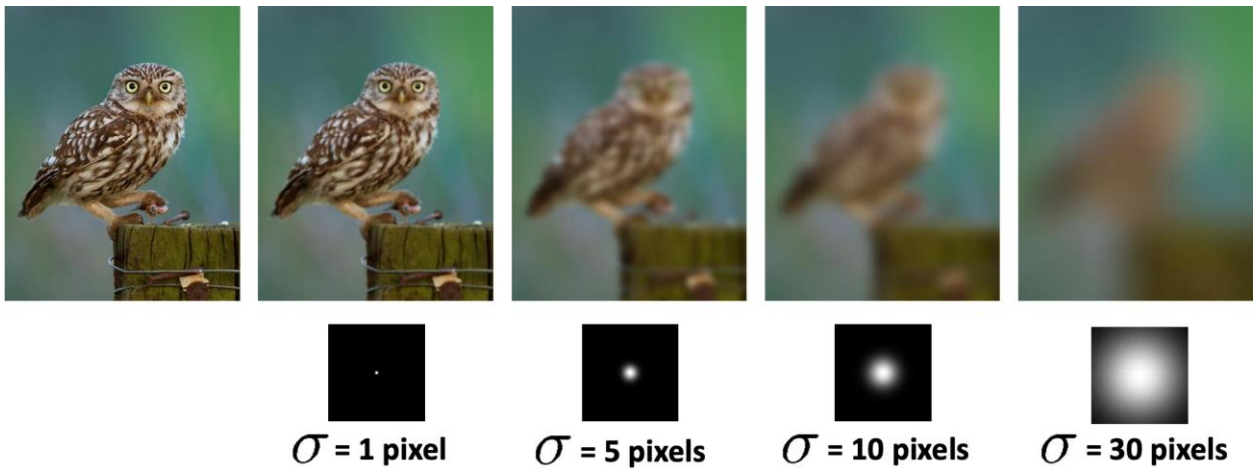

Fig. 4 Gaussian filter applied to an image [2]

2.5. Machine Learning: Overview

As opposed to traditional programming algorithms, machine learning algorithms allow for the performance of a task without explicit coding of the task in question. This is especially useful in applications wherein manual coding is not an effective option. For instance, identifying the image of a cat in a sample of images of different animals or predicting the price of a house given the floor size, the material, the numbers floors, etc. Manually coding these tasks entails countless loops, complicated functions, and endless arrays of data. Even then, the final product may still not perform as effectively [5].

Machine learning tasks are divided into two primary categories based on the type of problem: classification and regression. In the recent examples, the former (identifying the image of a cat) is an example of a classification problem whereas the latter is an example of a regression problem [5]. Simply put, a classification problems requires an answer or a *prediction* from a discrete set of possible outcomes—is this an image of a cat, dog, *or* car? is the tumor malignant *or* benign? In contrast, regression problems selects from a continuous stream of possible outputs—given these specifications, how *much* is this product?

Machine learning tasks are also divided based on the presence of preliminary information about the system: supervised and unsupervised. In supervised machine learning, training data is initially provided. The training data is a (huge) set of inputs wherein the correct outputs--otherwise identified as the "ground truth"--are already paired with their corresponding inputs [5]. These data will be used to create the algorithm by adjusting the necessary parameters so as to generate predictions that match the correct answers. On the other hand, unsupervised machine learning is an approach that directly tackles a data set, without the presence of any training data, and proceeds to find patterns and clusters among the analyzed data set. These patterns and clusters will then be grouped together under a single class or label that identifies them [5].

All these being said, this project is a supervised classification problem since the task revolves around the identification of handwritten digits from digital images and since training data was gathered and then utilized in the development of the machine learning algorithm called *Neural Network*.

2.6. Machine Learning: Neural Network

A Neural Network (NN) is a machine learning algorithm that is patterned on the behavior of an animal neuron. This model is comprised of 3 basic layers: an input layer, a hidden layer, and an output layer, as shown in Fig. 5 [6].

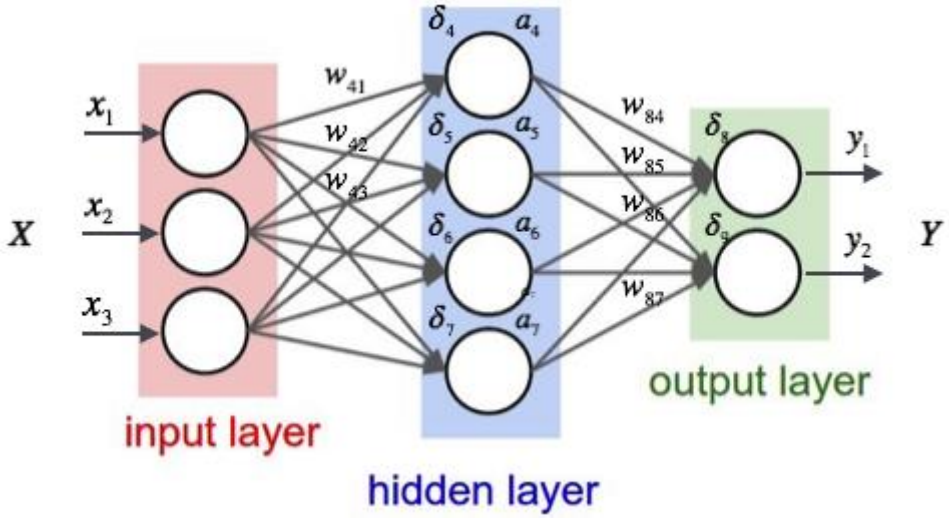

Fig. 5. Layers of a Neural Network with inputs x, outputs y, weights w, activations a, and derivates δ. The number of hidden layers can exceed 1 [6].

Each layer has an arbitrary number of neurons. In the mathematical context, a neuron can also be referred to as a perceptron. Each neuron carries a particular activation value which is a function of the inputs of that neuron that describes the degree onto which the neuron affects that particular input [6]. Numerous different activation functions exists; the simplest and possibly most effective in this application is the Sigmoid Function which is model by Eq (2) and graphically visualized in Fig. 6. This function takes in as input any real number and outputs a normal value between 0 and 1 [6].

$$g(z) = \frac{1}{1+e^{(-z)}} \tag{2}$$

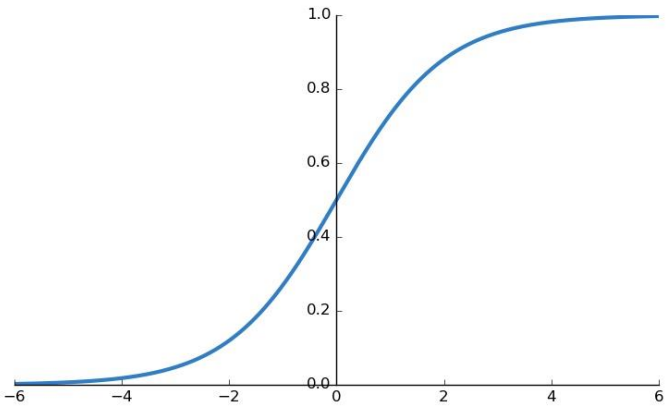

Fig. 6. The Sigmoid activation function [7].

All the neurons in one layer are connected to all the other neurons in the other layers via synapses that contains weight values. The weight describes how much that particular path from one neuron to the next contributes in the prediction of the output. A neuron typically has multiple input and output terminals [6]. The output of the neuron is calculated by taking the summation of the products of the corresponding inputs and weights. This is visually elaborated in Fig. 7. The neurons of the input layer take in as inputs the features directly from the data set and then passes the output to the next layer, and so forth, until the output layer receives the corresponding inputs and then calculates the output which will then indicate the prediction of the model [6].

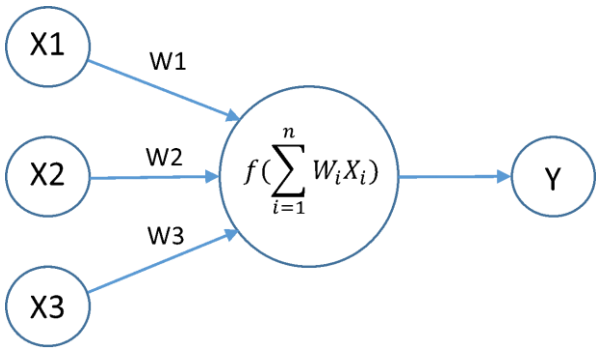

Fig. 7. The perceptron, a mathematical model of the biological neuron.

The Neural Network is trained by gathering inputs and calculating every perceptron outputs via activations starting from the input layer and progresses forward until the output layer [6]. This process is called forward propagation. The output layers will then determine the prediction of the model wherein the neuron with the highest output is the prediction. This final result will then be cross-referenced with the training data containing the correct output corresponding to the input. The error will be calculated and then the weights of the input terminals of the neurons will be adjusted based on the degree of error incurred defined by δ. The adjustments of the weights will progress from the output layer towards the input layer in a process called back propagation. This cycle repeats for every training data until a desired low prediction error level is achieved [6].

3. Project Implementation

The overview of the algorithms used to implement this project is shown in Figure 8. There are 8 algorithms that were utilized: image uploading, grayscale, color inversion, gaussian filter, getArray, Unrolling. Mapping, and the Neural Network--the details of which will be discussed in depth in the succeeding sections.

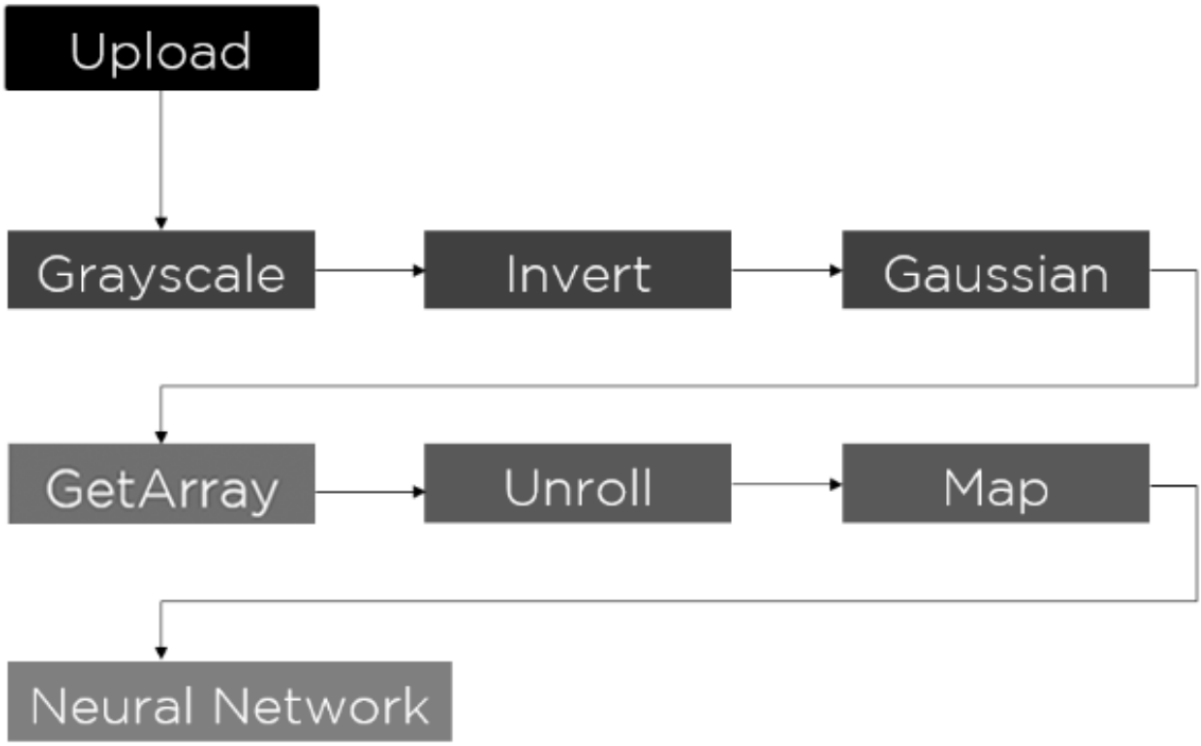

Fig. 8. Project Implementation Methodology

3.1. Creating the Graphical User Interface

The Graphical User Interface (GUI) of the program was designed using the software wxDevC++. This is a good software to be used for project integration because of its application and substitution to wxWidgets library.

In creating the GUI of the project, common controls were needed. Static text, static bitmap, and buttons were essential in designing the GUI of the project. The static text was used to show the title and the prediction of the handwritten image. The prediction will be based on the integration of neural networks to the GUI. Static bitmap was also used to display the pictures that was uploaded and manipulated using filters. Lastly, buttons were used to run the command of uploading an image, applying filters to the image and the handwriting image recognition process. The GUI of this project can be seen in Figure 9.

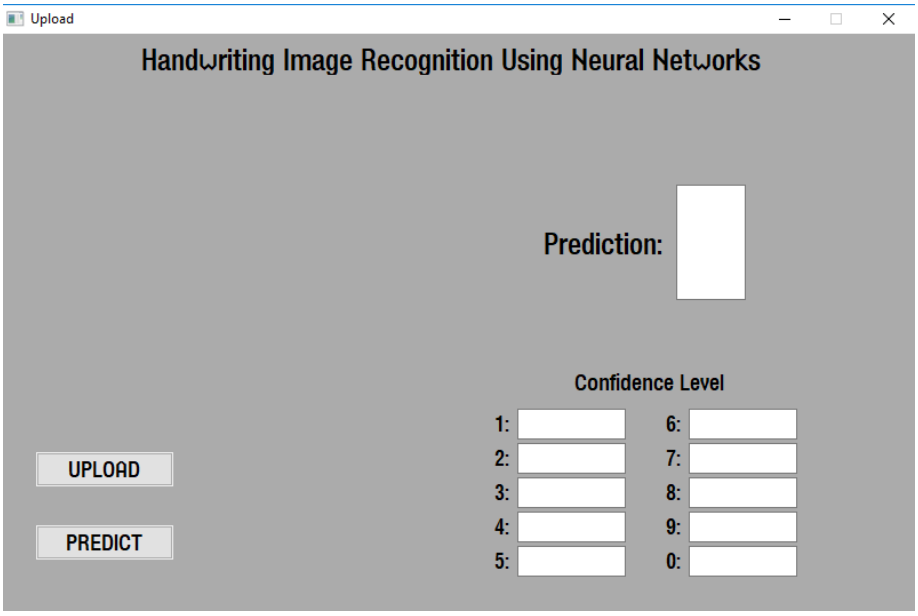

Fig. 9 Handwriting Image Recognition Graphical User Interface

3.2. Uploading an Image

A function that is important in developing the project is uploading an image. In the GUI of the project as can be seen in Figure 9, an uploading function is needed to add the input image for recognition. In order to execute this function, a button, static bitmap and an open file dialog were needed. The vital part of the uploading function is the open file dialog where in it is an additional GUI to choose and upload the picture from the computer storage.

```
void UploadFrm::WxButton1Click(wxCommandEvent& event)
{
UploadFileDialog -> ShowModal();
if (UploadFileDialog -> GetPath().IsEmpty())
{
    return;
}
Upload_Pic.LoadFile(UploadFileDialog -> GetPath(), wxBITMAP_TYPE_ANY);
int h = Upload_Pic.GetHeight();
int w = Upload_Pic.GetWidth();
if (h !=315 && w != 222)
{
    wxImage buff = Upload_Pic;
Upload_Bitmap -> SetBitmap(buff.Scale(315,222));
}
else
{
    Upload_Bitmap -> SetBitmap(Upload_Pic);
}
```

Fig. 10. Image Upload Function using wxDevC++

A backend code is also necessary in uploading images for the GUI of the project as it can be seen in Figure 10. First, *wxImage Upload_Pic* was declared on the *.h file* of the project as a global object. This was used to arbitrary store the uploaded picture and place it on the

static bitmap of the GUI. *UploadFileDialog* is the name of the open file dialog that was used for the project. Also, the uploaded picture was scaled in order to fit the dimensions of the static bitmap from the GUI. The dimensions were listed in the code. This uploaded picture will then be subjected to pixel manipulation.

3.3. Obtaining the Pixel Parameters

The RGB values of the pixels of the image were obtained using the command, *var.GetRed(x,y)*, *var.GetGreen(x,y)*, and *var.GetBlue(x,y)*. In these commands, *var* is the variable representing the image from which the pixels will be obtained, *x* and *y* are the horizontal and vertical components of the image; location of the pixel.

```
r_a = my_image.GetRed(x-1,y+1);
g_a = my_image.GetGreen(x-1,y+1);
b_a = my_image.GetBlue(x-1,y+1);
```

Fig. 11 Sample code of obtaining the RGB values of a pixel

3.4. Desaturation Function

The desaturation function was implemented by first obtaining the RGB values of each pixel of the image. For each pixel of the image, the obtained RGB values are used as inputs to Eqn. (1) in order to obtain the grayscale of the image. This implementation was modeled after the work of

```
wxImage final_grayscale(wxImage img)
{

  for (int i = 0; i < img.GetWidth(); i++)
  {
    for (int j = 0; j < img.GetHeight(); j++)
    {
      unsigned char r = img.GetRed(i,j);
      unsigned char g = img.GetGreen(i,j);
      unsigned char b = img.GetBlue(i,j);
      unsigned char gray = (r*299 + g*587 + b*114)/1000;
      img.SetRGB(i,j,gray,gray,gray);
```

```
    }
  }
  return img;
}
```

3.5. Color Inversion Function

The color inversion function was implemented by subtracting the RGB values of each pixel of the image from 255. Hence, the image RGB values were mirrored across RGB value 128, thereby inverting the desaturated image. Moreover, since this operation was performed after the image desaturation, obtaining one of the RGB values is sufficient. The code is shown below.

```
wxImage final_invert(wxImage images)
{
      for (int i = 0; i < images.GetWidth(); i++)
  {
    for (int j = 0; j < images.GetHeight(); j++)
    {
      unsigned char pre_invert = images.GetRed(i,j);

      unsigned char invert = 255 - pre_invert;
      images.SetRGB(i,j,invert,invert,invert);

    }
}
  return images;
}
```

3.6. Gaussian Filtering Function

The image was subjected to a Gaussian filter and the following 3x3 kernel was used. This was based on the previous implementation of this by Lungasin, Mallari, and Singh in 2015.

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

Fig. 11 3x3 Kernel used for Gaussian filtering [8]

The RGB values of each pixel of the image and its neighboring pixels are stored in respective variables in order to convolve these with the 3x3 kernel. Each RGB value obtained is then multiplied to the respective values of the 3x3 kernel and these are added and stored in an array. Afterwards, the new RGB value of the pixel is set to the obtained convolved values of pixels.

```
wxImage final_gaussian(wxImage my_image)
{
    int image_width = my_image.GetWidth();
    int image_height = my_image.GetHeight();
    unsigned char gaussian_red[image_width][image_height];
    unsigned char gaussian_green[image_width][image_height];
    unsigned char gaussian_blue[image_width][image_height];

    for(int y = 0 ; y < image_height; y++){

        for(int x = 0; x < image_width; x++){

            unsigned char rgb_a = 0;
            unsigned char rgb_b = 0;
            unsigned char rgb_c = 0;
            unsigned char rgb_d = 0;
            unsigned char rgb_e = 0;
            unsigned char rgb_f = 0;
            unsigned char rgb_g = 0;
            unsigned char rgb_h = 0;
            unsigned char rgb_i = 0;

            if(x>0 && y<image_height-1){
```

```
//x-1,y+1
rgb_a = my_image.GetRed(x-1,y+1);

}

if(y<image_height-1){
//x,y+1
rgb_b = my_image.GetRed(x,y+1);

}

if(x<image_width-1 && y<image_height-1){
//x+1,y+1
rgb_c = my_image.GetRed(x+1,y+1);

}

if(x>0){
//x-1,y
rgb_d = my_image.GetRed(x-1,y);

}


//x,y
rgb_e = my_image.GetRed(x,y);


if(x<image_width-1){
//x+1,y
rgb_f = my_image.GetRed(x+1,y);

}
```

```
if(x>0 && y>0){
//x-1,y-1
rgb_g = my_image.GetRed(x-1,y-1);

}

if(y>0){
//x,y-1
rgb_h = my_image.GetRed(x,y-1);

}

if(x<image_width-1 && y>0){
//x+1,y-1
rgb_i = my_image.GetRed(x+1,y-1);

}

gaussian_red[x][y] = ((rgb_a) + (2*rgb_b)
+ (rgb_c) + (2*rgb_d) + (4*rgb_e) +
(2*rgb_f) + (rgb_g) + (2*rgb_h) +
(rgb_i))/16;

gaussian_green[x][y] = ((rgb_a) + (2*rgb_b)
+ (rgb_c) + (2*rgb_d) + (4*rgb_e) +
(2*rgb_f) + (rgb_g) + (2*rgb_h) +
(rgb_i))/16;

gaussian_blue[x][y] = ((rgb_a) + (2*rgb_b)
+ (rgb_c) + (2*rgb_d) + (4*rgb_e) +
(2*rgb_f) + (rgb_g) + (2*rgb_h) +
(rgb_i))/16;

my_image.SetRGB(x, y, gaussian_red[x][y],
```

```
            gaussian_green[x][y], gaussian_blue[x][y]);
        }
    }

return my_image;

}
```

3.7. Get Array

The RGB values of each pixel were passed to the neural network algorithm by first storing these in a 20x20 array. That is, a for loop was implemented so that it would cover each pixel of the image and store each RGB value of the pixel. However, because the image was grayscaled, this meant that only one of the RGB values was needed since the others were equal. The code is shown below.

```
for (int i = 0; i < output.GetWidth(); i++)
{
  for (int j = 0; j < output.GetHeight(); j++)
  {
   int rgb_values=output.GetRed(i,j);
   rgb_array[i][j]=rgb_values;
  }
}
```

3.8. Unroll Array

After the 20x20 array was obtained, this was then passed to a 1x400 array. The rationale behind this was due to the specifications of the neural network algorithm. That is, it only accepts data in a 1x400 array. Hence, the 20x20 array was converted using the code below.

```
for (int i = 0; i < 20; i++)
 {
       for (int j = 0; j < 20; j++)
       {
```

```
            unrolledarray[1][i*20+j]= rgb_array[i][j];
        }

    }
```

3.9. Mapping Function

After the 20x20 array was unrolled, the last step before the data can be fully passed to the neural network algorithm is to map the pixel data.The values required for the neural network algorithm are from -0.15 to 1.10. Hence, as seen in the code below, each element of the 1x400 array is subject to calculations then stored in a new 1x400 array. Moreover, another specification of the neural network is to append a value of 1 in the first element of the new 1x400 array. The code below displays these processes

```
  for(int j = 0; j < 400; ++j)
        {
            mapped[1][j]=((unrolledarray[1][j]/255)*1.30)     -
0.15;
        }

        for(int i = 400; i>=0; --i)
            {
                input[1][i+1] = mapped[1][i];
            }
            input[1][0] = 1;
```

3.10. Implementing Neural Network in Handwriting Recognition

In order to perform the handwriting recognition task, a feedforward Neural Network algorithm was implemented. Neural networks perform classification through the use of a hypothesis function $h(\theta)$ characterized by the neural network architecture and the *weights* assigned to each link. As seen in Fig 5, each neuron, further deconstructed in Fig. 7, is connected to the neurons of the next layer via weighted links. The product of the weight and the output of a neuron in layer l will be part of the summed input to the neuron in layer $l+1$. Intuitively speaking, the weights determine the “degree of importance” of that particular link in the identification of a particular digit. Hence, “training a neural network” is numerically

summarized as finding the appropriate weight values for every link in the network that will correctly classify every class, given input values. Finding the appropriate weights is done through a process called *backpropagation* which compares the output prediction with the "ground truth" and then adjusts the prediction by nudging the weights of the links accordingly starting from the output layer towards the input layer; hence, backward propagation. As mentioned in Chapter 1.3, the training process is excluded from the scope and was instead implemented in another programming language (Octave 4.1.4).

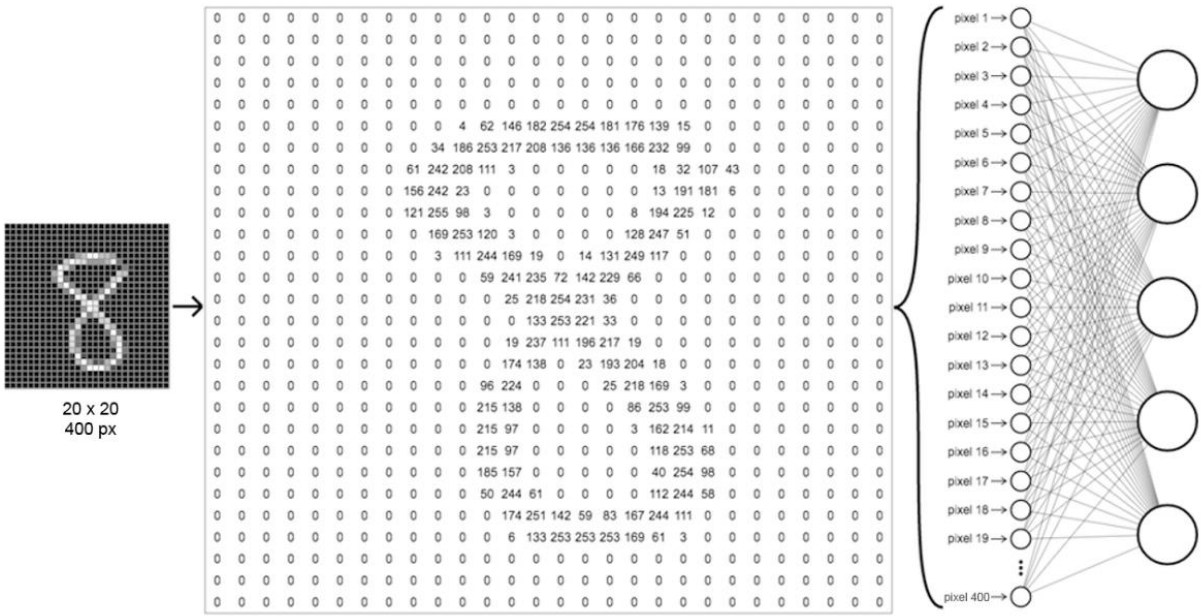

Fig. 12. Unmapped 20x20 px image array of the digit 8 and how it is allocated in the input layer of the network (for visualization purposes).

Our base input is a 20 x 20 pixel grayscale image of a handwritten digit (the image will be rescaled to this size if larger or smaller). The features taken from the image for recognition are the values pertaining to the brightness of every pixel in the image. In standard format, this value ranges from 0 to 255, wherein the closer the value is to 255, the whiter the pixel becomes. In our network format, this value ranges from -0.185 to 1.134 which are float values derived from the conversion from the idx non-standard image format. Nevertheless, the logic stays the same. The closer the value of the pixel is to 1.134, the whiter the pixel is; otherwise, the darker it is. Hence, we can represent our 20 x 20 pixel image with a 20 x 20 2D array of float values, each cell value representing the brightness of the corresponding pixel.

```
Sample # 0 :
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 1 1 0 0 0 1 1 1 0 0 0 1 1 0 0 0
0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0
0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6.8934e-005 -0.00078585 -0.0008851 2.7574e-005 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1.2766e-005 7.9997e-005 -0.0020127 -0.018915 -0.035914 -0.036998 -0.01756 0.00019361 -5.4041e-005 -0.009
0748 -0.030281 -0.024516 -0.0014621 0.00019659 1.4623e-017 7.2594e-019 0 0 0 4.0765e-006 0.00046041 -0.011399 -0.0043
844 0.29942 0.50987 0.4153 0.18671 -0.023609 -0.014401 0.096953 0.34107 0.27343 -0.0017996 -0.019422 -0.0016807 0.000
28595 0 0 0 0.00051867 -0.013703 0.098941 0.73224 0.8154 0.79305 0.874 0.84982 0.54439 0.62091 0.7971 0.86431 0.85968
 0.77146 0.3365 -0.024338 -0.002434 0 0 0 -0.0038168 -0.00069587 0.64372 0.8227 0.13533 -0.024728 0.068406 0.57666 1.
0365 0.57534 0.16414 0.10551 0.15132 0.60232 0.88613 0.24099 -0.01712 0 0 0 -0.028375 0.23431 0.78984 0.1651 -0.0189
0.00028709 -0.049654 0.52539 1.0072 0.32439 -0.059285 -0.010318 -0.024377 -0.014874 0.64905 0.62608 -0.019486 0 0 0 -
0.048616 0.43871 0.68823 -0.018447 -0.0029421 -0.0068117 0.018907 0.52737 0.78306 0.13431 -0.018254 2.0425e-005 0.001
052 -0.051429 0.41335 0.67652 -0.015074 0 0 0 -0.034789 0.30374 0.67481 0.066499 -0.024717 -0.037949 0.32647 0.99246
0.58479 -0.022544 -0.022924 -0.019797 -0.033459 0.099433 0.78738 0.46733 -0.020779 0 0 0 -0.0042852 0.010948 0.51519
0.73351 0.5273 0.61613 0.84462 0.51943 0.68022 0.74863 0.47159 0.21553 0.37012 0.8362 0.66715 0.067192 -0.0099989 0 0
 0 0.00051771 -0.0061195 -0.007527 0.30641 0.45857 0.53459 0.23696 -0.013189 0.03415 0.33039 0.61611 0.59722 0.62359
0.38171 0.027196 -0.013922 0.00038382 0 0 0 0 0.00015574 -0.00049488 -0.027104 -0.040203 -0.032277 -0.021593 -0.00068
423 -0.004339 -0.026182 -0.026625 -0.021016 -0.024972 -0.030663 -0.0043337 0.00057198 4.2552e-006 0 0 0 0 0 0 1.3067e
-018 6.5334e-019 -0.0014681 6.5334e-019 0 3.0637e-005 -0.00028493 -0.002759 -0.0031578 -0.0029887 -0.00032261 3.0331e
-005 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Fig. 13. A 20 x 20 (+1 bias) array of pixel values representing the digits 8. Binarized for visual purposes (top), the original float values used by the network (bottom). The image is tranposed to adapt to the network.

In order to input the image to our neural network, we have to 'unroll the array' by turning the 20 x 20 2D array into a 1D array containing 400 elements wherein the first 20 elements is the first row of the 2D array, the second 20 is the second row, and so forth. A collection of *m* samples of handwritten images are compiled in a .csv file. Hence, we have a *m* x 400 2D array, each row containing an array of 400 values describing the brightness of every pixel of one image, as shown below. For the purposes of premilinary testing, 10 *m* samples were used for ease of access and computation.

```
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1.
9301e-05,-0.00035064,-3.9675e-05,3.2169e-06,0,0,0,0,0,0,1.1029e-
05,-7.0772e-05,-0.0017263,-0.0013498,-0.00010747,2.0935e-05,0,0,
0,0,0.00073162,-0.01153,-0.027401,-0.007237,0.00089519,0,0,0,0,0
.00015409,-0.00034835,-0.019962,-0.016429,-0.020455,-0.0079295,0
.00048628,0,0,0,5.0917e-18,-0.0087902,0.14796,0.19643,0.025092,-
0.011998,0.00056168,5.3149e-18,0,0,-0.0011471,-0.00035692,0.1754
4,0.4835,0.42196,0.054849,-0.0077732,0,0,0,0.0002079,-0.034537,0
.52618,0.94707,0.50433,0.014336,-0.016736,0.00063998,3.0637e-05,
0,-0.0043745,0.014476,0.44782,0.89035,1.0408,0.53786,-0.021159,0
,0,0,-0.0071524,0.025663,0.73272,1.0381,0.99415,0.70782,0.10672,
-0.017824,0.0006769,0,-0.00058945,0.0035919,0.015456,0.18882,0.8
6312,0.87733,-0.020758,0,0,0,-0.0099095,0.050004,0.78266,1.0207,
1.0266,1.0599,0.63564,0.028027,-0.0089184,1.2561e-05,8.5606e-05,
-0.00067121,-0.036475,0.2551,0.9931,0.87426,-0.020712,0,0,0,-0.0
13561,0.084602,0.80438,1.0571,0.56579,0.56224,0.91068,0.16452,-0
.025388,3.1658e-05,8.0714e-05,-0.0033667,-0.0083516,0.61178,1.04
21,0.59921,-0.020875,0,0,0,-0.029276,0.23293,0.92263,0.93339,0.1
995,0.12762,1.0037,0.57592,-0.0061015,-0.0025795,0.0010501,-0.02
6395,0.22105,0.9416,1.0036,0.25305,-0.02014,0,0,0,-0.029673,0.23
663,0.93891,0.81779,0.068297,0.002608,0.62912,0.89109,0.044457,-
```

Fig. 14. A snippet of how the input data (array of float values) csv file looks like

Concretely, the neural network architecture for this application contains 3 layers, further illustrated in Figure 15. One input layer with 401 neurons, each neuron taking in one value from the array of 400 pixel values plus an extra bias neuron with the value permanently set to 1. One hidden layer with 25 neuron plus a bias neuron. And lastly, one output layer with 10 neurons, each neuron representing the 10 digits 0 ~ 9. Each neuron from one layer is connected to the next via fully-connected weighted links. The collection of links that connect the input layer to the hidden layer will herein be referred to as *Theta1* and the links that connect the hidden layer to the output layer, *Theta2*. Hence, *Theta1* will have an array dimension of 401 x 25 and *Theta2* will have a dimension of 26 x 10. The values of which have already been trained prior to implementation through Octave.

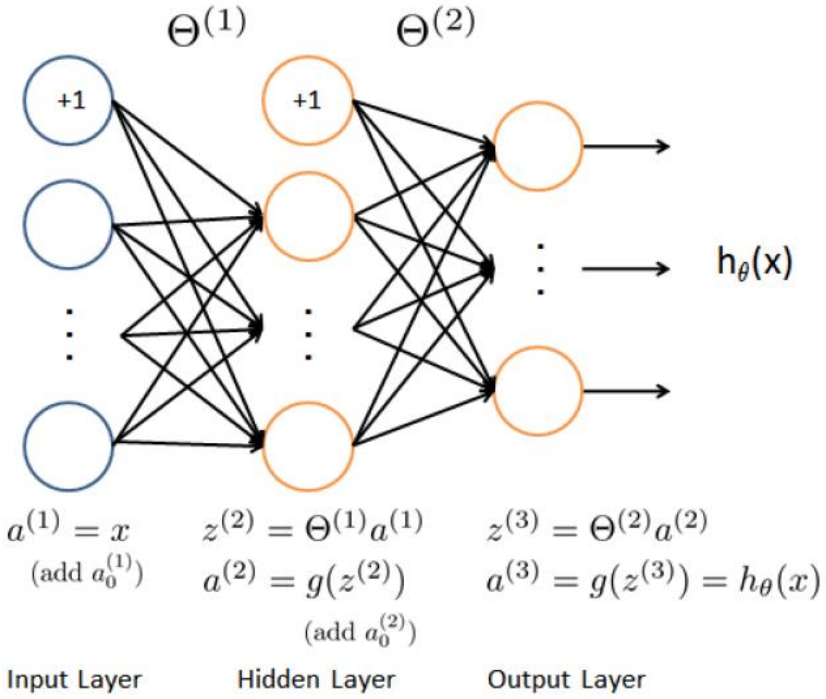

Fig. 15. The Neural Network Architecture, the first layer contains 401 neurons, the second contains 26, the last contains 10,

The implementation of the feedforward neural network is a straightforward series of matrix multiplications and activation operations among arrays containing the input values and arrays containing the parameters of the neural network. First, the input image in the input array will be loaded to the input layer, herein referred to as *a_1* by passing every value of input to *a_1*. Afterwards, the *a_1* 2D array will be multiplied to the tranpose of *Theta1* via matrix multiplcation. The product of which will be pass through a sigmoid activation function along with the bias neuron appended to it (refer to Eq. (2)) and is then passed to the hidden layer, *a_2*. In order to perform all of these, the functions shown below were written and utilized.

```
//MULTIPLY axb * bxc = axc
for (int i = 0; i < 10; i++)
{
    for (int j = 0; j < 25; j++)
    {
        a_2[i][j] = 0;
        for (int k = 0; k < 401; k++)
        {
            a_2[i][j] += a_1[i][k] * Theta1Trans[k][j];
        }
    }
}
```

Fig. 16. Matrix multiplication function

```
//TRANPOSE
for(int i = 0; i < 25; ++i)
{
    for(int j = 0; j < 401; ++j)
    {
        Theta1Trans[j][i]=Theta1[i][j];
    }
}
```

Fig. 17. Matrix transpose function

```
//SIGMOID
for(int i = 0; i < 10; ++i)
{
    for(int j = 0; j < 25; j++)
    {
        a_2[i][j] = 1 / (1 + exp(-1*a_2[i][j]));
    }
}
```

Fig. 18. The Sigmoid activation function

The transition from the hidden layer to the output layer runs the same procedure as well. Matrix *a_2* will be multiplied to the transpose of *Theta2* and then passed on a sigmoid activation function before storing in *a_3.* At this point, the values within *a_3* should be a set of float values between 0 and 1, referring to the 'confidence' level of the prediction scheme. That is, a 0.97 in the neuron corresponding to digit 3 means that the network is 97% confident that the input image is the handwritten digit 3. In a well-trained network, the output layer would typically contain only one dominant neuron while the other neurons hold very small values. Therefore, *prediction* in a neural network simply means identifying the output neuron with the largest value. In terms of arrays, this is done by identifying the index of the largest value of the output array.

```
a_3 values:
0.000447819 0.00249788 0.00630663 0.0027802 0.0192169 0.00127968 5.81825e-005 0.950303 0.00693832 0.00370668
0.000338134 0.000232397 0.00186752 0.00862076 0.00358321 0.000423603 0.018288 0.00303453 0.980949 0.000102509
0.000164354 0.0024371 0.988268 0.00100454 0.000247758 5.97841e-006 0.00741965 0.00461215 0.0277722 0.000278454
0.945684 0.0041381 0.00199106 7.56113e-005 0.00205423 0.0179937 0.00241033 0.0992218 0.00165431 0.000134131
0.000592284 0.000229075 0.0113775 0.962643 0.0476909 0.00165693 0.00182483 0.0275245 0.0706432 3.50209e-005
0.00483125 0.0131691 5.20122e-005 0.00181543 0.985156 0.00521417 0.000208069 0.00478554 0.000148248 0.0033242
0.00159191 0.000555048 0.0128271 0.0254455 0.0131623 0.000199458 0.0130089 0.000474583 0.947503 0.000114967
6.94637e-005 0.00252734 0.00310037 1.37406e-005 0.0245478 0.00853002 0.00240243 0.00110083 0.0141148 0.980428
0.00101983 0.0614509 0.992687 4.39511e-006 0.00949279 4.00889e-005 0.000467075 0.000575834 0.000239329 0.000687075
0.000253671 0.000419671 0.995674 0.00143064 0.00187592 1.31809e-006 0.00143004 0.010215 0.00285563 0.000756803
```

Fig. 19. a_3 values for 10 image examples. Each row corresponds to one sample image. Each column is the 'confidence' that the input image corresponds to the digit that is being represented by the index of the array.

Ideally, this means that if the largest value is in index 1, then our output prediction is 1, or when the largest value is in index 8, the prediction is 8, and so forth. However, recall that our network was previously trained in Octave 4.1.4 wherein arrays begins at index 1. The neuron corresponding to digit 0 was moved to index 10. Hence, if the value in index 10 is the largest, then the output prediction is 0. If the index 1 is the highest, the prediction is 1. In C++, arrays begin at index 0. This introduces a slight discrepancy between the indexing format that the network was trained for and the indexing format of the implementation. Here, index 0 now contains the value correponding to output prediction 1. On the other hand, index 9 now corresponds to output prediction 0. A simple calibration function was developed in order to resolve this discrepancy. Simply put, if the index of the highest value of the array lies between the 1st and the 9th index (index 0 to 8), the prediction will be shifted upward (+1). If the highest value lies in the 10th index (index 9), the prediction will be set to 0, as shown in the code snippet below.

```
//MAX INDEX
 for (int i = 0; i < 10; i++)
{
   uncalib_pred = distance(a_3[i], max_element(a_3[i], a_3[i] + 10));

   if (uncalib_pred < 9)
   {
       calib_pred = uncalib_pred + 1;
   }
   else
   {
       calib_pred = 0;
   }

   cout << "Prediction: "
   << "(for sample # " << i << ")    "
   << calib_pred
   << endl;
}
```

Fig. 20. .Prediction and Calibration function. In C++, the index of the largest element in an array can be determined by finding the distance between each element and the maximum of that array.

The accuracy of the prediction algorithm, which is 96.86%, was calculated in Octave 4.1.4 for technical simplicity and is calculated by taking the average of the percentile confidence level unto which the prediction matches the desired output, concretely shown by the following equation implemented in Octave 4.1.4 (can also be done in MATLAB).

$$mean(\ double(\ pred == y)\ ) \cdot 100 \tag{3}$$

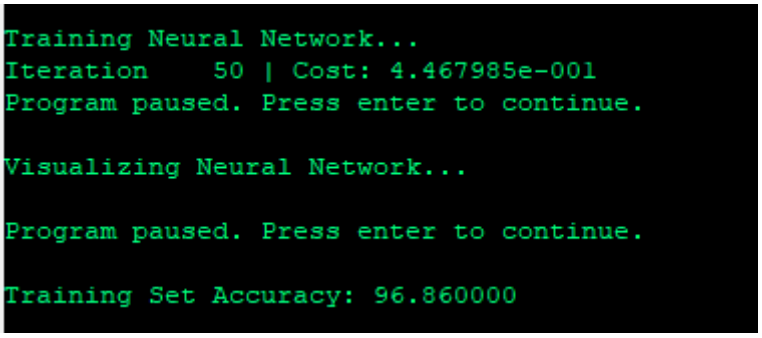

Fig. 21. Prediction algorithm training set accuracy after training for 50 iterations (Octave 4.1.4).

Lastly, the following figure depicts the result and output of the number detection algorithm. Presented are the uploaded image, thumbnails of the 3 pre-processing techniques applied, the actual numerical prediction, and the confidence level on each of the 10 digit choices in percentage.

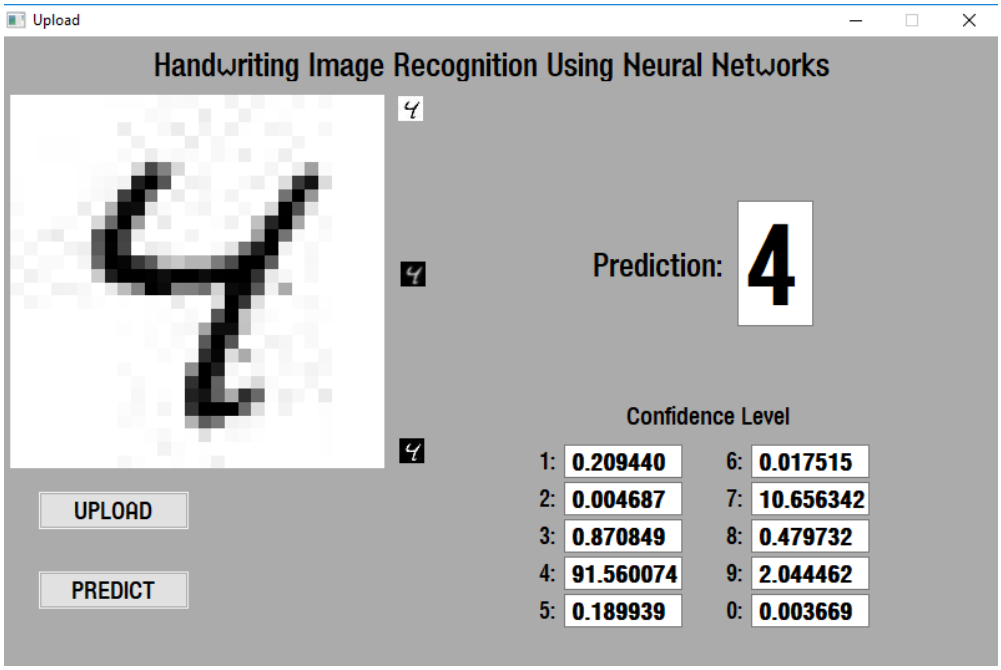

Fig. 22. .Output prediction of the program.

4. Conclusion and Recommendation

4.1. Conclusion

In conclusion, the students were able to succesfully develop a program that takes a user uploaded image of a number and outputs the ‘read’ prediction of what the input number is. This was done by applying 3 primary pre-processing techniques to enhance the image; namely and in order, grayscale/desaturation, color inversion, and gaussian filtering. Afterwards, the resulting image is converted to packets of data stored in specified arrays and mapped along a certain defined axis before being fed into the Neural Network algorithm for prediction. The prediction accuracy was calculated to be at 96.86%, therefore, the algorithm was able to accomplish its task within the well-defined scope of the study. The program was developed using the C++ language, and the GUI of the program was developed using wxDevC++ and wxWidgets.

4.2. Recommendation

For future studies, the students would like to recommend on researching on related literatures to implement multiple character recognition on one picture. That is, being able to recognize the number, ‘68’ by identifying the individual digits ‘6’ and ‘8.’ Also, symbols such as the alphabet, special characters, and foreign language alphabets are also possible areas of expansion. Furthermore, we strongly suggest that the future proponents that would be interested in continuing this project to develop a resampling algorithm which is an algorihtm capable of resizing (upscale or downscale) images. This can be used in order to expand the digit recognition algorithm to images of higher resolution. Examples of such algorithms are the k-Nearest Neighbor, Bilinear, and Bicubic Interpolation.

The primary challenge that the students faced during project development was that the uncertainty regarding the feasibility of the project since there wasn’t any past machine learning projects previously implemented in the past DSP classes that can be used an a basis for our development. On the other hand, available C++ source codes of handwriting recognition systems online often either uses libraries and functions prohibited in the DSP course or are too complicated to work on. Hence, the major chunk of our code regarding recognition had to be manually coded. The syntax of which could be further developed and optimized. We hope that our project can be used as a foundation for future digital signal processing projects that opt to implement neural networks or handwriting recognition systems.

Appendix

```
  ///-------------------------------------------------------------
--------
///
/// @file      UploadFrm.cpp
/// @author    Bugnot, Laboguin, Ramirez
/// Created:   22/10/2018 3:39:41 PM
/// @section   DESCRIPTION
///            UploadFrm class implementation
///
///-------------------------------------------------------------
-------

#include "UploadFrm.h"

//Do not add custom headers between
//Header Include Start and Header Include End
//wxDev-C++ designer will remove them
////Header Include Start
////Header Include End
#include <fstream>
#include <iostream>
#include <math.h>
#include <sstream>
#include <string>
#include <cmath>
#include <algorithm>
using namespace std;
//-------------------------------------------------------------
---------------
// UploadFrm
//-------------------------------------------------------------
---------------
```

```
//Add Custom Events only in the appropriate block.
//Code added in other places will be removed by wxDev-C++
////Event Table Start
BEGIN_EVENT_TABLE(UploadFrm,wxFrame)
     ////Manual Code Start
     ////Manual Code End

     EVT_CLOSE(UploadFrm::OnClose)
     EVT_ACTIVATE(UploadFrm::UploadFrmActivate)

     EVT_TEXT(ID_CONFIDENCE_VALUE_0,UploadFrm::WxEdit1Updated)

     EVT_TEXT(ID_CONFIDENCE_VALUE_9,UploadFrm::WxEdit1Updated)

     EVT_TEXT(ID_CONFIDENCE_VALUE_8,UploadFrm::WxEdit1Updated)

     EVT_TEXT(ID_CONFIDENCE_VALUE_7,UploadFrm::WxEdit1Updated)

     EVT_TEXT(ID_CONFIDENCE_VALUE_6,UploadFrm::WxEdit1Updated)

     EVT_TEXT(ID_CONFIDENCE_VALUE_5,UploadFrm::WxEdit1Updated)

     EVT_TEXT(ID_CONFIDENCE_VALUE_4,UploadFrm::WxEdit1Updated)

     EVT_TEXT(ID_CONFIDENCE_VALUE_3,UploadFrm::WxEdit1Updated)

     EVT_TEXT(ID_CONFIDENCE_VALUE_2,UploadFrm::WxEdit1Updated)

     EVT_TEXT(ID_CONFIDENCE_VALUE_1,UploadFrm::WxEdit1Updated)

     EVT_TEXT(ID_PREDICTION_VALUE,UploadFrm::WxEdit1Updated)
     EVT_BUTTON(ID_WXBUTTON2,UploadFrm::WxButton2Click)
     EVT_BUTTON(ID_UPLOAD,UploadFrm::WxButton1Click)
END_EVENT_TABLE()
```

```
////Event Table End

UploadFrm::UploadFrm(wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &position, const wxSize& size, long style)
: wxFrame(parent, id, title, position, size, style)
{
     CreateGUIControls();
}

UploadFrm::~UploadFrm()
{
}

void UploadFrm::CreateGUIControls()
{
     //Do not add custom code between
     //GUI Items Creation Start and GUI Items Creation End
     //wxDev-C++ designer will remove them.
     //Add the custom code before or after the blocks
     ////GUI Items Creation Start

     wxInitAllImageHandlers();     //Initialize graphic format handlers

     UploadFileDialog =   new wxFileDialog(this, _("Choose a file"), _(""), _(""), _("*.*"), wxFD_OPEN);

     confidence_value_0      =      new      wxTextCtrl(this, ID_CONFIDENCE_VALUE_0, _(""), wxPoint(600, 448), wxSize(120, 27), 0, wxDefaultValidator, _("confidence_value_0"));
     confidence_value_0->SetFont(wxFont(16, wxSWISS, wxNORMAL, wxBOLD, false));
```

```
    confidence_value_9 = new wxTextCtrl(this,
ID_CONFIDENCE_VALUE_9, _(""), wxPoint(600, 418), wxSize(120,
27), 0, wxDefaultValidator, _("confidence_value_9"));
    confidence_value_9->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxBOLD, false));

    confidence_value_8 = new wxTextCtrl(this,
ID_CONFIDENCE_VALUE_8, _(""), wxPoint(600, 388), wxSize(120,
27), 0, wxDefaultValidator, _("confidence_value_8"));
    confidence_value_8->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxBOLD, false));

    confidence_value_7 = new wxTextCtrl(this,
ID_CONFIDENCE_VALUE_7, _(""), wxPoint(600, 358), wxSize(120,
27), 0, wxDefaultValidator, _("confidence_value_7"));
    confidence_value_7->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxBOLD, false));

    confidence_value_6 = new wxTextCtrl(this,
ID_CONFIDENCE_VALUE_6, _(""), wxPoint(600, 328), wxSize(120,
27), 0, wxDefaultValidator, _("confidence_value_6"));
    confidence_value_6->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxBOLD, false));

    WxStaticText13 = new wxStaticText(this,
ID_WXSTATICTEXT13, _("0:"), wxPoint(580, 450), wxDefaultSize,
0, _("WxStaticText13"));
    WxStaticText13->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxNORMAL, false, _("Siemens AD Sans")));

    WxStaticText12 = new wxStaticText(this,
ID_WXSTATICTEXT12, _("9:"), wxPoint(580, 420), wxDefaultSize,
0, _("WxStaticText12"));
```

```
    WxStaticText12->SetFont(wxFont(16,   wxSWISS,   wxNORMAL,
wxNORMAL, false, _("Siemens AD Sans")));

    WxStaticText11       =       new       wxStaticText(this,
ID_WXSTATICTEXT11, _("8:"), wxPoint(580, 390), wxDefaultSize,
0, _("WxStaticText11"));
    WxStaticText11->SetFont(wxFont(16,   wxSWISS,   wxNORMAL,
wxNORMAL, false, _("Siemens AD Sans")));

    WxStaticText10       =       new       wxStaticText(this,
ID_WXSTATICTEXT10, _("7:"), wxPoint(580, 360), wxDefaultSize,
0, _("WxStaticText10"));
    WxStaticText10->SetFont(wxFont(16,   wxSWISS,   wxNORMAL,
wxNORMAL, false, _("Siemens AD Sans")));

    WxStaticText9 = new wxStaticText(this, ID_WXSTATICTEXT9,
_("6:"),     wxPoint(580,     330),     wxDefaultSize,     0,
_("WxStaticText9"));
    WxStaticText9->SetFont(wxFont(16,   wxSWISS,   wxNORMAL,
wxNORMAL, false, _("Siemens AD Sans")));

    confidence_value_5      =      new      wxTextCtrl(this,
ID_CONFIDENCE_VALUE_5, _(""), wxPoint(450, 448), wxSize(120,
27), 0, wxDefaultValidator, _("confidence_value_5"));
    confidence_value_5->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxBOLD, false));

    confidence_value_4      =      new      wxTextCtrl(this,
ID_CONFIDENCE_VALUE_4, _(""), wxPoint(450, 418), wxSize(120,
27), 0, wxDefaultValidator, _("confidence_value_4"));
    confidence_value_4->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxBOLD, false));
```

```
    confidence_value_3 = new wxTextCtrl(this,
ID_CONFIDENCE_VALUE_3, _(""), wxPoint(450, 388), wxSize(120,
27), 0, wxDefaultValidator, _("confidence_value_3"));
    confidence_value_3->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxBOLD, false));

    confidence_value_2 = new wxTextCtrl(this,
ID_CONFIDENCE_VALUE_2, _(""), wxPoint(450, 358), wxSize(120,
27), 0, wxDefaultValidator, _("confidence_value_2"));
    confidence_value_2->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxBOLD, false));

    WxStaticText8 = new wxStaticText(this, ID_WXSTATICTEXT8,
_("5:"), wxPoint(430, 450), wxDefaultSize, 0,
_("WxStaticText8"));
    WxStaticText8->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxNORMAL, false, _("Siemens AD Sans")));

    WxStaticText7 = new wxStaticText(this, ID_WXSTATICTEXT7,
_("4:"), wxPoint(430, 420), wxDefaultSize, 0,
_("WxStaticText7"));
    WxStaticText7->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxNORMAL, false, _("Siemens AD Sans")));

    WxStaticText6 = new wxStaticText(this, ID_WXSTATICTEXT6,
_("3:"), wxPoint(430, 390), wxDefaultSize, 0,
_("WxStaticText6"));
    WxStaticText6->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxNORMAL, false, _("Siemens AD Sans")));

    WxStaticText5 = new wxStaticText(this, ID_WXSTATICTEXT5,
_("2:"), wxPoint(430, 360), wxDefaultSize, 0,
_("WxStaticText5"));
```

```
    WxStaticText5->SetFont(wxFont(16,    wxSWISS,    wxNORMAL,
wxNORMAL, false, _("Siemens AD Sans")));

    confidence_value_1      =      new      wxTextCtrl(this,
ID_CONFIDENCE_VALUE_1,  _(""),  wxPoint(450,  328),  wxSize(120,
27), 0, wxDefaultValidator, _("confidence_value_1"));
    confidence_value_1->SetFont(wxFont(16, wxSWISS, wxNORMAL,
wxBOLD, false));

    WxStaticText4 = new wxStaticText(this, ID_WXSTATICTEXT4,
_("1:"),     wxPoint(430,     330),     wxDefaultSize,     0,
_("WxStaticText4"));
    WxStaticText4->SetFont(wxFont(16,    wxSWISS,    wxNORMAL,
wxNORMAL, false, _("Siemens AD Sans")));

    WxStaticText3 = new wxStaticText(this, ID_WXSTATICTEXT3,
_("Confidence Level (%)"), wxPoint(487, 292), wxDefaultSize,
0, _("WxStaticText3"));
    WxStaticText3->SetFont(wxFont(16,    wxSWISS,    wxNORMAL,
wxNORMAL, false));

    BmpOutput2  =  new  wxStaticBitmap(this,  ID_BMPOUTPUT2,
wxNullBitmap, wxPoint(318, 92), wxSize(20, 20) );

    prediction_value       =       new       wxTextCtrl(this,
ID_PREDICTION_VALUE,  _(""),  wxPoint(581,  108),  wxSize(84,
146), 0, wxDefaultValidator, _("prediction_value"));
    prediction_value->SetForegroundColour(wxColour(0,0,255));
    prediction_value->SetFont(wxFont(100, wxSWISS, wxNORMAL,
wxBOLD, false));

    WxStaticText2 = new wxStaticText(this, ID_WXSTATICTEXT2,
_("Prediction:"),   wxPoint(411,   161),   wxDefaultSize,   0,
_("WxStaticText2"));
```

```
    WxStaticText2->SetFont(wxFont(24,    wxSWISS,    wxNORMAL,
wxBOLD, false));

    WxButton2    =    new    wxButton(this,    ID_WXBUTTON2,
_("PREDICT"),   wxPoint(40,   430),   wxSize(230,   50),   0,
wxDefaultValidator, _("WxButton2"));
    WxButton2->SetFont(wxFont(22, wxSWISS, wxNORMAL, wxBOLD,
false));

    WxStaticText1 = new wxStaticText(this, ID_WXSTATICTEXT1,
_("Handwriting  Image  Recognition  Using  Neural  Networks"),
wxPoint(36, 5), wxDefaultSize, 0, _("WxStaticText1"));
    WxStaticText1->SetFont(wxFont(20,    wxSWISS,    wxNORMAL,
wxBOLD, false));

    BmpOutput1   =   new   wxStaticBitmap(this,   ID_BMPOUTPUT1,
wxNullBitmap, wxPoint(317, 48), wxSize(20, 20) );

    BmpOutput   =   new   wxStaticBitmap(this,   ID_BMPOUTPUT,
wxNullBitmap, wxPoint(317, 70), wxSize(20, 20) );

    Upload_Bitmap       =       new       wxStaticBitmap(this,
ID_UPLOAD_BITMAP,  wxNullBitmap,  wxPoint(6,  47),  wxSize(300,
300) );
    Upload_Bitmap-
>SetForegroundColour(wxSystemSettings::GetColour(wxSYS_COLOUR_
WINDOW));

    Upload   =   new   wxButton(this,   ID_UPLOAD,   _("UPLOAD"),
wxPoint(40,  366),  wxSize(230,  50),  0,  wxDefaultValidator,
_("Upload"));
    Upload->SetFont(wxFont(22,   wxSWISS,   wxNORMAL,   wxBOLD,
false));
```

```
    SetTitle(_("Upload"));
    SetIcon(wxNullIcon);
    SetSize(8,8,808,529);
    Center();

    ////GUI Items Creation End
}

void UploadFrm::OnClose(wxCloseEvent& event)
{
    Destroy();
}
//Gaussian Filter Algorithm
wxImage final_gaussian(wxImage my_image)
{
    int image_width = my_image.GetWidth();
    int image_height = my_image.GetHeight();
    unsigned char gaussian_red[image_width][image_height];
    unsigned char gaussian_green[image_width][image_height];
    unsigned char gaussian_blue[image_width][image_height];

    for(int y = 0 ; y < image_height; y++){

        for(int x = 0; x < image_width; x++){

            unsigned char rgb_a = 0;
            unsigned char rgb_b = 0;
            unsigned char rgb_c = 0;
            unsigned char rgb_d = 0;
            unsigned char rgb_e = 0;
            unsigned char rgb_f = 0;
            unsigned char rgb_g = 0;
            unsigned char rgb_h = 0;
            unsigned char rgb_i = 0;
```

```
if(x>0 && y<image_height-1){
//x-1,y+1
rgb_a = my_image.GetRed(x-1,y+1);
}

if(y<image_height-1){
//x,y+1
rgb_b = my_image.GetRed(x,y+1);
}

if(x<image_width-1 && y<image_height-1){
//x+1,y+1
rgb_c = my_image.GetRed(x+1,y+1);
}

if(x>0){
//x-1,y
rgb_d = my_image.GetRed(x-1,y);
}


//x,y
rgb_e = my_image.GetRed(x,y);


if(x<image_width-1){
//x+1,y
rgb_f = my_image.GetRed(x+1,y);
}

if(x>0 && y>0){
//x-1,y-1
rgb_g = my_image.GetRed(x-1,y-1);
```

```
            }

            if(y>0){
            //x,y-1
            rgb_h = my_image.GetRed(x,y-1);
            }

            if(x<image_width-1 && y>0){
            //x+1,y-1
            rgb_i = my_image.GetRed(x+1,y-1);
            }

            gaussian_red[x][y] = ((rgb_a) + (2*rgb_b)
            + (rgb_c) + (2*rgb_d) + (4*rgb_e) +
            (2*rgb_f) + (rgb_g) + (2*rgb_h) +
            (rgb_i))/16;

            gaussian_green[x][y] = ((rgb_a) + (2*rgb_b)
            + (rgb_c) + (2*rgb_d) + (4*rgb_e) +
            (2*rgb_f) + (rgb_g) + (2*rgb_h) +
            (rgb_i))/16;

            gaussian_blue[x][y] = ((rgb_a) + (2*rgb_b)
            + (rgb_c) + (2*rgb_d) + (4*rgb_e) +
            (2*rgb_f) + (rgb_g) + (2*rgb_h) +
            (rgb_i))/16;

            my_image.SetRGB(x, y, gaussian_red[x][y],
            gaussian_green[x][y], gaussian_blue[x][y]);
        }
    }

return my_image;
}
```

```
//Grayscale Filter Algorithm
wxImage final_grayscale(wxImage img)
{

  for (int i = 0; i < img.GetWidth(); i++)
  {
    for (int j = 0; j < img.GetHeight(); j++)
    {
      unsigned char r = img.GetRed(i,j);
      unsigned char g = img.GetGreen(i,j);
      unsigned char b = img.GetBlue(i,j);
      unsigned char gray = (r*299 + g*587 + b*114)/1000;
      img.SetRGB(i,j,gray,gray,gray);
    }
  }

  return img;
}

//Invert Algorithm
wxImage final_invert(wxImage images)
{
      for (int i = 0; i < images.GetWidth(); i++)
  {
    for (int j = 0; j < images.GetHeight(); j++)
    {
      unsigned char pre_invert = images.GetRed(i,j);

      unsigned char invert = 255 - pre_invert;
      images.SetRGB(i,j,invert,invert,invert);

    }
}
```

```
  return images;
}

float UploadFrm::loadData1(string fileloc, const int rows,
const int cols)
{
    ifstream file(fileloc.c_str());
    if (file.is_open()) {
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                file >> Theta1[i][j];
                file.get(); // Throw away the comma
            }
        }
    }
}

float UploadFrm::loadData2(string fileloc, const int rows,
const int cols)
{
    ifstream file(fileloc.c_str());
    if (file.is_open()) {
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                file >> Theta2[i][j];
                file.get(); // Throw away the comma
            }
        }
    }
}

void UploadFrm::WxButton1Click(wxCommandEvent& event)
{
```

```
UploadFileDialog -> ShowModal();
if (UploadFileDialog -> GetPath().IsEmpty())
{
    return;
}

Upload_Pic.LoadFile(UploadFileDialog       ->       GetPath(),
wxBITMAP_TYPE_ANY);
int h = Upload_Pic.GetHeight();
int w = Upload_Pic.GetWidth();
if (h !=300 && w != 300)
{
    wxImage buff = Upload_Pic;

Upload_Bitmap -> SetBitmap(buff.Scale(300,300));
}
else
{
    Upload_Bitmap -> SetBitmap(Upload_Pic);
}
}

void UploadFrm::UploadFrmActivate(wxActivateEvent& event)
{

}

void UploadFrm::WxButton2Click(wxCommandEvent& event)
{
    //Grayscale Click
       grayscale_filter                                          =
final_grayscale(Upload_Pic.Scale(20,20));
    output = grayscale_filter;
    BmpOutput1 -> SetBitmap(output);
```

```
    //Invert Click
    invert_pic = final_invert(grayscale_filter);
    output = invert_pic;
    BmpOutput2 -> SetBitmap(output);

    //Gaussian Click
    gaussian_filter = final_gaussian(invert_pic);
    output = gaussian_filter;
    BmpOutput -> SetBitmap(output);

       for (int i = 0; i < output.GetWidth(); i++)
  {
    for (int j = 0; j < output.GetHeight(); j++)
    {
     int rgb_values=output.GetRed(i,j);
     rgb_array[i][j]=rgb_values;
    }
  }


    for (int i = 0; i < 20; i++)
  {
        for (int j = 0; j < 20; j++)
        {
            unrolledarray[1][i*20+j]= rgb_array[i][j];
        }

    }

       for(int j = 0; j < 400; ++j)
        {
            mapped[1][j]=((unrolledarray[1][j]/255)*1.30)     -
0.15;
```

```
    }

    for(int i = 400; i>=0; --i)
        {
            input[1][i+1] = mapped[1][i];
        }
        input[1][0] = 1;

UploadFrm::loadData1("Theta1.txt", 25, 401);
 UploadFrm::loadData2("Theta2.txt", 10, 26);


    //PASS

    for (int j = 0; j < 401; ++j)
    {
        a_1[1][j]=input[1][j];
    }


//TRANPOSE
for(int i = 0; i < 25; ++i)
 {
    for(int j = 0; j < 401; ++j)
    {
        Theta1Trans[j][i]=Theta1[i][j];
    }
}

 //a_1 = 10x401, Theta1Trans = 401x25, a_2 = 10x25
 //MULTIPLY axb * bxc = axc
 for (int i = 0; i < 1; i++)
{
    for (int j = 0; j < 25; j++)
```

```
    {
        a_2[i][j] = 0;
        for (int k = 0; k < 401; k++)
        {
            a_2[i][j] += a_1[i][k] * Theta1Trans[k][j];
        }
    }
}

 //SIGMOID
 for(int i = 0; i < 1; ++i)
 {
    for(int j = 0; j < 25; j++)
    {
      a_2[i][j] = 1 / (1 + exp(-1*a_2[i][j]));
    }
}

//APPEND

for(int i = 0; i < 1; ++i)
 {
    for(int j = 24; j >= 0; --j)
    {
        a_2App[i][j+1]=a_2[i][j];
    }
    a_2App[i][0] = 1;
}

//TRANPOSE
for(int i = 0; i < 10; ++i) //Theta2'
 {
    for(int j = 0; j < 26; ++j)
    {
```

```
            Theta2Trans[j][i]=Theta2[i][j];
        }
    }



     //MULTIPLY axb * bxc = axc
     //a_2App = 10x26, Theta2Trans = 26x10, a_3 = 10x10;
     for (int i = 0; i < 1; i++) //a_3 = a_2App*Theta2Trans'
    {
        for (int j = 0; j < 10; j++)
        {
            a_3[i][j] = 0;
            for (int k = 0; k < 26; k++)
            {
                a_3[i][j] += a_2App[i][k] * Theta2Trans[k][j];
            }
        }
    }

     //SIGMOID
     for(int i = 0; i < 1; ++i) //a_3 = sig(a_3)
     {
        for(int j = 0; j < 10; j++)
        {
          a_3[i][j] = 1 / (1 + exp(-1*a_3[i][j]));
        }
    }

for(int i = 0; i < 1; ++i)
     {
        for(int j = 0; j < 10; j++)
        {
          confidence_a_3[i][j] = a_3[i][j]*100;
```

```
        }
    }

    a_3_1 = confidence_a_3[0][0];
    a_3_2 = confidence_a_3[0][1];
    a_3_3 = confidence_a_3[0][2];
    a_3_4 = confidence_a_3[0][3];
    a_3_5 = confidence_a_3[0][4];
    a_3_6 = confidence_a_3[0][5];
    a_3_7 = confidence_a_3[0][6];
    a_3_8 = confidence_a_3[0][7];
    a_3_9 = confidence_a_3[0][8];
    a_3_0 = confidence_a_3[0][9];


     //MAX

     for(int i = 0; i < 1; ++i)
     {
        for(int j = 0; j < 10; j++)
        {
          conf[i] = max(conf[i],a_3[i][j]);
        }
    }
    //MAX INDEX
     for (int i = 0; i < 1; i++)
    {
       uncalib_pred   =   distance(a_3[i],   max_element(a_3[i],
a_3[i] + 10));

       if (uncalib_pred < 9)
       {
               calib_pred = uncalib_pred + 1;
        }
```

```
        else
        {
                calib_pred = 0;
        }

    }

prediction_value->SetValue(wxString::Format(wxT("%i"),
calib_pred));
confidence_value_1->SetValue(wxString::Format(wxT("%1f"),
a_3_1));
confidence_value_2->SetValue(wxString::Format(wxT("%1f"),
a_3_2));
confidence_value_3->SetValue(wxString::Format(wxT("%1f"),
a_3_3));
confidence_value_4->SetValue(wxString::Format(wxT("%1f"),
a_3_4));
confidence_value_5->SetValue(wxString::Format(wxT("%1f"),
a_3_5));
confidence_value_6->SetValue(wxString::Format(wxT("%1f"),
a_3_6));
confidence_value_7->SetValue(wxString::Format(wxT("%1f"),
a_3_7));
confidence_value_8->SetValue(wxString::Format(wxT("%1f"),
a_3_8));
confidence_value_9->SetValue(wxString::Format(wxT("%1f"),
a_3_9));
confidence_value_0->SetValue(wxString::Format(wxT("%1f"),
a_3_0));
}

void UploadFrm::WxEdit1Updated(wxCommandEvent& event)
{
}
```

References

[1] “1. What is a digital Image? - Learn ImageJ,” Google Sites. [Online]. Available: https://sites.google.com/site/learnimagej/image-processing/what-is-a-digital-image. [Accessed: 03-Nov-2018].

[2] N. Snavely, "Computer Vision", Online Class Lecture Notes, Cornell University. [Online]. Available: https://www.cs.cornell.edu/courses/cs6670/2011sp/lectures/lec02_filter.pdf. [Accessed: 03-Nov-2018]

[3] A. Lim, “Digitally Converting Color Image to Different Grayscale Intensities by Varying the RGB Scale”, Ateneo de Manila University, Quezon City, Philippines

[4] A. Toshniwal, “OpenCV C Tutorial And Examples,” How to convert a color image into grayscale image without using CV_LOAD_IMAGE_GRAYSCALE, 01-Jan-1970. [Online]. Available: http://opencv-tutorials-hub.blogspot.com/2015/12/how-to-convert-color-image-into-grayscaleopencvimage-without-using-CVLOADIMAGEGRAYSCALE-methods-of-converting-RGB-into-greyscale-image-luminosity-method-weighted-average-pixels-channels-of-greyscale-image-opencv-hub-code-examples.html. [Accessed: 03-Nov-2018].

[5] A. Ng, "Introduction to Machine Learning", Online Class Lecture Notes, Machine Learning - Stanford University, Coursera.org, 2011. [Online]. Available: https://www.coursera.org/learn/machine-learning/lecture/Ujm7v/what-is-machine-learning. [Accessed: 10- Sep- 2018]

[6] A. Ng, "Neural Networks: Representation", Online Class Lecture Notes, Machine Learning - Stanford University, Coursera.org, 2011. [Online]. Available: https://www.coursera.org/learn/machine-learning/lecture/ka3jK/model-representation-i. [Accessed: 10- Sep- 2018]

[7] V. Valkov, "Creating a Neural Network from Scratch", Medium, 2017. [Online]. Available: https://medium.com/@curiousily/tensorflow-for-hackers-part-iv-neural-network-from-scratch-1a4f504dfa8. [Accessed: 16- Oct- 2018].

[8] M. Lugnasin, D. Mallari, and J. Singh, “Edge Detection Using Gradients”, Ateneo de Manila University, Quezon City, Philippines