**TEAM ID :PNT2022TMID43906**

**Add CNN Layers**

import cv2

import numpy as np

from keras.datasets import mnist

from keras.layers import Dense, Flatten, MaxPooling2D, Dropout

from keras.layers.convolutional import Conv2D

from keras.models import Sequential

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

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

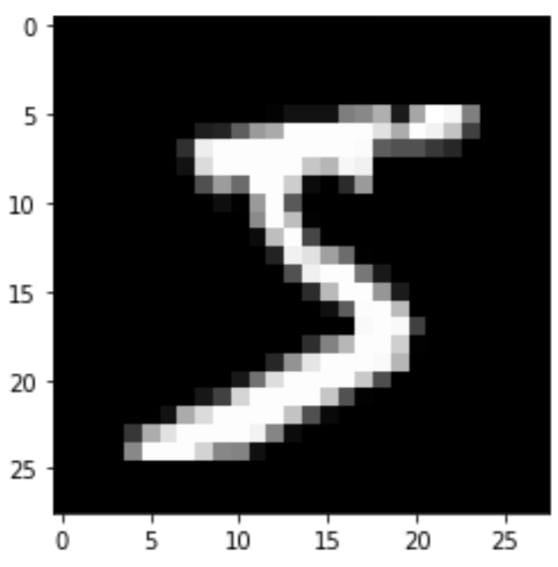
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz

11490434/11490434 [==============================] - 0s 0us/step

plt.imshow(X\_train[0], cmap="gray")

plt.show()

print (y\_train[0])



print ("Shape of X\_train: {}".format(X\_train.shape))

print ("Shape of y\_train: {}".format(y\_train.shape))

print ("Shape of X\_test: {}".format(X\_test.shape))

print ("Shape of y\_test: {}".format(y\_test.shape))

Shape of X\_train: (60000, 28, 28)

Shape of y\_train: (60000,)

Shape of X\_test: (10000, 28, 28)

Shape of y\_test: (10000,)

# Reshaping so as to convert images for our model

X\_train = X\_train.reshape(60000, 28, 28, 1)

X\_test = X\_test.reshape(10000, 28, 28, 1)

print ("Shape of X\_train: {}".format(X\_train.shape))

print ("Shape of y\_train: {}".format(y\_train.shape))

print ("Shape of X\_test: {}".format(X\_test.shape))

print ("Shape of y\_test: {}".format(y\_test.shape))

Shape of X\_train: (60000, 28, 28, 1)

Shape of y\_train: (60000,)

Shape of X\_test: (10000, 28, 28, 1)

Shape of y\_test: (10000,)

#one hot encoding

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

model = Sequential()

## Declare the layers

layer\_1 = Conv2D(64, kernel\_size=3, activation='relu', input\_shape=(28, 28, 1))

layer\_2 = MaxPooling2D(pool\_size=2)

layer\_3 = Conv2D(32, kernel\_size=3, activation='relu')

layer\_4 = MaxPooling2D(pool\_size=2)

layer\_5 = Dropout(0.5)

layer\_6 = Flatten()

layer\_7 = Dense(128, activation="relu")

layer\_8 = Dropout(0.5)

layer\_9 = Dense(10, activation='softmax')

## Add the layers to the model

model.add(layer\_1)

model.add(layer\_2)

model.add(layer\_3)

model.add(layer\_4)

model.add(layer\_5)

model.add(layer\_6)

model.add(layer\_7)

model.add(layer\_8)

model.add(layer\_9)

**Compiling The Model**

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

**Train The Model**

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=3)

Epoch 1/3

1875/1875 [==============================] - 58s 31ms/step - loss: 0.8654 - accuracy: 0.7801 - val\_loss: 0.1307 - val\_accuracy: 0.9630

Epoch 2/3

1875/1875 [==============================] - 58s 31ms/step - loss: 0.2703 - accuracy: 0.9201 - val\_loss: 0.0750 - val\_accuracy: 0.9757

Epoch 3/3

1875/1875 [==============================] - 56s 30ms/step - loss: 0.2055 - accuracy: 0.9385 - val\_loss: 0.0746 - val\_accuracy: 0.9772

**Observing The Metrics**

example = X\_train[1]

prediction = model.predict(example.reshape(1, 28, 28, 1))

print ("Prediction (Softmax) from the neural network:\n\n {}".format(prediction))

hard\_maxed\_prediction = np.zeros(prediction.shape)

hard\_maxed\_prediction[0][np.argmax(prediction)] = 1

print ("\n\nHard-maxed form of the prediction: \n\n {}".format(hard\_maxed\_prediction))

print ("\n\n--------- Prediction --------- \n\n")

plt.imshow(example.reshape(28, 28), cmap="gray")

plt.show()

print("\n\nFinal Output: {}".format(np.argmax(prediction)))

1/1 [==============================] - 0s 83ms/step

Prediction (Softmax) from the neural network:

[[9.99999881e-01 7.21094625e-13 7.90088137e-08 3.49195464e-11

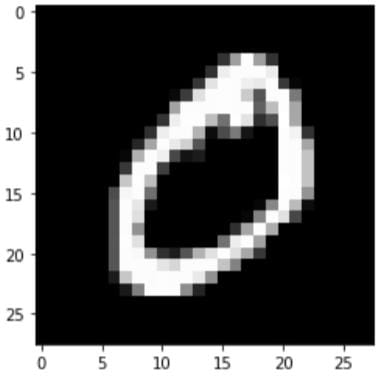
1.54954244e-11 5.48896974e-13 1.05098525e-08 1.00683108e-10

7.00186797e-10 1.28125794e-08]]

Hard-maxed form of the prediction:

[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

--------- Prediction ---------



Final Output:0

metrices=model.evaluate(X\_test,y\_test,verbose=0)

print("Metrices(test loss and Test Accuracy):")

print(metrices)

Metrices(test loss and Test Accuracy):

[0.07461030036211014, 0.9771999716758728]

**Test The Model**

image = cv2.imread('test\_image.jpg')

image = np.full((100,80,3), 12, dtype = np.uint8)

grey = cv2.cvtColor(image.copy(), cv2.COLOR\_BGR2GRAY)

ret, thresh = cv2.threshold(grey.copy(), 75, 255, cv2.THRESH\_BINARY\_INV)

contours,hierarchy = cv2.findContours(thresh.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

preprocessed\_digits = []

for c in contours:

x,y,w,h = cv2.boundingRect(c)

# Creating a rectangle around the digit in the original image (for displaying the digits fetched via contours)

cv2.rectangle(image, (x,y), (x+w, y+h), color=(0, 255, 0), thickness=2)

# Cropping out the digit from the image corresponding to the current contours in the for loop

digit = thresh[y:y+h, x:x+w]

# Resizing that digit to (18, 18)

resized\_digit = cv2.resize(digit, (18,18))

# Padding the digit with 5 pixels of black color (zeros) in each side to finally produce the image of (28, 28)

padded\_digit = np.pad(resized\_digit, ((5,5),(5,5)), "constant", constant\_values=0)

# Adding the preprocessed digit to the list of preprocessed digits

preprocessed\_digits.append(padded\_digit)

print("\n\n\n----------------Contoured Image--------------------")

import os, types

import pandas as pd

def \_iter\_(self): return 0

print=("\n\n\n----------------Contoured Image--------------------")

plt.imshow(image, cmap="gray")

plt.show()

inp = np.array(preprocessed\_digits

----------------Contoured Image--------------------

