**Ex4 - Classification of Email spam and MNIST data**

**GitHub Link:**

**Aim:**

To develop a python program

(i) To classify Emails as Spam or Ham

(ii) To recognize the digits of the MNIST dataset

Using Support Vector Machine (SVM) Model

**4.1 Classification of Email Spam or Ham using Support Vector Machine**

**(SVM)**

**Code & Output:**

**Import Dependencies**

import tensorflow as tf

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import os

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import MaxPool2D

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

import cv2

import time

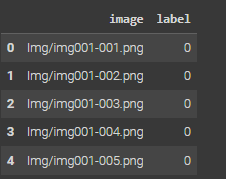
import urllib.request

from PIL import Image

**Reading And Sampling Data**

df = pd.read\_csv('english.csv')

df.head()



df.columns



df.index



im = Image.open(df.iloc[np.random.randint(0,3410),0])

im.show()

**Pre Processing**

**Handling NULL Values**

df['image'].isnull().sum()



df['label'].isnull().sum()



**URL Image Read Function with OpenCV**

def read\_url\_pic(x):

image\_url = x

with urllib.request.urlopen(image\_url) as url:

s = url.read()

arr = np.asarray(bytearray(s), dtype=np.uint8)

image = cv2.imdecode(arr, -1)

return image

**Plot Image from URL with Matplotlib**

def plot\_url\_pic(x):

image\_url = x

with urllib.request.urlopen(image\_url) as url:

s = url.read()

arr = np.asarray(bytearray(s), dtype=np.uint8)

image = cv2.imdecode(arr, -1)

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.show()

**Convert Image to Array with OpenCV**

def img\_to\_arr(x):

img = cv2.imread(x)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

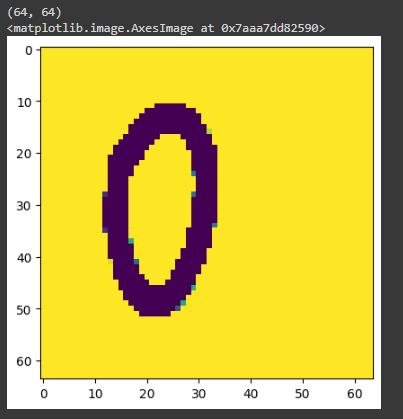
img = cv2.resize(img,(64,64))

return img

**Displaying Image Array Shape and Visualization**

print(img\_to\_arr(df['image'][0]).shape)

plt.imshow(img\_to\_arr(df['image'][0]))



**Parallel Image Processing with Multiprocessing**

start\_time = time.time()

import multiprocessing

with multiprocessing.Pool(4) as p:

    images = p.map(img\_to\_arr, df['image'])

end\_time = time.time()

print("with multiprocessing:", end\_time-start\_time)



**Reshape Image Array for Deep Learning Model Input**

x = np.array(images).reshape(3410, 64, 64,1)

x.shape

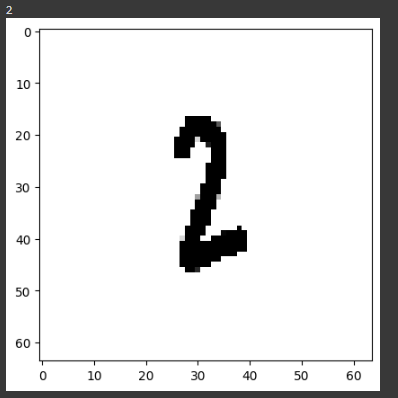


**Visualizing Image with Corresponding Label**

img = x[119].reshape((64,64))

plt.imshow(img,cmap='gray')

print(df['label'].iloc[119])



**Label Encoding for Classification**

y = df['label']

le = LabelEncoder()

y\_label = le.fit\_transform(y)

**Splitting Data into Training and Testing Sets**

train\_images,test\_images,train\_labels,test\_labels = train\_test\_split(x,y\_label,test\_size=0.2,random\_state=42)

train\_images = train\_images/255.0

test\_images = test\_images/255.0

**Convolutional Neural Network Model Training**

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

model = Sequential()

model.add(Conv2D(512, (5, 5), activation='relu', input\_shape=(64, 64, 1)))

model.add(MaxPooling2D(2, 2))

model.add(Conv2D(256, (3, 3), activation='relu'))

model.add(MaxPooling2D(2, 2))

model.add(Conv2D(256, (3, 3), activation='relu'))

model.add(MaxPooling2D(2, 2))

model.add(Flatten())

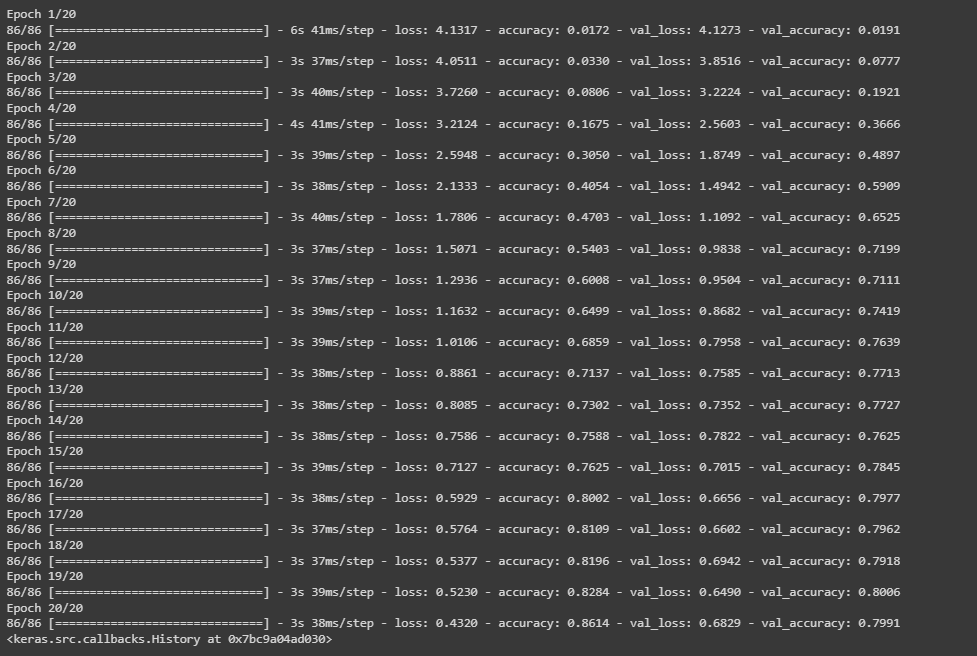
model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

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

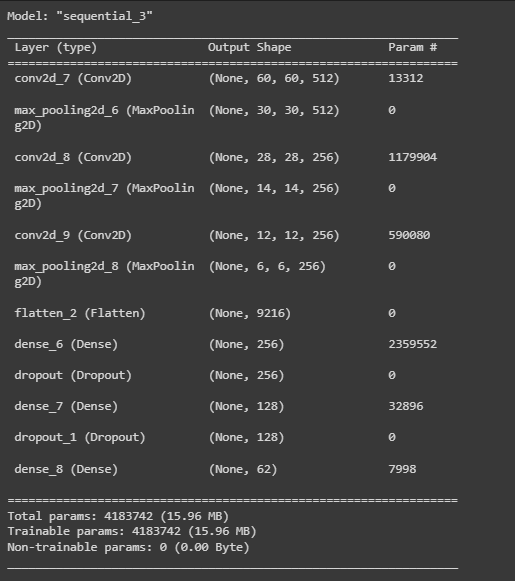
model.add(Dropout(0.5))

model.add(Dense(62, activation='softmax'))



**Display Model Summary**

model.summary()



**Inverse Transform Predicted and Actual Labels**

predicted\_labels = le.inverse\_transform(model.predict(test\_images).argmax(axis=1))

actual\_labels = le.inverse\_transform(test\_labels)



**Display Predicted and Actual Labels for a Specific Index**

print('predicted label is ;',predicted\_labels[67])

print('Actual Label is :',actual\_labels[67])



**Displaying Predicted and Actual Labels for a Specific Image**

# number of image to be predicted

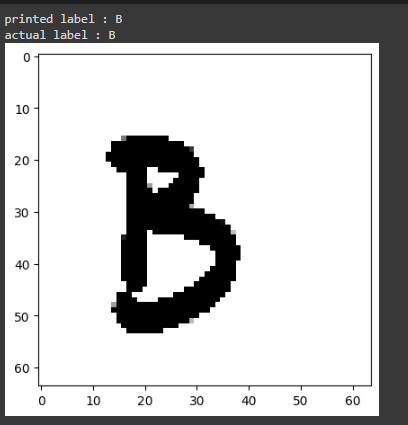
# change the value of I

i = 450

plt.imshow(test\_images[i],cmap='gray')

print('printed label :',predicted\_labels[i])

print('actual label :', actual\_labels[i])



**Evaluate Model Performance with Various Metrics**

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

y\_pred = model.predict(test\_images)

y\_pred\_labels = np.argmax(y\_pred, axis=1)

# Calculate evaluation metrics

accuracy = accuracy\_score(test\_labels, y\_pred\_labels)

precision = precision\_score(test\_labels, y\_pred\_labels, average='weighted')

recall = recall\_score(test\_labels, y\_pred\_labels, average='weighted')

f1 = f1\_score(test\_labels, y\_pred\_labels, average='weighted')

confusion\_mat = confusion\_matrix(test\_labels, y\_pred\_labels)

# Print or use the evaluation metrics as needed

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

print("Confusion Matrix:")

print(confusion\_mat)

