***HANDWRITTEN DIGITAL RECOGNITION USING CNN***

***SOURCE CODE:***

1. ***Import Libraries:***

import tensorflow as tf

import matplotlib.pyplot as plt

import seaborn as sn

import numpy as np

import pandas as pd

import math

import datetime

import platform

### *Splitting data into training and validation dataset*

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

X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X, y, test\_size = 0.2,random\_state = 1212)

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

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

print('X\_validation:', X\_validation.shape)

print('y\_validation:', y\_validation.shape)

### *3.  Reshaping train, test, and validation data*

x\_train\_with\_chanels = x\_train\_re.reshape(

x\_train\_re.shape[0],

IMAGE\_WIDTH,

IMAGE\_HEIGHT,

IMAGE\_CHANNELS

)

x\_validation\_with\_chanels = x\_validation\_re.reshape(

x\_validation\_re.shape[0],

IMAGE\_WIDTH,

IMAGE\_HEIGHT,

IMAGE\_CHANNELS

)

x\_test\_with\_chanels = x\_test\_re.reshape(

x\_test\_re.shape[0],

IMAGE\_WIDTH,

IMAGE\_HEIGHT,

IMAGE\_CHANNELS

)

## *Build the CNN model :*

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Convolution2D(

input\_shape=(IMAGE\_WIDTH, IMAGE\_HEIGHT, IMAGE\_CHANNELS),

kernel\_size=5,

filters=8,

strides=1,

activation=tf.keras.activations.relu,

kernel\_initializer=tf.keras.initializers.VarianceScaling()

))

model.add(tf.keras.layers.MaxPooling2D(

pool\_size=(2, 2),

strides=(2, 2)

)) model.add(tf.keras.layers.Convolution2D(

kernel\_size=5,

filters=16,

strides=1,

activation=tf.keras.activations.relu,

kernel\_initializer=tf.keras.initializers.VarianceScaling()

))

model.add(tf.keras.layers.MaxPooling2D(

pool\_size=(2, 2),

strides=(2, 2)

))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(

units=128,

activation=tf.keras.activations.relu

));

model.add(tf.keras.layers.Dropout(0.2))

model.add(tf.keras.layers.Dense(

units=10,

activation=tf.keras.activations.softmax,

kernel\_initializer=tf.keras.initializers.VarianceScaling()

))

### *5.  Visualization of the model:*

tf.keras.utils.plot\_model(

model,

show\_shapes=True,

show\_layer\_names=True,

)

### *6.* *Confusion matrix of validation dataset:*

confusion\_matrix = tf.math.confusion\_matrix(y\_validation\_re, predictions)

f, ax = plt.subplots(figsize=(9, 7))

sn.heatmap(

confusion\_matrix,

annot=True,

linewidths=.5,

fmt="d",

square=True,

ax=ax

)

plt.show()

### *7.* *Train the model:*

log\_dir=".logs/fit/" + datetime.datetime.now().strftime("%Y%m**%d**-%H%M%S")

tensorboard\_callback = tf.keras.callbacks.TensorBoard(log\_dir=log\_dir, histogram\_freq=1)

training\_history = model.fit(

x\_train\_normalized,

y\_train\_re,

epochs=10,

validation\_data=(x\_validation\_normalized, y\_validation\_re),

callbacks=[tensorboard\_callback]

)

print("The model has successfully trained")

### *8.* *Loss plot curve for training and validation:*

lt.xlabel('Epoch Number')

plt.ylabel('Accuracy')

plt.plot(training\_history.history['loss'], label='training set')

plt.plot(training\_history.history['val\_loss'], label='validation set')

plt.legend()

***9. Final output:***

***import numpy as np***

***import argparse***

***import cv2***

***from cnn.neural\_network import CNN***

***from keras.utils import np\_utils***

***from keras.optimizers import SGD***

***# from sklearn.datasets import fetch\_mldata***

***from sklearn.datasets import fetch\_openml***

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

***# Parse the Arguments***

***ap = argparse.ArgumentParser()***

***ap.add\_argument("-s", "--save\_model", type=int, default=-1)***

***ap.add\_argument("-l", "--load\_model", type=int, default=-1)***

***ap.add\_argument("-w", "--save\_weights", type=str)***

***args = vars(ap.parse\_args())***

***# Read/Download MNIST Dataset***

***print('Loading MNIST Dataset...')***

***# dataset = fetch\_mldata('MNIST Original')***

***dataset = fetch\_openml('mnist\_784')***

***# Read the MNIST data as array of 784 pixels and convert to 28x28 image matrix***

***mnist\_data = dataset.data.reshape((dataset.data.shape[0], 28, 28))***

***mnist\_data = mnist\_data[:, np.newaxis, :, :]***

***# Divide data into testing and training sets.***

***train\_img, test\_img, train\_labels, test\_labels = train\_test\_split(mnist\_data/255.0, dataset.target.astype("int"), test\_size=0.1)***

***# Now each image rows and columns are of 28x28 matrix type.***

***img\_rows, img\_columns = 28, 28***

***# Transform training and testing data to 10 classes in range [0,classes] ; num. of classes = 0 to 9 = 10 classes***

***total\_classes = 10 # 0 to 9 labels***

***train\_labels = np\_utils.to\_categorical(train\_labels, 10)***

***test\_labels = np\_utils.to\_categorical(test\_labels, 10)***

***# Defing and compile the SGD optimizer and CNN model***

***print('\n Compiling model...')***

***sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)***

***clf = CNN.build(width=28, height=28, depth=1, total\_classes=10, Saved\_Weights\_Path=args["save\_weights"] if args["load\_model"] > 0 else None)***

***clf.compile(loss="categorical\_crossentropy", optimizer=sgd, metrics=["accuracy"])***

***# Initially train and test the model; If weight saved already, load the weights using arguments.***

***b\_size = 128 # Batch size***

***num\_epoch = 20 # Number of epochs***

***verb = 1 # Verbose***

***# If weights saved and argument load\_model; Load the pre-trained model.***

***if args["load\_model"] < 0:***

***print('\nTraining the Model...')***

***clf.fit(train\_img, train\_labels, batch\_size=b\_size, epochs=num\_epoch,verbose=verb)***

***# Evaluate accuracy and loss function of test data***

***print('Evaluating Accuracy and Loss Function...')***

***loss, accuracy = clf.evaluate(test\_img, test\_labels, batch\_size=128, verbose=1)***

***print('Accuracy of Model: {:.2f}%'.format(accuracy \* 100))***

***# Save the pre-trained model.***

***if args["save\_model"] > 0:***

***print('Saving weights to file...')***

***clf.save\_weights(args["save\_weights"], overwrite=True)***

***# Show the images using OpenCV and making random selections.***

***for num in np.random.choice(np.arange(0, len(test\_labels)), size=(5,)):***

***# Predict the label of digit using CNN.***

***probs = clf.predict(test\_img[np.newaxis, num])***

***prediction = probs.argmax(axis=1)***

***# Resize the Image to 100x100 from 28x28 for better view.***

***image = (test\_img[num][0] \* 255).astype("uint8")***

***image = cv2.merge([image] \* 3)***

***image = cv2.resize(image, (100, 100), interpolation=cv2.INTER\_LINEAR)***

***cv2.putText(image, str(prediction[0]), (5, 20),cv2.FONT\_HERSHEY\_SIMPLEX, 0.75, (0, 255, 0), 2)***

***# Show and print the Actual Image and Predicted Label Value***

***print('Predicted Label: {}, Actual Value: {}'.format(prediction[0],np.argmax(test\_labels[num])))***

***# cv2.imshow('Digits', image)***

***# cv2.waitKey(0)***