SEMINAR REPORT
On

DIGIT RECOGNITION

Submitted by

ARUNAV SUTAR
(**REGD NO**.: -1901110047)

in partial fulfilment for the award of the

degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND
ENGINEERING

GOVERNMENT COLLEGE OF ENGINEERING, KALAHANDI

MARCH 2023

CERTIFICATE

Certified that this minor project report **"DIGIT RECOGNITION"** is the bonafide work of "**ARUNAV SUTAR"** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Signature Of supervisor

Dr. Somya Das
Department Of Computer
Science Engineering

Head Of the Department

Dr. Basanta Kumar Swain
Department Of Computer
Science Engineering

ACKNOWLEDGEMENT

The satisfaction that successful completion of this project would be incomplete without the mention of the people who made it possible, without whose constant guidance and encouragement would have made effort go in vain. I consider myself privileged to express gratitude and respect towards all those who guided us through the completion of this project. I convey thanks to my guide **Dr. Somya Das** mam for providing encouragement, constant support and guidance which was of great help to complete this project successfully. I am very grateful to **Dr. Basanta Kumar Swain** sir Head of the Department of Computer Sc. and Engineering for giving support and encouragement that was necessary for the completion of this project.

I would also like to express my gratitude to **Dr. Dulu Patnaik** Principal, Government College of Engineering Kalahandi, Bhawanipatna for providing us a congenial environment to work in.

Arunav Sutar

(1901110047)

ABSTRACT

The basic fundamentals of this Digit recognition project is image processing. In here I process some pre-verified images of digits of size 28*28 pixel with their labels. And after the model is trained then the model can verify any new image of 28*28-pixel digit image. The dataset I collected is the MNIST hand written image dataset. This dataset contains a lot of images of hand written digits of size 28*28 pixel. Which I am going to use to train my model.

Here we can use so many classification algorithms to train the model. One model is logistic regression. But we cannot use it because our output is classified into 10 different labels. But logistic regression can only classify up to two different labels. Logistic regression algorithm will form a linear equation line which will divide the examples into two sub group. So, we have to choose a different classification algorithm.

Here we can use deep neural network to structure this problem. Using neural network, we can make a non-linear function which will classify the output into 10 different labels. We can tune the hyperparameters of the neural network to get the optimal result. Using neural network, we can make complex function to optimally divide the output into proper groups.

CONTENT

List of Figures

Chapter-1
INTRODUCTION

1.1 Basics of Neural network

In Machine learning, Artificial Neural Networks (ANN) play a major role in showcasing the power of statistics and mathematics to solve complex and non-linear problems. ANN has been there in the field of machine learning and Artificial Intelligence for a long time, but recent improvements in computational power and big data is helping them to show how powerful they are. They can be used to solve both supervised and unsupervised, classification and regression problems, computer vision etc. Here we shall be implementing an ANN from scratch and apply it to solve a simple problem of detecting digits from 0–9. Neural Network is similar to logistic regression (perceptron) but with more layers and hidden units. Here's is what a single layer perceptron (logistic regression) looks like.

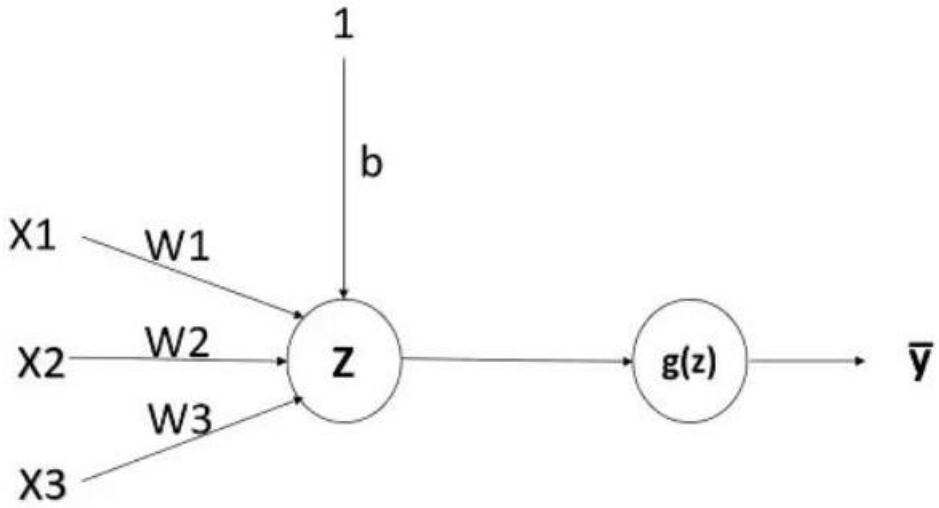

Fig.1.1 A single Neuron[1]

The input layer is where the features of the input features (X) is fed, the Hidden layer will be the weighted sum of the inputs with parameters (W) followed by an activation function. There will always be bias (b) for each hidden layer. The output of the neural network will be the weighted sum of outputs of previous hidden layer followed by an activation function. For a two-class classifier problem, output layer will be the probability that given training example belongs to the layer (l_i).

$Z_1 = W_1 . X + b_1$, where Z_1 is the weighted sum of inputs and b_1 is the bias. X is the input matrix where each training example is stacked horizontally via columns. So dimensions of X are (N_x, m) where N_x is the number of features, m is the number of training examples. If the number of units in the hidden layer is N_1, then dimensions of W_1 will be (N_1, N_x). So dimensions of Z_1 will be (N_1, m). b_1 is a scalar quantity and when we add it in python, it will be added by broadcasting rules. The output of the hidden layer is the result of passing Z_1 to the activation function. Similarly for an N layer neural network, at any hidden layer, dimensions of W_n will be $(N_n, Nn -1)$. Similarly, our problem is to classify the digits. And we are going to use a neural network. And there are so many non-linear activation function. And different function gives us different result. The most choice activation functions are sigmoid function, Relu function, tanh function, SoftMax function, Leaky Relu activation function.

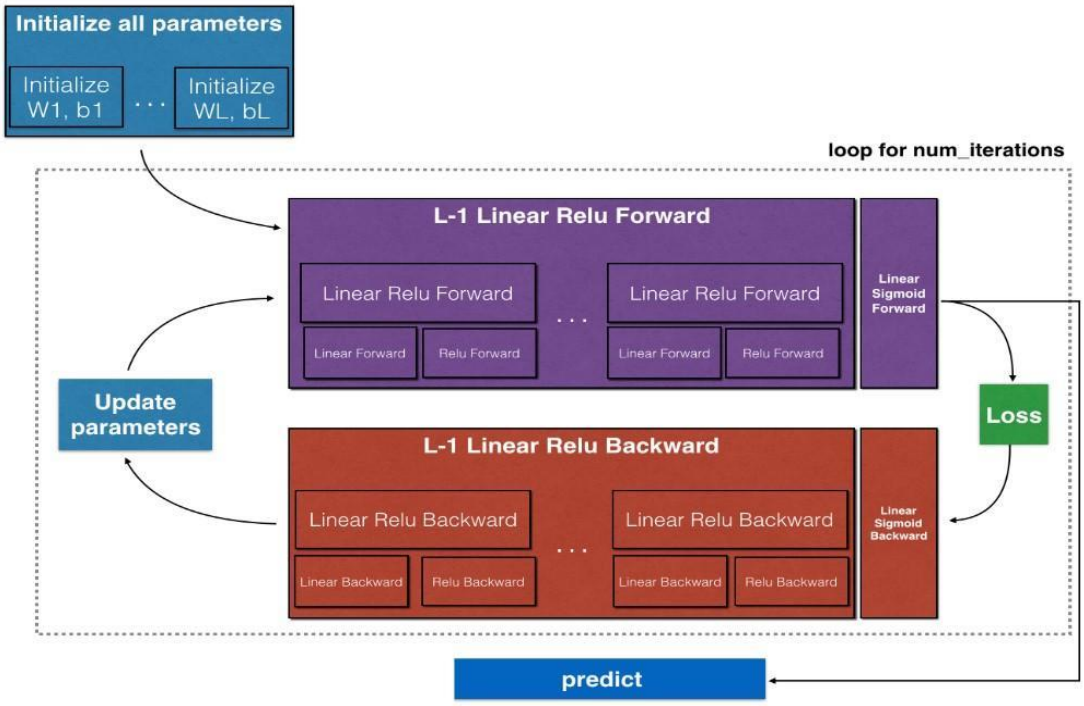

Fig.1.2 Data Flow Diagram[1]

After that in backward propagation, we are going to optimise the weights assigned to the equation using the gradient descent method. The gradient descent method is most often used to optimise the weights and the bias term.

CHAPTER 2

Problem formulation

2.1 Dataset Understanding [1][2][4]

The dataset we are using is the MNIST handwritten digit dataset. Which contains some thousands of images and their labels. The size of each image will be 28*28 pixel. But those are in image format. We have to covert them into NumPy array format in which the image array will contain pixel value(0-255). And the out put array is of only values determining the result i.e. 0-9. Now from this data set we have to separate some examples for testing purpose and remaining is used for training purpose. And some pre-processing have to be done before the model training because noisy data can not be properly trained by the algorithm.

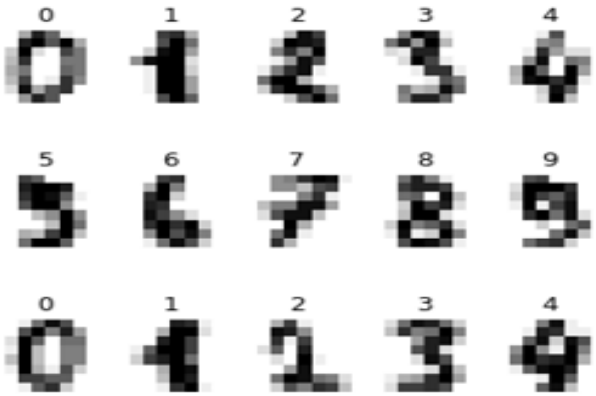

Fig.2.1 Input image examples [3]

2.2 Input image pre-processing

After we got the images in array format. The single image must be in 2-dimensional plane because we are using a grey scale image. Then we must do the normalization of the value, there we scale the pixel values. Here we are using min-max scaling to bring the scale in to (0-1). But scaling does not change the actual information of the data. It just brings the value into small scale.

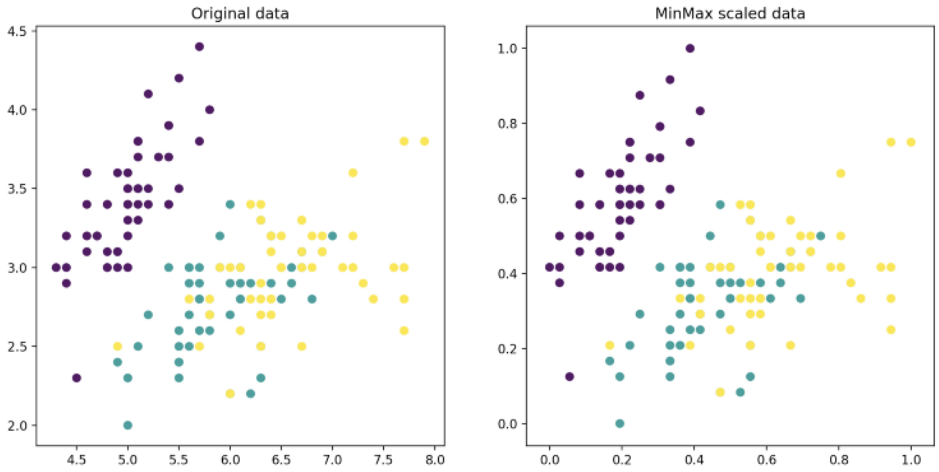

Fig. 2.2 Min Max Scaling [4]

And the need of Scaling is to easy fitting of the data into the algorithm. Let take a example the inputs value ranges between (0-255) but the output value is ranges between (0-1)(we will see this in the next part).So to reduce the gap between the input and the output we have to do the normalization. After that we have to convert the 2-d array into linear because we are going to process all pixel values. Se we are going to make it linear. And the dimension should be of the form $(N_x,1)$. Where Nx is the no. of pixels present in the array i.e. 784(28*28). And for m image examples the size should be (N_x,m).

2.3 Output label pre-processing

As the output is a linear array which contains the value between (0-9) of the form (1, m). But like this we cannot compare the given output and the algorithm output. So, we have to do some transformation such that we can easily compare between the found output and the actual output. For that we are using One Hot-Encoding using this we are making a 2d array of 10 columns (0-9) and m rows (m=no. of examples) (m,10) where and the conversion is like this such as if the output is 5 then the 5^{th} column value is 1 and others are 0 as shown below(Fig 5)

Fig. 2.3 One hot Encoding

CHAPTER 3

Proposed solution

3.1 *Neural Network Implementation* [1][2]

There are so many steps involved in this algorithm.

3.1.1 Initialize parameters

3.1.2 Forward propagation

3.1.3 Cost measurement

3.1.4 Backward propagation

3.1.5 Updating the parameters

In this problem we are going to implement a 3 Layer Neural network in which one is input layer two are hidden layers and last one is the output layer.

In this problem we are using only one hidden layer. Which contains 20 neurons in it .

And the output layer is of 10 neurons. The 10 output neurons represent values for 10 different digits. And the input layer consists of 784 neurons as it is the input layer. Form now we will see how these steps are going to be executed.

3.2 Initialization of Parameters

It includes all W's and b's for every hidden layer. And all W's have to initialize randomly otherwise if we initialize 0 to all the W then for some steps in neural network it will contain same step. So we have to initialize the W's randomly. B's term will doesn't matter. For a layer l (from 1 to L) the shape of the weight for that layer will be (no. of neurons on layer l , no. of neurons in layer l-1). And then initialize it with random values.

3.3 Forward Propagation

In forward propagation we operate 2 operations. One is linear forward propagation and other one is non linear forward propagation using any non-linear function. These 2 steps are followed by every layer.

First we calculate the linear function i.e.

$$Z_l = W_l . A_{l-1} + b_l \tag{3.1}$$

Then we can use any non-linear function to calculate the output of the layer. Non-linear functions are used to form a complex equation of the structure. For this case

we are going to use the relu function for the hidden layer. And we will use the SoftMax function for the output layer. Because SoftMax function finds out the probability of getting the output as 1.

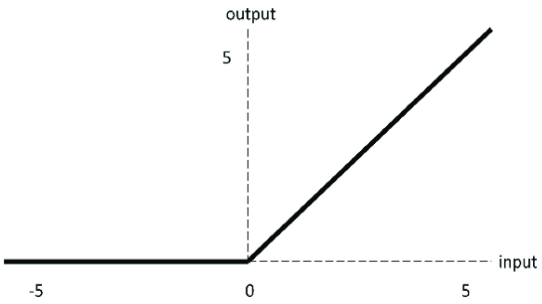

Fig. 3.1 Relu[7]

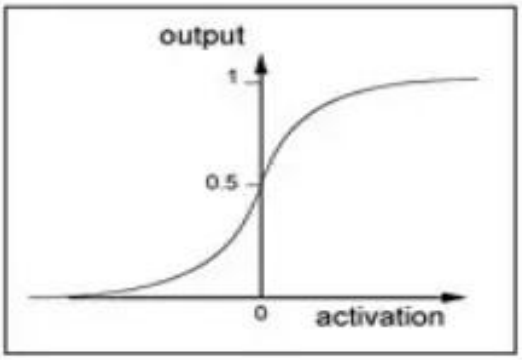

Fig. 3.2 SoftMax[6]

$$\textbf{Relu } (A) = \max(0, x) \tag{3.2}$$

$$\textbf{SoftMax } \quad \sigma(Z) = \frac{e^{z}}{\sum_{i=1}^{m} e^{z}} \tag{3.3}$$

3.4 Cost Measurement

From the last step we get the measured output named Ycap. Which is same as A from the output layer. This Ycap is the calculated from the random weights and the bias term. And Ycap is not so similar to Y i.e. the original output. So We have to find the difference which is also known as loss . So the loss function used here for the Softmax function is :

$$\textbf{loss} = (-1/m) \sum_{i=1}^{m} (Y * \log(Ycap) + (1 - Y) * \log(1 - Ycap)) \tag{3.4}$$

So our main goal is to reduce this loss such that the corresponding weights are the optimal one.

3.5 Backward Propagation

For this step we are going to use gradient descent to get the optimal weights. So using the partial derivative of the loss function from the previous step. Using chain rule we are going

to find the partial derivative of loss function with respect to the weights and the bias term and find the term to reduce from the original W and b.

$$\mathbf{dw^{[l]} = \frac{1}{m} dz^{[l]} A^{[l-1]T}} \quad (3.5)$$

$$\mathbf{db^{[l]} = \frac{1}{m} \sum_{i=1}^{m} dz^{[l](i)}} \quad (3.6)$$

$$\mathbf{dA^{[l-1]} = w^{[l]T} dz^{[l]}} \quad (3.7)$$

These partial derivatives are calculated from dZ.

And we have to find dZ from the derivative of the activation function.

All this are done using chain rule.

Derivative of non-linear function:-

For softmax function :- $\mathbf{f'(X) = f(x)(1 - f(x))}$ (3.8)

For relu function :- $f'(x) = \begin{cases} 0 & \text{if } x<0, \\ 1 & \text{if } x>0 \end{cases}$ (3.9)

Using these functions, we have to find the partial derivative of dZ. At last we will find the dW1, db1, dW2, db2.

3.6 Parameters Updation

The terms that we found in the previous step from the partial derivative, a small part of that we have to reduce from the original W and b such that we go to a optimal direction. The part of the term is calculated by another term that is learning-rate. Which is multiplied with the term and subtracted from the original term.

The Update equations are: -

W1=W1-learning_rate*dW1 (3.10)

W2=W2-learning_rate*dW2 (3.11)

b1=b1-learning_rate*db1 (3.12)

b2=b2-learning_rate*db2 (3.13)

This all steps will be looping for a particular number of iterations.

CHAPTER 4

Implementation Results

4.1 Libraries Used [5]

For this Project there are so many libraries are used.

4.1.1 OpenCV

The OpenCV (Open Source Computer Vision Library) is a python DIL library which is very updated in the work of Image processing .One image file and pixel values can easily come into surface by this library. This library provides a common infrastructure and module related to computer vision technologies. The most important thing about this tool is it is totally free and can be easily modified and changed respective to input by the programmer.

4.1.2 NumPy

NumPy is a library for the python programming language, adding support for large multidimensional array and metrices. This package contains a large collection of high level mathematical functions to operate on those arrays. NumPy is open source and has many contributors.

4.1.3 TensorFlow

TensorFlow is an end-to-end open source package in python for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

4.1.4 PIL

PIL is a library of advanced image tools having full name Pillow. It is used for noise cancellation and to draw modified pixels by noise. It is useful to Image crop and various subjectable techniques.

4.1.5 Streamlit

Streamlit is a library of web development provided to the data science developers to quickly deploy their model using this library. It provides easy implementation of the web development features. And in a very easy manner we can easily deploy it in the server also.

4.2 Result[4][6][8]

This model uses 3 layer structure from where the input layer is of 784 input neurons for all the pixels of a image, then for the1st hidden layer is of 128neurons and the 2nd hidden layer consists of 32 neurons and at last the output layer consists of 10 neurons. So the model summery looks like this.

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 784)               0

 dense_5 (Dense)             (None, 128)               100480

 dense_6 (Dense)             (None, 32)                4128

 dense_7 (Dense)             (None, 10)                330

=================================================================
Total params: 104,938
Trainable params: 104,938
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4.1 Model Summery

And then the model shows 97%accuracy in this model. And it looks like this:-

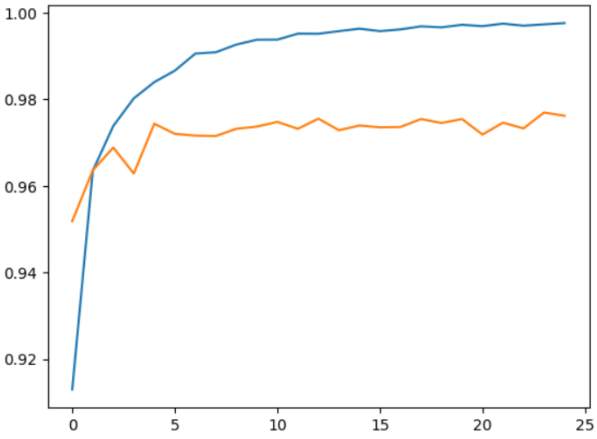

Fig. 4.2 Accuracy of the model

It correctly detects the unseen data after the training of the model.And when this model predict the untrained data from the test dataset it looks something like this:-

```
In [105]: ind=990
          plt.imshow(x_test[ind])
          print("Labeled Image:-"+str(y_test[ind]))
          print("Recognised Image:- "+str(model.predict(x_test[ind].reshape(1,28,28)).argmax()))
          #model.predict(x_test[ind].reshape(1,28,28)).argmax()
```

```
Labeled Image:-2
1/1 [==============================] - 0s 33ms/step
Recognised Image:- 2
```

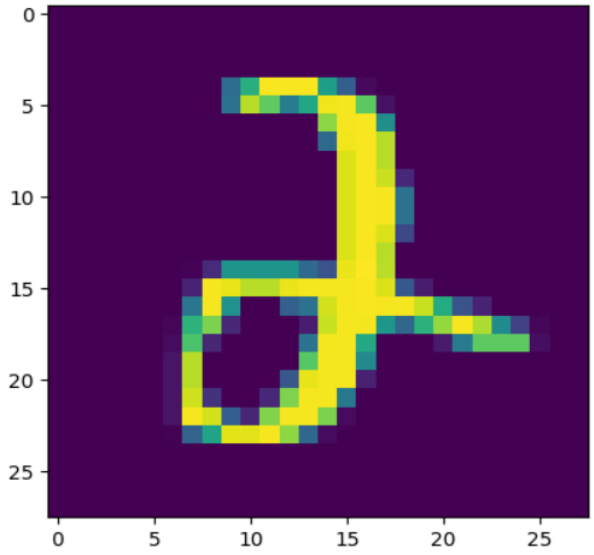

Fig. 4.3 Recognition to untrained image

It is a simple user interface which is going to take input image of the digit from the user and recognise the input digit. And the user interface of the project which is done using streamlit looks something like this:-

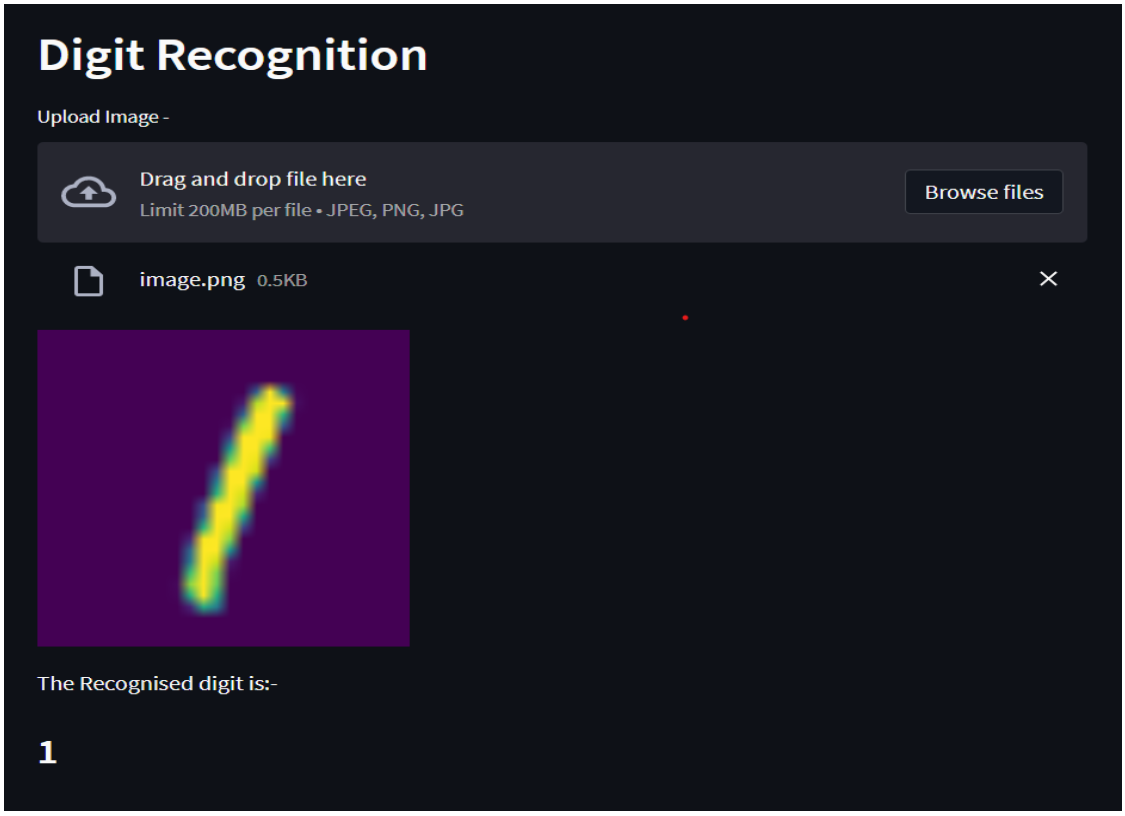

Fig. 4.2 User Interface

The whole implementation of the project is like the described above.

CHAPTER 5

Discussion

After that when the loss will be very low or unchanging for some iterations, then the corresponding W and b are the optimal coefficient of the equation. And those are the results of the algorithm. That means W and b are those resultant coefficients of the equation. Any new image of size 28*28 pixel is given to the algorithms then with the combination or forward propagation of with resulting weight and bias it will provide the high probability of that digit and low probability of others. Like this it will measure new images.

If we are going to integrate this with some web application, we only have picked this resultant W and b and integrate these two terms in the web application when some new image arrives then we have to pre-process that and then pass through this algorithm. And then we will get our corresponding result.

CHAPTER 6

Conclusion

At last I want to conclude that if we want to get the optimized algorithm then we have to run the algorithms for different hyper-parameters. To get the high accuracy we have to do the hyper-parameter tunning and select the proper hyperparameter. We can use different layers. And we use different no of neurons in a layer or we can change the learning rate. And if the algorithm shows high variance, then we can use Regularization. Or we can reduce the complexity of the neural network structure.

REFERENCES

[1] Andrew NG , Deep learning specialization course , Coursera

[2] Samson Zhang (2020), youtube video

[3] Pavan kalyan urandur (2019), Medium blogpost

[4] Deep Learning From Scratch: Building with Python from First Principles by Seth

[5] Deep learning in Python/ Pytorch by Manning Publications

[6] Grokking Deep Learning by Andrew W. Trask published by Manning Publications

[7] https://www.researchgate.net/figure/ReLU-activation-function_fig7_333411007

[8] Hands-on machine learning with Scikit-learn Keras and TensorFlow by Aurelion Geron