**RECOGNITION OF HANDWRITTEN DIGITS**

A MINI PROJECT REPORT

*Submitted by*

**VELPRAKASH.S 311521243062**

**TARUN.A 311521243059**

**SRIRAM.R 311521243053**

*In partial fulfillment for the award of the degree Of*

**BACHELOR OF TECHNOLOGY**

*In*

**ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE KODAMBAKKAM, CHENNAI – 600024.**

**ANNA UNIVERSITY: CHENNAI 600 025**

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **S.NO** | **TITLE** |
| 1 | Introduction |
| 2 | Hardware and Software Requirements |
| 3 | Approach |
| 4 | Neural networks – Methodology |
| 5 | Modules Used |
| 6 | Training the model |
| 7 | Testing the model |
| 8 | Pridiction |
| 9 | Codings |
| 10 | Sample Outputs |
| 11 | Conclusion |
| 12 | References |

**INTRODUCTION:**

Handwritten image recognition is probably one of the most interesting and challenging applications in the field of pattern recognition. Handwritten recognition is divided into two types of techniques: on-line and off-line. Off-line techniques include reading the character using an image capture device, such as a camera. while the technique which is being dealt here is Off-line which means to convert a handwritten image into a machine readable form.

The major factor behind choosing this particular application is its numerous applications such as Automatic Number Plate Recognition, assisting blind and visually impaired people,automatic check processing for banks, and to process huge number of documents in industries like healthcare, legal, education, and finance the focus if the work described in this paper is on handwritten digits. The paper will further be covering data collection, image pre- processing, feature extraction, and finally classification.

**HARDWARE AND SOFTWARE REQUIREMENTS:**

* Python editor [visual studio code] with all required packages
* High speed internet facility
* Minimum 2 GB ram influenced computer with high speed data transfer capability

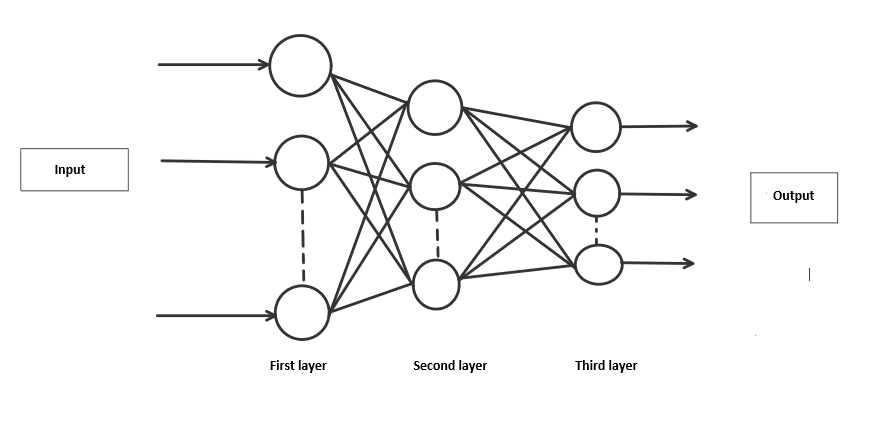
**APPROACH:**

We will approach this project by using a three-layered Neural Network.

* **The input layer:** It distributes the features of our examples to the next layer for calculation of activations of the next layer.
* **The hidden layer:** They are made of hidden units called activations providing nonlinear ties for the network. A number of hidden layers can vary according to our requirements.
* **The output layer:** The nodes here are called output units. It provides us with the final prediction of the Neural Network on the basis of which final predictions can be made.

A neural network is a model inspired by how the brain works. It consists of multiple layers having many activations, this activation resembles neurons of our brain. A neural network tries to learn a set of parameters in a set of data which could help to recognize the underlying relationships.

**NEURAL NETWORKS – METHODOLOGY:**

We have implemented a Neural Network with 1 hidden layer having *100* activation units (excluding bias units).. Then features are divided by *255* to rescale them into a range of *[0,1]* to avoid overflow during computation. Data is split up into *60,000* training and *10,000* testing examples. The regularization parameter lambda is set to 0.1 to address theproblem of overfitting. Optimizer is run for 70 iterations to find the best fit. 

**MODULES USED:**

1. TensorFlow
2. Keras
3. Tkinter
4. Win32gui
5. OpenCV

**TRAINING THE DATASET:**

A lot of other models have been published with 97-99% accuracy but they still guess the number wrong even when the digit is written clearly. Although the training accuracy is less compared to the primitive models it can guess the number more effectively on the test set or any new set. the **ImageDataGenerator()** function from **tensorflow.keras.preprocessing.image**has been used for the augmentation purpose. I’ve trained the model for 10 epochs.

**CODING:**

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras import backend as K

# the data, split between train and test sets

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

print(x\_train.shape, y\_train.shape)

x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1)

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1)

input\_shape = (28, 28, 1)

# convert class vectors to binary class matrices

y\_train = keras.utils.to\_categorical(y\_train, 10)

y\_test = keras.utils.to\_categorical(y\_test, 10)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

print('x\_train shape:', x\_train.shape)

print(x\_train.shape[0], 'train samples')

print(x\_test.shape[0], 'test samples')

batch\_size = 128

num\_classes = 10

epochs = 10

model = Sequential()

model.add(Conv2D(32, kernel\_size=(5, 5),activation='relu',input\_shape=input\_shape))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.3))

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical\_crossentropy,optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])

hist = model.fit(x\_train, y\_train,batch\_size=batch\_size,epochs=epochs,verbose=1,validation\_data=(x\_test, y\_test))

print("The model has successfully trained")

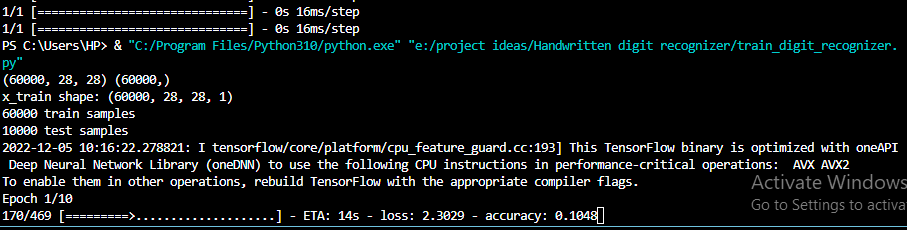
score = model.evaluate(x\_test, y\_test, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

model.save('mnist.h5')

print("Saving the model as mnist.h5")



**TESTING THE MODEL:**

As said earlier the model predicted 100% correct on the test set. In addition, it predicts better than the primitive model on the new test set. This proves the effectiveness of the algorithm.



**PREDICTION**

I’ve made a kind of a scribble pad where the user can write the number to get the model to predict it. This uses the Pygame module for this purpose. Once the user has completed writing the number the user could press any key to predict the number. The program could also be tuned to auto predict after drawing without inputting a key.

The user interface is designed very friendly (Pygame window):

The left mouse button is used to write and the right mouse button could be used as an eraser.

The process of predicting starts when the user completes the drawing or when the keystroke is given. For the sake of simplicity, the Pygame window is saved as a PNG image and then passed to the Guess function.

**CODING:**

from keras.models import load\_model

from tkinter import \*

import tkinter as tk

import win32gui

from PIL import ImageGrab, Image

import numpy as np

model = load\_model('mnist.h5')

def predict\_digit(img):

    #resize image to 28x28 pixels

    img = img.resize((28,28))

    #convert rgb to grayscale

    img = img.convert('L')

    img = np.array(img)

    #reshaping to support our model input and normalizing

    img = img.reshape(1,28,28,1)

    img = img/255.0

    res = model.predict([img])[0]

    return np.argmax(res), max(res)

class App(tk.Tk):

    def \_\_init\_\_(self):

        tk.Tk.\_\_init\_\_(self)

        self.x = self.y = 0

        # Creating elements

        self.canvas = tk.Canvas(self, width=300, height=300, bg = "white", cursor="cross")

        self.label = tk.Label(self, text="Draw..", font=("Helvetica", 48))

        self.classify\_btn = tk.Button(self, text = "Recognise", command = self.classify\_handwriting)

        self.button\_clear = tk.Button(self, text = "Clear", command = self.clear\_all)

        # Grid structure

        self.canvas.grid(row=0, column=0, pady=2, sticky=W, )

        self.label.grid(row=0, column=1,pady=2, padx=2)

        self.classify\_btn.grid(row=1, column=1, pady=2, padx=2)

        self.button\_clear.grid(row=1, column=0, pady=2)

        #self.canvas.bind("<Motion>", self.start\_pos)

        self.canvas.bind("<B1-Motion>", self.draw\_lines)

    def clear\_all(self):

        self.canvas.delete("all")

    def classify\_handwriting(self):

        HWND = self.canvas.winfo\_id()  # get the handle of the canvas

        rect = win32gui.GetWindowRect(HWND)  # get the coordinate of the canvas

        a,b,c,d = rect

        rect=(a+4,b+4,c-4,d-4)

        im = ImageGrab.grab(rect)

        digit, acc = predict\_digit(im)

        self.label.configure(text= str(digit)+', '+ str(int(acc\*100))+'%')

    def draw\_lines(self, event):

        self.x = event.x

        self.y = event.y

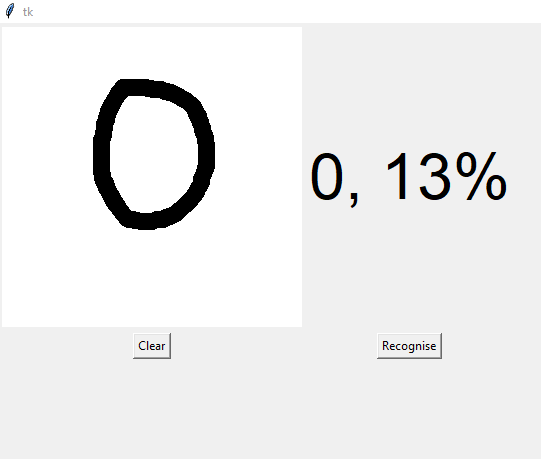
        r=8

        self.canvas.create\_oval(self.x-r, self.y-r, self.x + r, self.y + r, fill='black')

app = App()

mainloop()

**SAMPLE OUTPUTS:**

**CONCLUSION**

In this project we use Artificial Neural Network for digit recognition. It turned out that all the three methods were very promising but Neural Networks was very challenging to apply and yielded very good results. Due to time constraints our project was restricted to digits, and for future work it will be interesting to investigate characters and more advanced applications could involve facial or handwriting recognition

**REFERENCES:**.

1. Utkarsh Shaw (<https://auth.geeksforgeeks.org/user/utkarshshaw/profile>)
2. Tania (<https://auth.geeksforgeeks.org/user/taniachanana02/profile>)
3. Rishab Mamgai (<https://auth.geeksforgeeks.org/user/rishabmamgai/profile>)

4.Pavithrakumar(git clone <https://github.com/pavitrakumar78/Python-Custom-Digit->Recognition.git)