**MACHINE LEARNING & DEEP LEARNING**

NAME: MONIKA BHAGWAN DESAI

ROLL NO: 531

MSC-CS (SEM 3)

**MINI PROJECT: Implementation of handwritten digit recognition using MNIST dataset.**

Handwritten digit recognition:

This is the process to provide the ability to machines to recognize human handwritten digits. It is not an easy task for the machine because handwritten digits are not perfect, vary from person-to-person, and can be made with many different flavours. Commands to install the necessary libraries for this project:

**pip install numpy**

**pip install tensorflow**

**pip install keras**

**pip install pillow**

The MNIST dataset

MNIST is the most popular dataset for enthusiasts of machine learning and deep learning. Above 60,000 plus training images of handwritten digits from zero to nine and more than 10,000 images for testing are present in the [MNIST dataset](http://yann.lecun.com/exdb/mnist/).

**Steps to build Handwritten Digit Recognition System**

#### 1. Import libraries and dataset

Import all the needed modules for training our model. Keras library already contains many datasets and MNIST is one of them. We call mnist.load\_data() function to get training data with its labels and also the testing data with its labels.

**import keras**

**from keras.datasets import mnist**

**from keras.models import Sequential**

**from keras.layers import Dense Flatten**

**from keras.layers import Dropout**

**from keras.layers import Flatten**

**from keras.layers import Conv2D**

**from keras.layers import MaxPooling2D**

**from keras import backend as K**

**# to split the data of training and testing sets**

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

## The Data Pre-processing

Model cannot take the image data directly so we need to perform some basic operations and process the data to make it ready for our neural network. The dimension of the training data is (60000\*28\*28). One more dimension is needed for the CNN model so we reshape the matrix to shape (60000\*28\*28\*1).

**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)**

**# conversion of class vectors to matrices of binary class**

**y\_train = keras.utils.to\_categorical(y\_train, num\_classes)**

**y\_test = keras.utils.to\_categorical(y\_test, num\_classes)**

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

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

**x\_train /= 255**

**x\_test /= 255**

## Create the model

A convolutional layer and pooling layers are the two wheels of a CNN model. The reason behind the success of CNN for image classification problems is its feasibility with grid structured data. We will use the Adadelta optimizer for the model compilation.

**batch\_size = 128**

**num\_classes = 10**

**epochs = 10**

**model = Sequential()**

**model.add(Conv2D(32, kernel\_size=(3, 3),activation='relu',input\_shape=input\_shape))**

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

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

**model.add(Dropout(0.25))**

**model.add(Flatten())**

**model.add(Dense(256, 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'])**

## Train the model

To start the training of the model we can simply call the model.fit() function of Keras. It takes the training data, validation data, epochs, and batch size as the parameter.

The training of model takes some time. After succesful model training, we can save the weights and model definition in the ‘mnist.h5’ file.

**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")**

**model.save('mnist.h5')**

**print("Saving the bot as mnist.h5")**

**Output** Evaluate the model

## To evaluate how accurate our model works, we have around 10,000 images in our dataset. In the training of the data model, we do not include the testing data that’s why it is new data for our model. Around 99% accuracy is achieved with this well-balanced MNIST dataset.

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

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

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



## Create GUI to predict digits

guidigit\_recog.py :

**from keras.models import load\_model**

**from Tkinter import \***

**import Tkinter successful 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 for model normalization**

**img = img.reshape(1,28,28,1)**

**img = img/255.0**

**#predicting the class**

**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=200, height=200, bg = "black", cursor="cross")**

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

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

**self.button\_clear = tk.Button(self, text = "Dlt", 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("", self.start\_pos)**

**self.canvas.bind("", self.draw\_lines)**

**def clear\_all(self):**

**self.canvas.delete("all")**

**def classify\_handwriting(self):**

**Hd = self.canvas.winfo\_id() # to fetch the handle of the canvas**

**rect = win32gui.GetWindowRect(Hd) # to fetch the edges of the canvas**

**im = ImageGrab.grab(rect)**

**digit, acc = predict\_digit(im)**

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

**def draw\_lines(slf, event):**

**slf.x = event.x**

**slf.y = event.y**

**r=8**

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

**app = App()**

**mainloop()**



