**Source Code**

# importing Required Libraries

import tensorflow as tf

import seaborn as sns

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from keras.utils.vis\_utils import plot\_model

# loading the MNIST dataset

mnist = tf.keras.datasets.mnist

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

sns.countplot(y\_train)

input\_shape = (28, 28, 1)

x\_train=x\_train.reshape(x\_train.shape[0], x\_train.shape[1], x\_train.shape[2], 1)

x\_train=x\_train / 255.0

x\_test = x\_test.reshape(x\_test.shape[0], x\_test.shape[1], x\_test.shape[2], 1)

x\_test=x\_test/255.0

y\_train = tf.one\_hot(y\_train.astype(np.int32), depth=10)

y\_test = tf.one\_hot(y\_test.astype(np.int32), depth=10)

plt.imshow(x\_train[100][:,:,0])

print(y\_train[100])

X\_train\_plot = x\_train.reshape(-1, 28, 28)

def Show\_example\_digits(mono = 'gray'):

    fig = plt.figure(figsize = (16, 16))

    for idx in range(15):

plt.subplot(5, 5,idx+1)

plt.imshow(X\_train\_plot[idx], cmap = mono)

plt.title("Digit {}".format(y\_train[idx]))

plt.tight\_layout()

Show\_example\_digits()

# determine the shape of the input images

inp\_shape = x\_train.shape[1:]

print(inp\_shape)

batch\_size = 64

num\_classes = 10

epochs = 10

# defining the model

model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(32, (5,5), padding='same', activation='relu', input\_shape=input\_shape),

    tf.keras.layers.Conv2D(32, (5,5), padding='same', activation='relu'),

    tf.keras.layers.MaxPool2D(),

tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(64, (3,3), padding='same', activation='relu'),

    tf.keras.layers.Conv2D(64, (3,3), padding='same', activation='relu'),

    tf.keras.layers.MaxPool2D(strides=(2,2)),

tf.keras.layers.Dropout(0.25),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(num\_classes, activation='softmax')

])

model.compile(optimizer=tf.keras.optimizers.RMSprop(epsilon=1e-08), loss='categorical\_crossentropy', metrics=['acc'])

# text Description of model

model.summary()

history = model.fit(x\_train, y\_train,

batch\_size=batch\_size,

                    epochs=epochs,

validation\_split=0.1)

# ploting the learning curves

fig, ax = plt.subplots(1,1)

ax.plot(history.history['loss'], color='b', label="Training Loss")

ax.plot(history.history['val\_loss'], color='r', label="Validation Loss")

legend = ax.legend(loc='best', shadow=True)

# evaluate the model

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

print(f'Accuracy: {accuracy\*100}')

y\_pred = model.predict(x\_test)

def draw\_output(idx\_nums):

plt.figure(figsize = (20, 20))

plt.xticks( range(10) )

    x = np.ceil(np.sqrt(len(idx\_nums)))

cnt = 1

    for ph in idx\_nums:

plt.subplot(x, x, cnt)

curr\_photo = y\_test[ph]

plt.xlim(0, 10)

plt.title("Digit: {0}\n idx: {1} ".format(np.argmax(y\_test[ph]), ph), fontsize = 10)

plt.bar(range(10), y\_pred[ph])

cnt += 1

cnt\_error = []

for idx, (a, b) in enumerate(zip(y\_test, y\_pred)):

    if np.argmax(a) == np.argmax(b): continue

cnt\_error.append( (np.argmax(a)) )

cnt\_error = np.unique(cnt\_error, return\_counts = True)

sns.set\_style("darkgrid")

plt.figure(figsize = (15, 7))

bar\_plot = sns.barplot(cnt\_error[0], cnt\_error[1], palette="muted")

plt.show()

cnt\_ind = 1

list\_idx = []

X\_val\_plot = x\_test.reshape( x\_test.shape[:-1] )

fig = plt.figure(figsize=(14, 14))

for idx, (a, b) in enumerate(zip(y\_test, y\_pred)):

    if np.argmax(a) == np.argmax(b): continue

    if (np.argmax(a) == 2 or np.argmax(a) == 9):

plt.subplot(5, 5, cnt\_ind)

plt.imshow(X\_val\_plot[idx], cmap='gray', interpolation='none')

plt.title('y\_true={0}\ny\_pred={1}\n ind = {2}'.format(np.argmax(a), np.argmax(b), idx))

plt.tight\_layout()

list\_idx.append(idx)

cnt\_ind += 1

image = x\_train[0]

# lets display the image which we want to predict

plt.imshow(np.squeeze(image), cmap='gray')

image.shape[0],image.shape[1],image.shape[2]

# make a prediction

# reshaping the image for model input

image= image.reshape(1,input\_shape[0],input\_shape[1],input\_shape[2])

# predicting the label of image

yhat = model.predict([image])

print('Predicted: {}'.format(np.argmax(yhat)))

# Predict the values from the testing dataset

Y\_pred = model.predict(x\_test)

# Convert predictions classes to one hot vectors

Y\_pred\_classes = np.argmax(Y\_pred,axis = 1)

# Convert testing observations to one hot vectors

Y\_true = np.argmax(y\_test,axis = 1)

# compute the confusion matrix

confusion\_mtx = tf.math.confusion\_matrix(Y\_true, Y\_pred\_classes)

plt.figure(figsize=(10, 8))

sns.heatmap(confusion\_mtx, annot=True, fmt='g')

model.save('hand\_written\_digits\_CNN.h5')

model = tf.keras.models.load\_model('hand\_written\_digits\_CNN.h5')

image = x\_test[100]

# lets display the image which we want to predict

plt.imshow(np.squeeze(image), cmap='gray')

# make a prediction

# reshaping the image for model input

image= image.reshape(1,input\_shape[0],input\_shape[1],input\_shape[2])

# predicting the label of image

yhat = model.predict([image])

print('Predicted: {}'.format(np.argmax(yhat)))

!tar -zcvf model.tgz hand\_written\_digits\_CNN.h5

!pip install watson-machine-learning-client

!pip install ibm\_watson\_machine\_learning

from ibm\_watson\_machine\_learning import APIClient

wml\_credentials = {

    "url":"https://jp-tok.ml.cloud.ibm.com",

    "apikey":"Y0hOkxEIr9-Qwjc7rRJqcboPqPn2GdjCddwHsedqsc8N"

}

client = APIClient(wml\_credentials)

client

client.spaces.get\_details()

space\_id = 'f5b3e32f-adf0-45f4-918c-88ac9ee62536'

client.set.default\_space(space\_id)

client.software\_specifications.list()

software\_space\_uid=client.software\_specifications.get\_uid\_by\_name('tensorflow\_rt22.1-py3.9')

software\_space\_uid

model\_details = client.repository.store\_model(model='model.tgz',meta\_props={

    client.repository.ModelMetaNames.NAME : 'A Novel Method for Handwritten Digit Recognition System',

    client.repository.ModelMetaNames.TYPE:'tensorflow\_2.7',

    client.repository.ModelMetaNames.SOFTWARE\_SPEC\_UID:software\_space\_uid

})

model\_details

model\_id = client.repository.get\_model\_id(model\_details)

model\_id

client.repository.download(model\_id,'model.tar.gb')

**app.py**

from flask import Flask,render\_template,request

from PIL import Image, ImageOps

import os

import random

import string

from pathlib import Path

import numpy as np

def random\_name\_generator(n):

  return ''.join(random.choices(string.ascii\_uppercase + string.digits, k=n))

app=Flask(\_\_name\_\_)

@app.route('/')

def home\_page():

    return render\_template('index.html')

@app.route('/index')

def ai\_engine\_page():

    return render\_template('index.html')

@app.route('/submit',methods=['POST'])

def submit():

    if request.method=='POST':

      image = request.files['image']

img\_name = image.filename

file\_path = os.path.join('static/data/', img\_name)

image.save(file\_path)

img = Image.open(image).convert("L")

img = img.resize((255, 255))

img.save(os.path.join('static/thumb/', "255X255\_"+img\_name))

      best = (9, 73.19)

      # others = [(0, 9.15),(1, 0.35000000000000003), (2, 0.4), (3, 0.0), (4, 109.9), (5, 4.1499999999999995), (6, 3.5), (7, 3.4000000000000004), (8, 3.15), (9, 365.95)]

      others = [(0, 1.83), (1, 0.07), (2, 0.08), (3, 0.0), (4, 21.98), (5, 0.83), (6, 0.7), (7, 0.68), (8, 0.63),(9, 73.19)]

      return render\_template("submit.html", best=best, others=others, img\_name=img\_name)

if \_\_name\_\_=="\_\_main\_\_":

app.run()

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**CNN.py**

import os

import random

import string

from pathlib import Path

import numpy as np

from tensorflow.keras.models import load\_model

from PIL import Image, ImageOps

def recognize(image):

  model=load\_model(Path("./model/model.h5"))

img = Image.open(image).convert("L")

img\_name = image.filename

  # img\_name = random\_name\_generator(10) + '.jpg'

  # if not os.path.exists(f".static/data/"):

  #   os.mkdir(os.path.join('./static', 'data'))

  #img.save(Path(f".static/data/{img\_name}"))

img = ImageOps.grayscale(img)

img = ImageOps.invert(img)

img = img.resize((28, 28))

  img2arr = np.array(img)

  img2arr = img2arr / 255.0

  img2arr = img2arr.reshape(1, 28, 28, 1)

  results  = model.predict(img2arr)

  best = np.argmax(results,axis = 1)[0]

  pred = list(map(lambda x: round(x\*100, 2), results[0]))

  values = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

  others = [(i,pred[i]) for i in range(0,10)]

  best = (pred.index(max(pred)),max(pred))

  #best = others.pop(best)

  return best,others