

HANDWRITING RECOGNITION AND IMAGE PROCESSING USING NEURAL NETWORKS

Report

Submitted in fulfilment of the requirements of

F376 Design Project

By

AISHWARYA SAIRAMA



**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
DUBAI CAMPUS**

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE No.
I.	Acknowledgement	2
II.	Certificate	3
III.	List of Figures	4
IV.	Abstract	5
V.	Table of Contents	6
VI.	Section	
1	Introduction	
1.1	Introduction.....	8
1.2	Objective.....	8
1.3	Approach.....	8
2	Background	
2.1	Line Segmentation.....	10
2.2	Character Segmentation.....	11
2.3	Pre – processing.....	11-12
3	Design and Implementation	
3.1	Convolutional Neural Network.....	13
3.2	Back-Propagation.....	13
3.3	Multi-Layered Perceptron.....	13-14
4	Code for Digit Recognition using CNN	15-16
5	Results and Explanation of Digit Recognition	17-18

6	Code for Handwriting Text Recognition using Neural Network	19-22
7	Results Of Handwriting Text Recognition	23-24
8	Conclusion	25
VII.	References	26

1 INTRODUCTION

1.1 Introduction

Handwriting Character Recognition (HCR) has the ability to receive an image from the computer and interpret handwritten inputs from sources such as documents, images, touch screen and many other devices. I will be building a character recognition system by using Python in this project. There are many other applications that include storing written content digitally, to convert digital signature etc. It also handles formatting, segmentations into characters. This project will be concentrating on making a character recognition system by employing neural networks in Python.

1.2 Objective

The aim of this project is:

- The project aims to identify handwritten characters with the use of neural networks.
- We need to build a suitable neural network and train it properly.
- This project aims to develop such a system which takes an image as an input and extract characters which could be an alphabets/digits. In this project I am considering the Image to be handwritten only.

1.3 Approach

The approach used for my design project, handwritten character recognition of classification, I used Python and Neural Network

- Steps in handwriting recognition are as follows:
 - Line segmentations
 - Character segmentations
 - Pre-processing of the image to remove noise
 - Feature extraction
 - Creating an Artificial Neural Network

- Training & Testing of the network
- Recognition

FLOW CHART THAT SHOWS GENERAL STEPS OF OCR

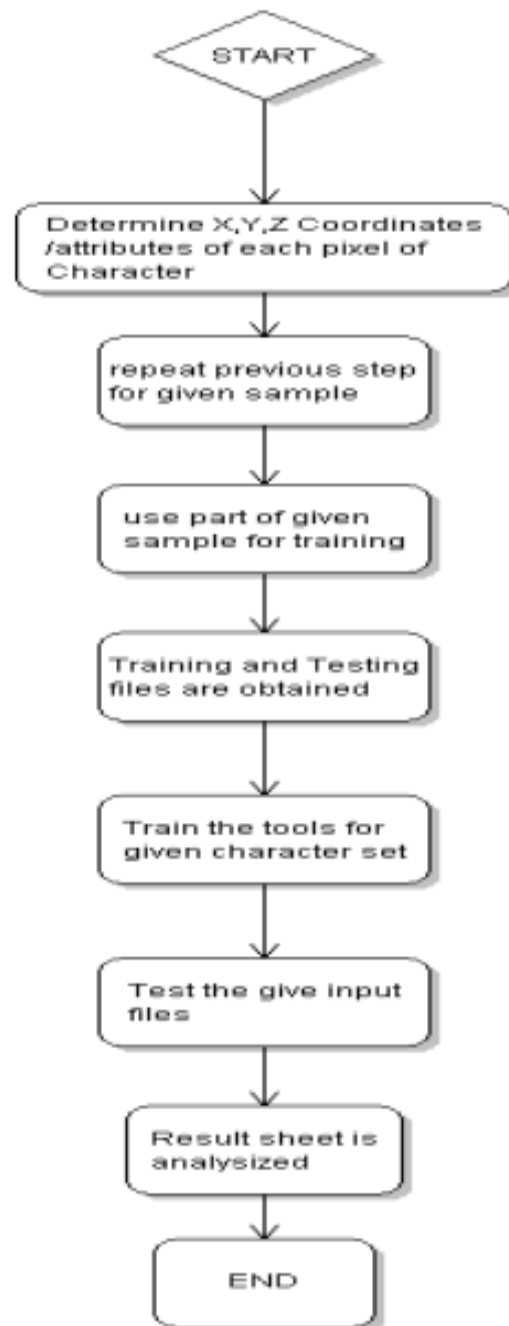


Fig1: Flow chart of OCR

2 BACKGROUND

The OCR system is divided into 2 sections. The first section is the training and the second section is the recognition of images. The stages of this system are shown in the figure below. Both sections the training and testing system includes image acquisition, preprocessing and feature extraction.

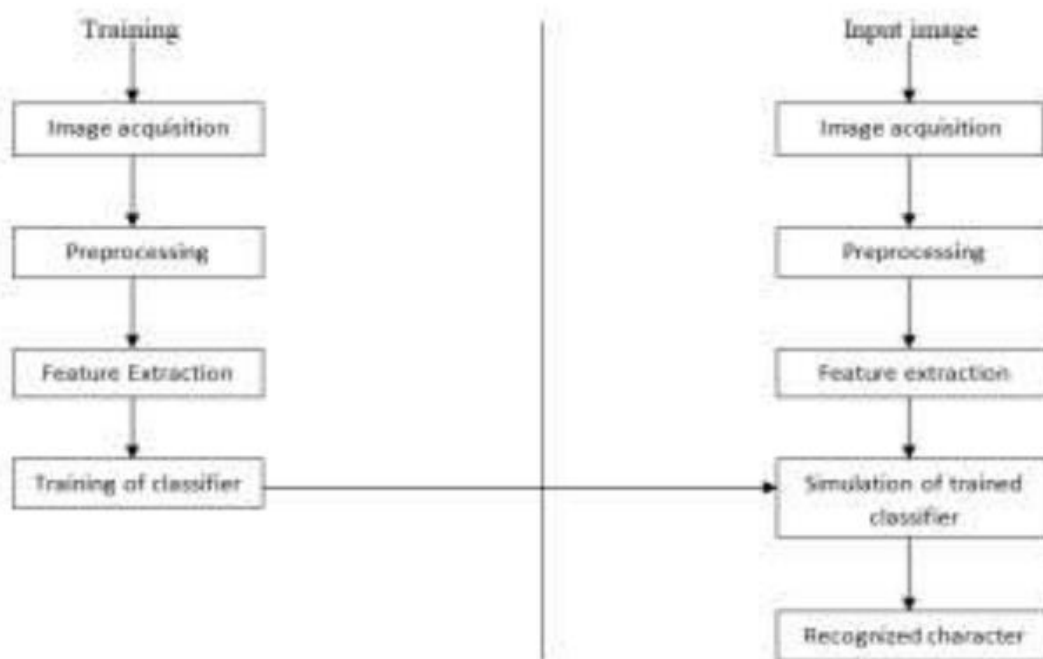


Fig2: 2 sections of OCR

2.1 Line Segmentation

Line segmentation involves separating the text line by line. This is crucial because the characters of handwritten text need to be isolated before applying the recognition techniques. In handwritten texts this becomes a challenge when lines are not spaced uniformly or when lines are very close to one another. Classical algorithms used here include edge detection techniques. Some of the sophisticated techniques involve using prototype based fuzzy-clustering algorithms, split and merge segmentation based on fuzzy clustering.

2.2 Character Segmentation

Character segmentation is an operation that seeks to decompose an image of sequence of characters into sub-images of individual symbols. This is necessary because some of the texts might be cursive handwritten texts, some might have very close characters, some far. It depends on the individual and the style of his handwriting. It is achieved by template matching, extracting the distinguishing attribute of character image. Finding the member of given set whose attributes best match to those of the input and give the output.

2.3 Pre-processing

Most common noise now a days is the salt & pepper noise, it is basically black and white dots in an image. These noises occur due to the disturbance in the environment.

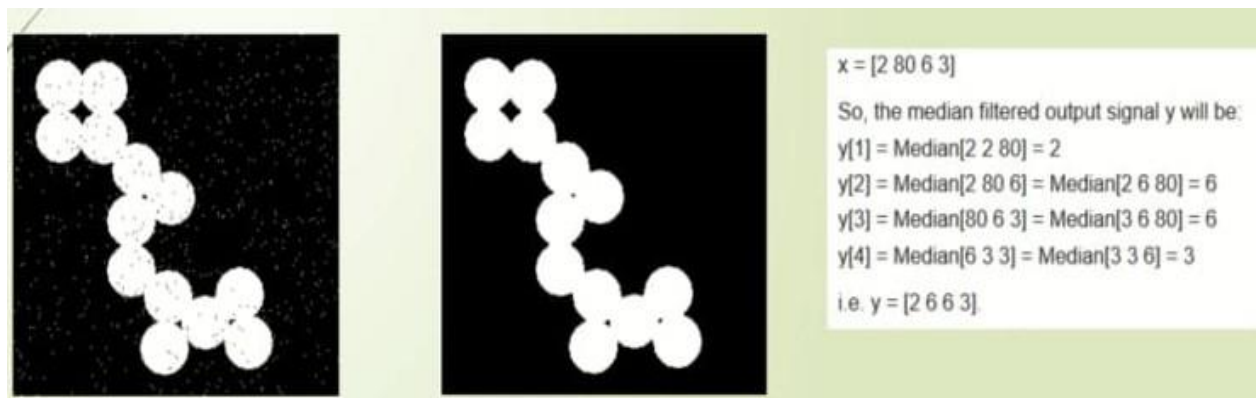


Fig3: SALT & PEPPER NOISE

Effective noise reduction function for this kind of noise is due to “Median Filtering “

Median Filtering works by choosing the median of a group of pixels and assigning its values to the neighboring pixels.

Suppose there is a pixel $X=[2 \ 80 \ 6 \ 3]$, let us assume a value $Y[1]= [2 \ 2 \ 80]$ and we arrange it in ascending order therefore, the median of $Y[1]=2$ and similarly we find the median for $Y[2]$, $Y[3]$, $Y[4]$

Finally, when we combine all the median values of $Y[1]$, $Y[2]$, $Y[3]$, $Y[4]$ we get $Y=[2 \ 6 \ 6 \ 3]$, so this is replaced by $X=[2 \ 80 \ 6 \ 3]$

Therefore, by doing this we have a uniform complexion.

3 DESIGN AND IMPLEMENTATION

3.1 Convolutional Neural Network

The models used in neural networks is convolutional neural networks. They are computing systems that are inspired by biological NN in the brain and works in pretty much the similar way. These networks are based on the collection of connected units called artificial neurons or just neurons. Each connection between these neurons can transmit a signal from one neuron to another and receiving neuron then process the signal and then signals downstream neurons connected to it.

Typically neurons are organized in layers, different layers may perform different kinds of transformations on their inputs and signals essentially travel from the first layer called the input layer to the last layer called output layer and any layer between the output and input layer is called hidden layer.

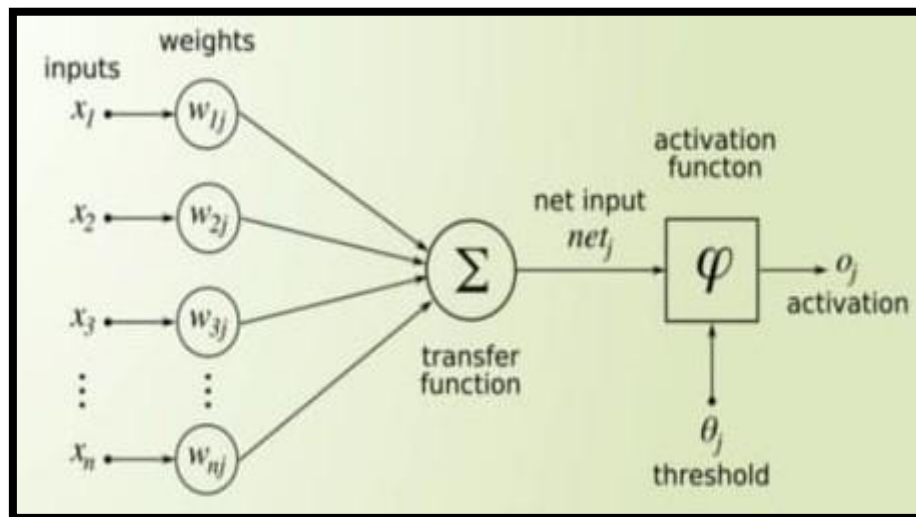


Fig4: Basic Neural Network

3.2 Back-Propagation

Back propagation method is used for training a multi-layer artificial neural network with mathematical formulation. Training the network to balance between the input patterns that are used in training and their responses to the input patterns is the main objective of this network.

Phases of back propagation algorithm:

1. Propagate the output results back to the neural network.
2. Update the weight after each propagation.

3.3 Multi-layered Perceptron (MLP)

It is a feed-forward ANN, data is propagated from input to output units in a feed-forward way i.e. there is no feedback from the output and we can have multiple classifiers running parallel.

Data processing may extend over multiple layers. One hidden layer MLP can approximate any function to any desired accuracy because of these advantages MLP classifiers can avoid the bulk of calculations performed in conventional neural networks.

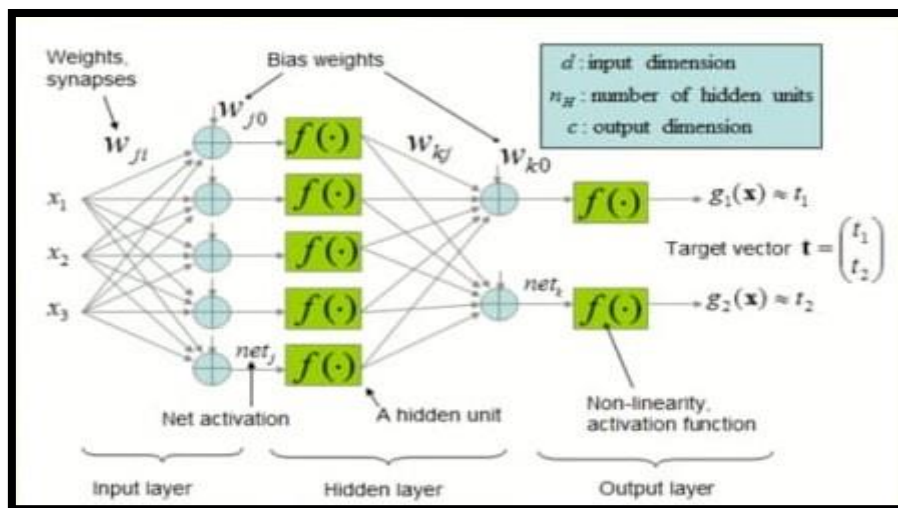


Fig5: Multi-layer Perceptron

- Weights synapses – it is the strength or amplitude of connections between the nodes i.e. how strongly X_1 and the node is connected.
- It contains bias weights of each node
- $f(\cdot)$ transfer function, in the hidden unit

CODE FOR DIGIT RECOGNITION USING CNN

Digit Recognition

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import keras
keras.backend.backend()
from keras.datasets import digit_mnist
```

End of Description

```
(x_train, y_train), (x_test, y_test) = digit_mnist.load_data()
```

Training the Neural Network

```
x_train.shape
x_test.shape
plt.matshow(x_train[1])
y_train[1]
x_train = x_train/255
x_test = x_test/255
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(20, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

Summary of the model

```
model.summary()
```

```
model.summary()
```

Layer (type)	Output Shape	Param #
=====	=====	=====
flatten_1 (Flatten)	(None, 784)	0
dense_1 (Dense)	(None, 20)	15700
dense_2 (Dense)	(None, 10)	210
=====	=====	=====

```
model.compile(loss="sparse_categorical_crossentropy",  
              optimizer="adam",  
              metrics=["accuracy"])
```

```
##### The model performs 5 iterations #####
```

```
model.fit(x_train, y_train, epochs=5)
```

```
##### Testing the Neural Network #####
```

```
plt.matshow(x_test[1])  
x_test.shape  
yp = model.predict(x_test)  
yp[1]  
np.argmax(yp[1])  
model.evaluate(x_test, y_test)
```

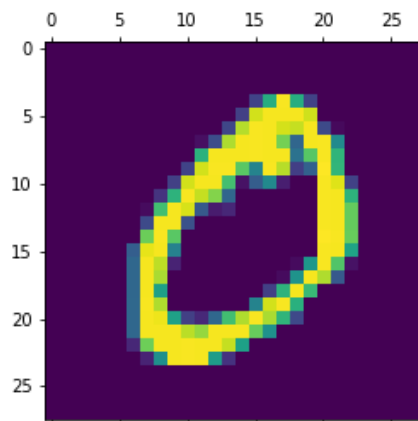
RESULTS AND EXPLANATION OF DIGIT RECOGNITION

Initially I did a digit recognition using convolution neural networks.

- I used a dataset from keras
- each image is 28x28 pixels grid
- I used the backend to be tensorflow
- The training set data has 60,000 images and testing set data has 10,000 images
- command --> plt.matlabshow , gives us the pictures

```
In [7]: plt.matshow(x_train[1])
```

```
Out[7]: <matplotlib.image.AxesImage at 0x1b077bf1748>
```



Building a neural network

I used a sequential model that helps in building a linear stack of neural networks .The first layer is the input layer for which I used Flatten, it converts 2 dimensional to 1 dimensional array and the second layer is the hidden layer for which I used dense layer , it is a trial and error method , in my case I gave 20 layers as that gave me maximum accuracy .The third layer is the output layer for which I used the activation function to be softmax , in my case I gave 10 layers .

- ✚ Neurons have an activation function, it can be either sigmoidal , relu etc. relu is a popular activation function.

- ✚ softmax is the distribution of a set of numbers into probability of available classification classes.
- ✚ There are 784 neurons in the input layer.

Compiling of the model

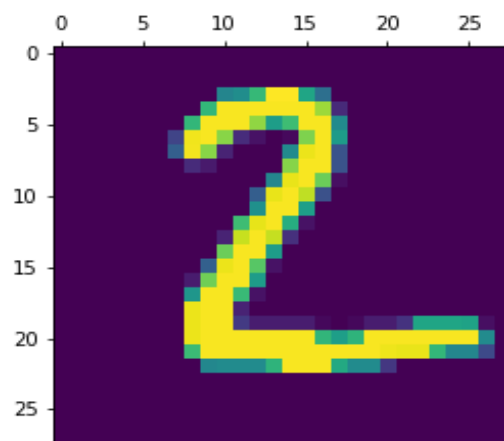
The parameters that are used for the code is loss, optimizes and fit

- loss helps in the calculation
- optimize uses adam , which is used to adjust weights
- fit function is used for training epochs
- I had taken 5 iteration (epochs =5)

Prediction

```
In [15]: plt.matshow(x_test[1])
```

```
Out[15]: <matplotlib.image.AxesImage at 0x1b07486c5f8>
```



```
In [16]: yp = model.predict(x_test)
```

```
In [17]: yp[1]
```

```
Out[17]: array([7.4918694e-06, 1.4652424e-03, 9.9673307e-01, 1.6251106e-03,
 1.3527719e-14, 4.3687709e-05, 1.0643885e-04, 7.0786224e-12,
 1.8901279e-05, 8.6483251e-13], dtype=float32)
```

Evaluate method of accuracy

Command → `model.evaluate (x_test, y_test)`

The loss is found to be 0.16363458970151842

The accuracy is found to be 0.9511

CODE FOR HANDWRITING TEXT RECOGNITION USING NEURAL NETWORK

Handwriting Text Recognition

```
from __future__ import absolute_import, division,
print_function, unicode_literals
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds
from PyInquirer import prompt
import os
```

End of Description

```
history = ""
train_images = []
train_labels = []
test_images = []
test_labels = []
```

Opening an environment for Tensorflow 2.0

```
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

Training the Neural Network

```
def train_model():
```

```
    global history, test_images, test_labels, train_images,
    train_labels
```

```

model = models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(27, activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=20)
model.save('assets/model.h5')

```

Finding the epochs and accuracy for 20 iterations

```
def visualise_train_process():
```

```

    global history
    model = models.load_model('assets/model.h5')

    plt.plot(history.history['accuracy'], label='accuracy')
    plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.ylim([0.5, 1])
    plt.legend(loc='lower right')

```

```

    test_loss, test_acc = model.evaluate(test_images,
    test_labels, verbose=2)
    print(test_acc)

```

Splitting the dataset into training and testing

```
def load_dataset():
```

```

    mnist_train = tfds.load(name="emnist/letters", split="train")
    mnist_test = tfds.load(name="emnist/letters", split="test")
    assert isinstance(mnist_train, tf.data.Dataset)
    assert isinstance(mnist_test, tf.data.Dataset)

```



```
global train_images, train_labels, test_images, test_labels
```

```
##### Image and label of training dataset #####
```

```
for mnist_example in mnist_train:
```

```
    image, label = mnist_example["image"],  
    mnist_example["label"]  
    train_images.append(image.numpy()[:, :, 0])  
    train_labels.append(label.numpy())
```

```
##### Image and label of testing dataset #####
```

```
for mnist_example in mnist_test:
```

```
    image, label = mnist_example["image"],  
    mnist_example["label"]  
    test_images.append(image.numpy()[:, :, 0])  
    test_labels.append(label.numpy())
```

```
train_images, test_images = np.asarray(train_images) /  
255.0, np.asarray(test_images) / 255.0  
train_labels, test_labels = np.asarray(train_labels),  
np.asarray(test_labels)
```

```
##### Predicting the alphabet with its position #####
```

```
def predict_stuff(num):
```

```
    global test_images, test_labels  
    keys = [  
        'nil', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',  
        'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'  
    ]
```

```

model = models.load_model('assets/model.h5')
predictions = model.predict(test_images)
print(test_labels[num], keys[np.argmax(predictions[num])])

img = test_images[num]
plt.imshow(img)
plt.show()

##### Making a choice #####

choices = ["Train Model", "Make a prediction", "Exit"]

questions = [
    {
        'type': 'list',
        'name': 'choice',
        'message': 'What do you want to do?',
        'choices': choices
    }
]

print("loading Dataset... \r", end="")
load_dataset()

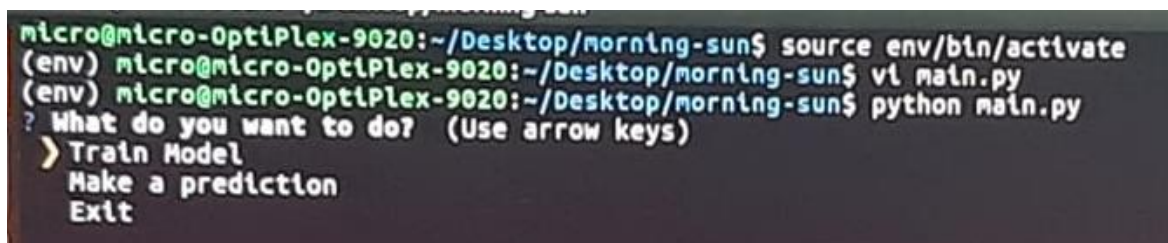
while True:
    answer = prompt(questions)
    if answer['choice'] == choices[0]:
        train_model()
    elif answer['choice'] == choices[1]:
        predict_stuff(np.random.randint(len(test_images),
                                         size=1).item())
    else:
        print("Thank you")
        break

```

RESULTS OF HANDWRITING TEXT RECOGNITION

We will be examining the images of English alphabet letters that are generated randomly by the machine using the Neural Network that was trained and tested

1. Make a choice



```
micro@micro-OptiPlex-9020:~/Desktop/morning-sun$ source env/bin/activate
(env) micro@micro-OptiPlex-9020:~/Desktop/morning-sun$ vi main.py
(env) micro@micro-OptiPlex-9020:~/Desktop/morning-sun$ python main.py
? What do you want to do? (Use arrow keys)
> Train Model
  Make a prediction
  Exit
```

2. Train Model

It runs 20 epochs and shows the accuracy in each iteration

ITERATION	ACCURACY
1 epoch	0.75
2 epoch	0.85
3 epoch	0.88
4 epoch	0.89
5 epoch	0.90
6 epoch	0.905
7 epoch	0.911
8 epoch	0.916
9 epoch	0.918
10 epoch	0.921
11 epoch	0.924
12 epoch	0.927
13 epoch	0.929
14 epoch	0.931
15 epoch	0.932
16 epoch	0.934
17 epoch	0.936
18 epoch	0.9383

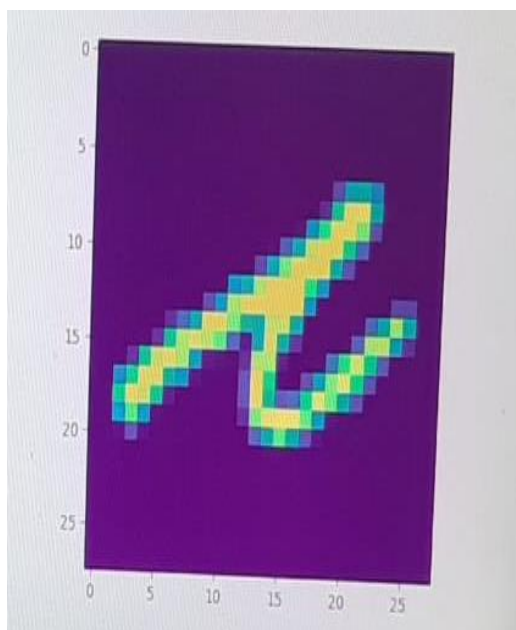
19 epoch	0.9389
20 epoch	0.94

3. Making Prediction

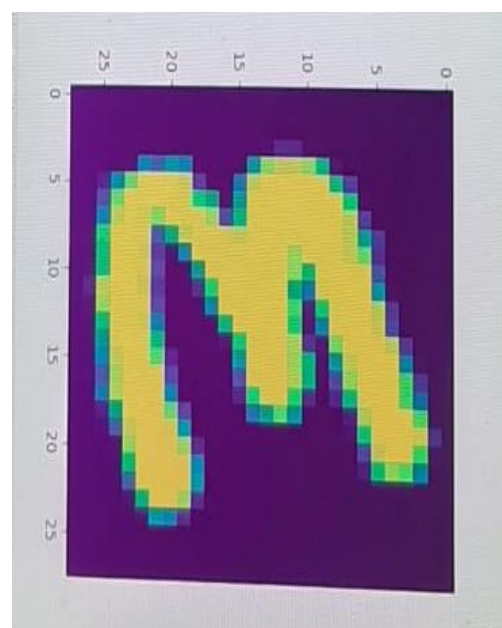
```

micro@micro-OptiPlex-9020:~/Desktop/morning-sun
(env) micro@micro-OptiPlex-9020:~/Desktop/morn
? What do you want to do? Make a prediction
8 H
? What do you want to do? Make a prediction
16 P
? What do you want to do? Make a prediction
13 M

```



8 (label) H (prediction)



13 (label) M (prediction)

9 CONCLUSION

Identifying the alphabet characters using neural network is implemented in this project. Identifying the correct accurate label is the main function of OCR. Several alphabet identification becomes difficult as they have same feature such as D and B (d and b) , I and L (i and l) . If the dataset has poor quality of images , it could cause difficulty in identifying the letter.

REFERENCES

Handwriting Text Recognition using Deep Learning (Convolutional Neural Networks)

[https:// towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5](https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5)

Dataset for character recognition

<http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/>

Offline Handwriting Recognition

<https://www.kaggle.com/tejasreddy/offline-handwriting-recognition-cnn>

