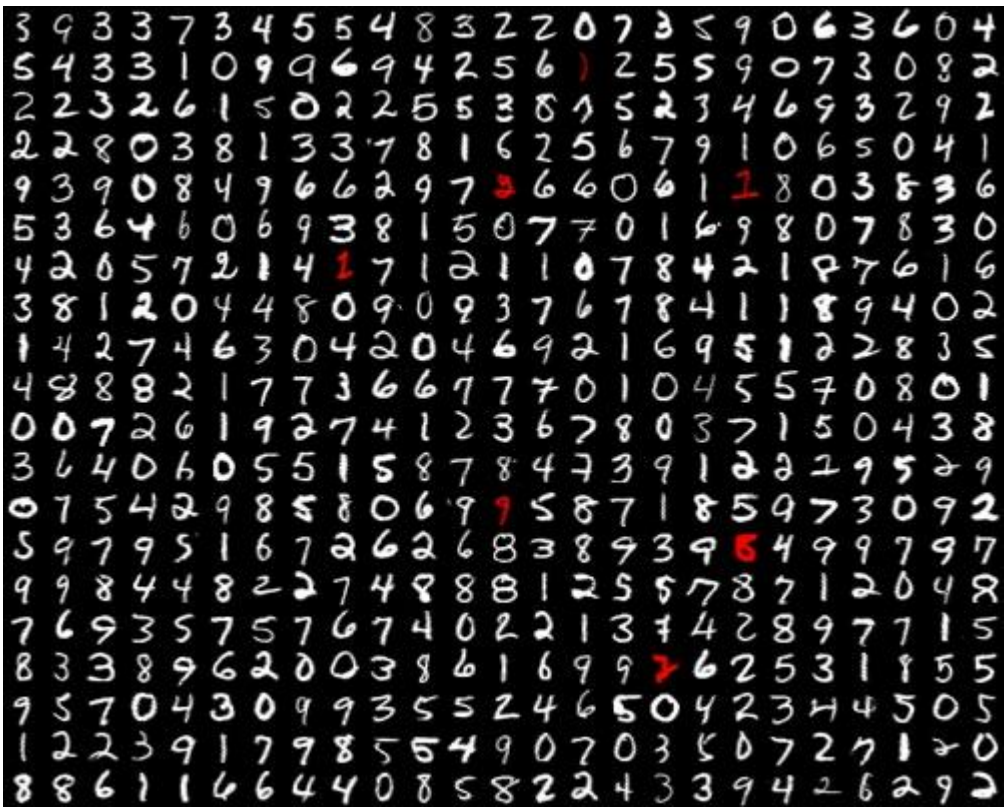

Handwritten Digit Recognition

Helani Jayawardhane

Machine Learning for Cyber Security

5/7/20

Table of Contents

Table of Figures

1. Handwritten Digit Recognition

1.1. Idea

After going through many sources through the internet the topic was selected to be “Handwritten Digit Recognition” and basically the idea was to come up with a web application that will have the ability to get a handwritten digit as a user input and determine what it is with the highest level of accuracy possible.

Therefore, the web application will contain of a sketching canvas where the user will be able to write a digit with the use of the cursor and the application will take that digit as an input and determine what digit it is.

1.2. Technologies used

The model has been created using Neural Network concept and the datasets were downloaded fro MNIST. The following technologies were used in order to implement the idea,

- Python 3
- PyCharm IDE
- Javascript
- HTML
- Jupyter Notebook
- Colaboratory – (Google Online platform)

1.3. Methodology

As shown in figure 2.1, the input taken from the user will be sent to an input layer with an input layer neuron count of 784 which will be one for each pixel of the 28 x 28 pixel image input. Which will then be sent to a hidden layer of arbitrary neuron count of 100 and the output layer will have

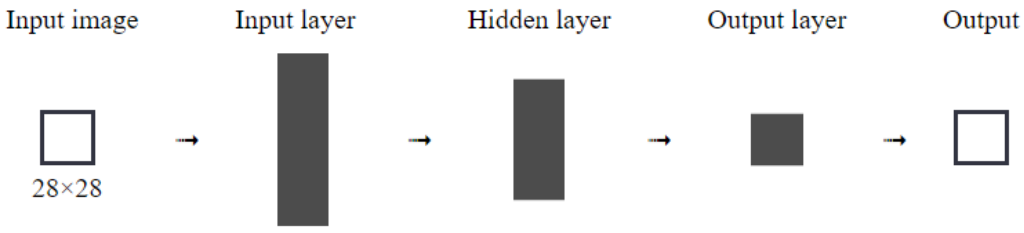

Figure 1.1: Neural network illustration

a neuron count of 10 which will be one for each number from 0 to 9.

1.3.1 Implementation steps

1.3.1.1. Training the Model using PyCharm IDE/ Colaboratory

1. Firstly, the necessary libraries will be needed to be imported

```
import numpy as np
import matplotlib.pyplot
from matplotlib.pyplot import imshow
import scipy.special as scipy
from PIL import Image
import codecs, json
```

Figure 1.2: Import libraries

2. Create and Neural Network object and set the values for input layer. hidden layer, output layer and the self-learning rate (the amount of change to the model during the process)

```
class NeuralNetwork(object):
    def __init__(self):
        self.input_neuron_count = 28*28
        self.hidden_neuron_count = 100
        self.output_neuron_count = 10
        self.learning_rate = 0.1
```

Figure 1.3: Set neuron counts for input, hidden and output layers

3. Create a generate_ random_weight_matrix function and then generate weights for input layer-hidden layer and hidden layer-output layer.

```
generate_random_weight_matrix = lambda input_neuron_count, output_neuron_count: (
    np.random.normal(0.0,  pow(input_neuron_count, -0.5), (input_neuron_count, output_neuron_count)) #center of
)

self.input_x_hidden_weights = generate_random_weight_matrix(self.input_neuron_count, self.hidden_neuron_count) #
self.hidden_x_output_weights = generate_random_weight_matrix(self.hidden_neuron_count, self.output_neuron_count)
```

Figure 1.4: Generate weight matrices

Input layer-hidden layer will take input neuron count of 784 and hidden neuron count of 100 and the hidden layer-output layer will take 100 as hidden neuron count and 10 as the output neuron count.

4. In this case the sigmoid function will be used as the activation function hence it will then be defined.

```
self.activation_function = lambda value: scipy.expit(value)
```

Figure 1.5: Define activation function

5. In order to train the model two arrays will be created as input and target with minimum dimension of 2

```
def train(self, input_array, target_array):
    inputs = np.array(input_array, ndmin=2) #create input array with minimum dimension of 2
    targets = np.array(target_array, ndmin=2) #create target array with minimum dimension of 2
```

Figure 1.6: create input and target arrays

6. Define the input for the hidden layer and the output of hidden layer.
   The input for hidden layer will be the multiplication of the input array and the weight for the hidden layer input and the output of the hidden layer will be the hidden layer input after activating the sigmoid function on it.

```
hidden_layer_input = np.dot(inputs, self.input_x_hidden_weights) #
hidden_layer_output = self.activation_function(hidden_layer_input)
```

Figure 1.7: Calculate hidden layer input and output

7. Define input for the output layer and the output of output layer.
   The input for output layer will be the multiplication of the hidden layer output array and the weight for the output layer input and the output of the output layer will be the output layer input after activating the sigmoid function on it.

```
output_layer_input = np.dot(hidden_layer_output, self.hidden_x_output_weights)
output_layer_output = self.activation_function(output_layer_input) #sigmoid fur
```

Figure 1.8: Calculate output layer input and output

8. The model will need corrections for both hidden layer output and output layer output. They corrections will be made as follows.

```
output_errors = targets - output_layer_output
self.hidden_x_output_weights += self.learning_rate * np.dot(hidden_layer_output.T, (output_errors * output_layer_output * (1 - output_laye
#

hidden_errors = np.dot(output_errors, self.hidden_x_output_weights.T)
self.input_x_hidden_weights += self.learning_rate * np.dot(inputs.T, (hidden_errors * hidden_layer_output * (1 - hidden_layer_output)))
```

Figure 1.9: Corrections for possible errors in hidden layer and output layers

9. Create function to retrieve the final output of output layer.
   Take the input and create a 2-dimensional array, send it through the input layer, hidden layer and output layer and finally retrieve the output.

```
def query(self, input_array):
    inputs = np.array(input_array, ndmin=2)

    hidden_layer_input = np.dot(inputs, self.input_x_hidden_weights)
    hidden_layer_output = self.activation_function(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_output, self.hidden_x_output_weights)
    output_layer_output = self.activation_function(output_layer_input)

    return output_layer_output #output layer output
```

Figure 1.10: Calculations to retrieve output of output layer

10. I converted the generated weights for the hidden layer to list and created json files to store them so that each time the code is run I wouldn't have to grab data from the server in the process of testing.

```
def export(self):
    input_x_hidden_weights = self.input_x_hidden_weights.tolist()
    json.dump(input_x_hidden_weights, codecs.open('input_x_hidden_weights.json', 'w', encoding='utf-8'), separators=(',', ':'), sort_keys=True, indent=4)

    hidden_x_output_weights = self.hidden_x_output_weights.tolist()
    json.dump(hidden_x_output_weights, codecs.open('hidden_x_output_weights.json', 'w', encoding='utf-8'), separators=(',', ':'), sort_keys=True, indent=4)
```

Figure 1.11: Write data to json files

11. Prepare an array to store the handwritten data and flatten the array into a 1-dimensional array will be done.

```
def prepare_data(handwritten_digit_array):
    return ((handwritten_digit_array / 255.0 * 0.99) + 0.0001).flatten()
```

Figure 1.12: Preparing the handwritten digit array

12. Create a target array and retrieve target.

```
def create_target(digit_target):
    target = np.zeros(10) + 0.01
    target[digit_target] = target[digit_target] + 0.98
    return target
```

Figure 1.13: Retrieving target value

13. Create function to get the index of the maximum of an array

```
def get_index_of_max(array):
    array = array[0]
    index = 0
    m = max(array)
    for n in array:
        if n == m:
            return index
        index = index + 1
```

Figure 1.14: Function to retrieve index of the greatest number predicated

14. To start the training process first a new neural network object will be created and the training data file “mnist_train_100.csv” will be read.

```
neural_network = NeuralNetwork()
training_data_file = open('mnist_train_100.csv', 'r')
training_data = training_data_file.readlines()
training_data_file.close()
```

Figure 1.15: Read training data file

15. Then to train the data, first separate data in the training dataset and assign to handwritten_digit_raw variable. Then the data will be assigned to an array and all elements starting from 1 will be reshaped in to 28 x 28 because the model is being trained to take inputs of 28 x 28 pixel images. The variable handwritten_digit_target will hold the integer values of the handwritten_digit_raw array holds. Start training the after preparing the two arrays.

```
for data in training_data:
    handwritten_digit_raw = data.split(',')
    handwritten_digit_array = np.asfarray(handwritten_digit_raw[1:]).reshape((28, 28))
    handwritten_digit_target = int(handwritten_digit_raw[0])
    neural_network.train(prepare_data(handwritten_digit_array), create_target(handwritten_digit_target))
```

Figure 1.16: Prepare training data

16. To start the training process first a new neural network object will be created and the training data file “mnist_test_10.csv” will be read.

```
test_data_file = open('mnist_test_10.csv', 'r')
test_data = test_data_file.readlines()
test_data_file.close()
```

Figure 1.17: Read testing data file

17. Then to test the data, first separate data in the test dataset and assign to handwritten_digit_raw variable. Then the data will be assigned to an array and all elements starting from 1 will be reshaped in to 28 x 28 because the model is being tested to take inputs of 28 x 28 pixel images and retrieve output by using the query function and flatten the handwritten_digit_array in to a 1-dimensional array.

```
for data in test_data:

    handwritten_digit_raw = data.split(',')
    handwritten_digit_array = np.asfarray(handwritten_digit_raw[1:]).reshape((28, 28))
    handwritten_digit_target = int(handwritten_digit_raw[0])
    output = neural_network.query(handwritten_digit_array.flatten())
```

Figure 1.18: Prepare testing data

18. Initialize the test_count, hit_count and test_count to 0. Then hit_or_miss will grab the index of the max index of the array “output”. If the hint_or_miss equals to the target, then the hit_count will be 1. And else the miss_count will be 1. Oncw the test is completed the test count will increment by 1.

```
test_count = 0
hit_count = 0
miss_count =0

for data in test_data:

    handwritten_digit_raw = data.split(',')
    handwritten_digit_array = np.asfarray(handwritten_digit_raw[1:]).reshape((28, 28))
    handwritten_digit_target = int(handwritten_digit_raw[0])
    output = neural_network.query(handwritten_digit_array.flatten())

    hit_or_miss = get_index_of_max(output)
    if hit_or_miss == handwritten_digit_target:
        hit_count = hit_count + 1
    else:
        miss_count = miss_count + 1
    test_count = test_count + 1

print("test_count", test_count)
print("hit_count", hit_count)
print("miss_count", miss_count)
print("hit ratio", hit_count/test_count * 100)

neural_network.export();
```

Figure 1.19: hit and miss counts

19. In here I have taken the image with the index of 8 from the test dataset to test it out. Then open the test data file, read it and fetch the image with index 8. Split the handwritten_digit_raw array then reshape it into a 28x28 array and assign it to handwritten_digit_array. Get the integer values of the handwritten_digit_array and assign to target array named handwritten_digit_target. Once this has been done then use the “query” function to do the neural network operation upon the prepared flattened array. Finally get the value that the maximum index holds.

```
test_image_index = 8
test_data_file = open('mnist_test_10.csv', 'r')
test_data = test_data_file.readlines()
test_data_file.close()
data = test_data[test_image_index]
handwritten_digit_raw = data.split(',')
handwritten_digit_array = np.asfarray(handwritten_digit_raw[1:]).reshape((28, 28))
handwritten_digit_target = int(handwritten_digit_raw[0])
matplotlib.pyplot.imshow(handwritten_digit_array, cmap='Greys', interpolation='None')
output = neural_network.query(handwritten_digit_array.flatten())
print(output)
print(get_index_of_max(output))
```

Figure 1.20: Reshape and get output

1.3.1.2. Creating Web Application using PyCharm IDE, JavaScript and HTML

Once the training was successful it was needed to then be implemented as a web application therefore, the following steps were used to build this as a web application.

I have created 3 JavaScript files in order to achieve the goal of implementation.

1. neuralNetwork.js

   This file will handle the creation and functions of the neural network object.

   Firstly, I have defined the sigmoid function which I used as the activation function when training the model. Then I created two input and output weight arrays for the hidden layer and assigned the weight matrices created during the training process.

```
function NeuralNetwork(input_x_hidden_weights, hidden_x_output_weights){
    this.activationFunction = function sigmoid(x) {
        return 1 / (1 + Math.exp(-x));
    }
    this.input_x_hidden_weights = math.matrix(input_x_hidden_weights);
    this.hidden_x_output_weights = math.matrix(hidden_x_output_weights);
}
```

Figure 1.21: Define activation function and weights

Then I proceeded with creating the input array in order to create hidden layer input by multiplying the weights and input array and activating the sigmoid function which will then result in the hidden layer output which will act as the input for the output layer.

```
NeuralNetwork.prototype.query = function(input_array){
    inputs = math.matrix(input_array);

    hidden_layer_input = math.multiply(inputs, self.input_x_hidden_weights);
    hidden_layer_output = hidden_layer_input.map(x => this.activationFunction(x));
```

Figure 1.22: Define inputs and outputs of hidden layer

By taking the output of hidden layer and the weight matrix generated for the output layer I then created the input for output layer by multiplying the two arrays and then by activating the sigmoid function on it I was able to get the output layer output value which I have taken as the return value from the function.

```
NeuralNetwork.prototype.query = function(input_array){
    inputs = math.matrix(input_array);

    hidden_layer_input = math.multiply(inputs, self.input_x_hidden_weights);
    hidden_layer_output = hidden_layer_input.map(x => this.activationFunction(x));

    output_layer_input = math.multiply(hidden_layer_output, self.hidden_x_output_weights);
    output_layer_output = output_layer_input.map(x => this.activationFunction(x));

    return output_layer_output; //return value
}
```

Figure 1.23: Define output layer inputs and outputs and retrieve output

2. outputChart.js

This file was created with the objective of depicting the maximum output value which also in this case will be the determined/ recognized value by the trained model. The chart will be a bar chart with labels from 0 to 9 (which will be the input values).

```
function OutputChart(){
    this.ctx = document.getElementById("outputChart");
    this.ctx.width = 650;
    this.ctx.height=250;
    this.chart = new Chart(this.ctx, {
        type: 'bar',
        data: {
            labels: ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"],
            scaleShowLabels : false,
            datasets: [{
                label: 'Match rate',
                data: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                borderWidth: 1
            }]
        },
        options: {
            responsive: false,
            legend: {
                display: false
            },
            scales: {
                yAxes: [{
                    ticks: {
                        beginAtZero:true,
                        max: 1.0
                    }
                }]
            }
        }
    });
```

Figure 1.24: Chart creation

The chart will get the index value of the maximum output value and depict the value it holds.

```
OutputChart.prototype.plot = function(output){
    var resetBgColor = '0'.repeat(10).split('').map(x => 'rgba(59, 79, 94, 0.6)');
    var resetBorder = '0'.repeat(10).split('').map(x => 'rgba(77, 82, 110, 0.6)');
    var maxIndex = output.indexOf(Math.max(...output));
    this.chart.data.datasets[0].backgroundColor = resetBgColor;
    this.chart.data.datasets[0].borderColor = resetBorder;
    this.chart.data.datasets[0].data = output;
    this.chart.data.datasets[0].backgroundColor[maxIndex] = 'rgb(208, 79, 94)';
    this.chart.data.datasets[0].borderColor[maxIndex] = 'rgba(219, 181, 36, 0.5)';

    this.chart.update();
```

Figure 1.25: Depict maximum value

Once the clear button is clicked on the chart will reset the values to all zeros.

```
OutputChart.prototype.clear = function(){
    this.chart.data.datasets[0].data = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
    this.chart.update();
}
```

Figure 1.26: Reset upon clearance

3. main.js

This file was created to handle the main functionalities of the web application such as taking user input and depicting the value recognized by the trained model.

New objects of neural networks and output charts were created as the first step to commence with the implementation.

```
let neuralNetwork = new NeuralNetwork(input_x_hidden_weights, hidden_x_output_weights);
let outputChart  = new OutputChart();
```

Figure 1.27: Create objects

Once that was done then, the drawing canvas was created in order to get the user input and for that a 2-dimensional canvas with a height of 84 and width of 84 was created.

```
let drawingCanvas =  document.getElementById('input-canvas');
drawingCanvas.width= 84;
drawingCanvas.height= 84;
let drawingContext = drawingCanvas.getContext('2d');
```

Figure 1.28: Create drawing canvas to get user input

Then the feed canvas, a 2-dimensional canvas with a height of 28 and width of 28 was created to feed the digit given as the input.

```
let feedCanvas = document.getElementById('feed-image');
feedCanvas.width= 28;
feedCanvas.height= 28;
let feedContext = feedCanvas.getContext('2d');
```

Figure 1.29: Create feed canvas to feed image taken as input

Then a function to get the mouse position from drwaing canvas object was written and and the in order to convert the client window relative mouse position ‘x’ and ‘y’ to the canvas element the following substractions were done.

```
function getMousePos(evt) {
    var rect = drawingCanvas.getBoundingClientRect();
    return {
        x: evt.clientX - rect.left,
        y: evt.clientY - rect.top
    }
```

Figure 1.30: Get mouse position and resize

To detect mouse movement on the canvas the following were written.

```
var mouseIsDown = false;
drawingCanvas.onmousedown =  ev => {
    mouseIsDown = true;
}
drawingCanvas.onmouseup =  ev => {
    mouseIsDown = false;
}
```

Figure 1.31: Detect mouse movement

On the event were mouse movement is detected the following was written to get the mouse position x, y and set radius, start angle and end angle.

```
drawingCanvas.onmousemove = e => {
    if(!mouseIsDown) return;

    drawingContext.beginPath();
    drawingContext.arc(getMousePos(e).x, getMousePos(e).y, 3, 0, 2 * Math.PI, false);
    drawingContext.fillStyle = '#000000';
    drawingContext.fill();

    return false;
}
```

Figure 1.32: Detect mouse movement

Once the user had drawn the digit, has been fed to the canvas and upon clicking on the “recognize” button the handwritten image will be positioned on the canvas specifying width and height of 0,0. And it will feed the positioned image as data values to the array.
An array called pixel will be initiated and the “query” function where the neural network will initiate process of creating input for hidden layer to getting output from output layer will

kickstart. And the chart will be plotted accordingly to depict the index of maximum output value's value. Then the greatest value from the "output" variable will be extracted and assigned to the inner text of the of the "output-block" element which shows the value that was recognized.

```
document.getElementById('recognise-button').onclick = function(){
    feedContext.drawImage(drawingCanvas, 0, 0, feedCanvas.width, feedCanvas.height);
    let rgbaArray = feedContext.getImageData(0, 0, feedCanvas.width, feedCanvas.height).data;
    let pixels = [];
    for(let i = 3; i<rgbaArray.length; i = i + 4)
        pixels.push(rgbaArray[i]);
    let output = neuralNetwork.query(pixels)._data;
    outputChart.plot(output)
    document.getElementById('output-block').innerText = output.indexOf(Math.max(...output));
    document.getElementById('output-block').className += ' has-output';
};
```

Figure 1.33: Show output

Upon clicking the "clear" button, the coordinates of the drawing canvas and feed canvas will be reset to 0,0 and the output block which depicts the recognized output value will be cleared.

```
document.getElementById('clear-button').onclick = function(){
    outputChart.clear();
    drawingContext.clearRect(0, 0, drawingCanvas.width, drawingCanvas.height);
    feedContext.clearRect(0, 0, feedCanvas.width, feedCanvas.height);
    document.getElementById('output-block').innerText = '';
    document.getElementById('output-block').classList.remove('has-output');
};
```

Figure 1.34: Clear canvas

The next step was to create a index.html page for the user to access the application this was done as shown in the below figure 1.35.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Handwritten Digit Recognition</title>

    <link rel="stylesheet" href="./styles/normalize.css">
    <link rel="stylesheet" href="./styles/btns.css">
    <link rel="stylesheet" href="./styles/main.css">
</head>
<body>
    <div class="content">
       <h1 style="text-align:center">Handwritten Digit Recognition</h1>
        <div class="input-canvas-container">
            <span class="label">Draw any single digit from 0 to 9. (center the drawing inside canvas)</span>
            <canvas id="input-canvas"></canvas>
            <div class="controls">
                <a href="#" class="btn btn--s btn--gray-dark" id="recognise-button">Recognise</a>
                <a href="#" class="btn btn--s btn--gray-dark" id="clear-button">Clear</a>
            </div>
        </div>

        <div class="feed-image-container">
            <span class="label"> Neural network illustration</span>
            <div class="layers-container">
                <div class="feed-image-wrapper">
                    <span class="title">Input image </span>
                    </svg>
                    <span class="dimensions">784 neurons</span>
                </div>
                <span class="direction-arrow">→</span>
                <div class="layer">
                    <span class="title">Hidden layer</span>
                    <svg width="30" height="100" class="layer-block">
                        <rect y="15" width="30" height="70" style="fill:rgb(91, 145, 189)" />
                    </svg>
                    <span class="dimensions">100 neurons</span>
                </div>
                <span class="direction-arrow">→</span>
                <div class="layer">
                    <span class="title">Output layer</span>
                    <svg width="30" height="100" class="layer-block">
                        <rect y="35" width="30" height="30" style="fill:rgb(91, 203, 79)" />
                    </svg>
                    <span class="dimensions">10 neurons</span>
                </div>
                <span class="direction-arrow">→</span>
                <div class="output">
                    <span class="title">Output</span>
                    <div class="output-block" id="output-block"></div>
                </div>
            </div>

        </div>
        <div class="output-chart-container">
            <span class="label">Match rate</span>
            <canvas id="outputChart"></canvas>
```

Figure 1.35: index. html

1.3.1.3. Final Output

Once web application was hosted on local host and the handwritten digit was given as ‘1’ to the input canvas the application recognizes it as ‘1’ itself as shown in figure 1.36.

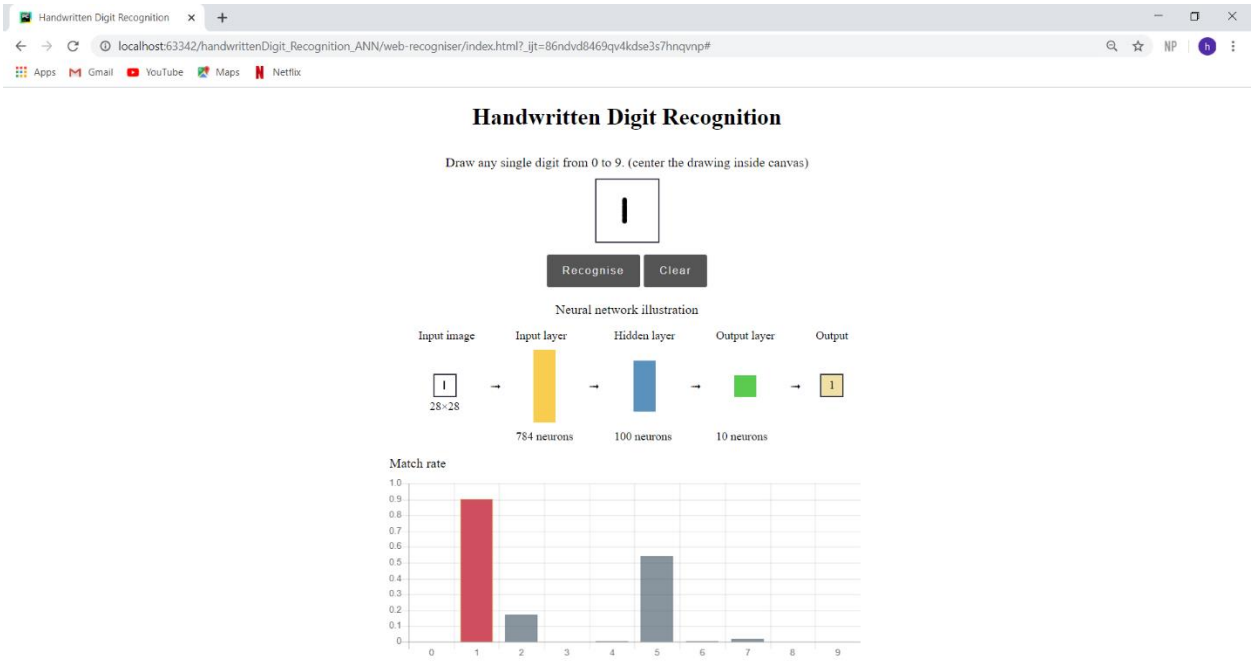

Figure 1.36: Final output

References

[1] M. Jelaska, "GitHub," [Online]. Available: https://github.com/MiroslavJelaska/handwritten-digit-recognition. [Accessed 16 04 2020].

[2] "w3Schools," [Online]. Available: https://www.w3schools.com/.

[3] "parsers," 20 05 2019. [Online]. Available: https://parsers.me/deep-learning-machine-learning-whats-the-difference/.

[4] "Stack Overflow," [Online]. Available: https://stackoverflow.com.

[5] M. Hargrave, "Investopedia," 30 04 2019. [Online]. Available: https://www.investopedia.com/terms/d/deep-learning.asp.

[6] "Numpy," [Online]. Available: https://numpy.org/devdocs/.