***ASSIGNMENT:-2***

import tensorflow as tf

from tensorflow import keras

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import random

get\_ipython().run\_line\_magic("matplotlib","inline")

mnist = tf.keras.datasets.mnist

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

len(x\_train)

len(x\_test)

x\_train.shape

x\_test.shape

x\_train[0]

plt.matshow(x\_train[11]) #we can change it by changing the argument

x\_train = x\_train/255

x\_test = x\_test/255

x\_train[11]

model = keras.Sequential([

    keras.layers.Flatten(input\_shape=(28, 28)),

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

    keras.layers.Dense(10, activation='softmax')

])

model.summary()

model.compile(optimizer='sgd',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

history=model.fit(x\_train, y\_train,validation\_data=(x\_test,y\_test),epochs=10)

test\_loss, test\_acc=model.evaluate(x\_test,y\_test)

print("Loss=%.3f" %test\_loss)

print("Accuracy=%.3f" %test\_acc)

n=random.randint(0,9999)

plt.imshow(x\_test[n])

plt.show()

predicted\_value=model.predict(x\_test)

print("Handwritten nuber in the image is= %d" %np.argmax(predicted\_value))

get\_ipython().run\_line\_magic('pinfo2','history.history')

history.history.keys()

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Training Loss and accuracy')

plt.ylabel('accuracy/Loss')

plt.xlabel('epoch')

plt.legend(['accuracy', 'val\_accuracy','loss','val\_loss'])

plt.show()

keras\_model\_path="/content/sample\_data"

model.save(keras\_model\_path)

restored\_keras\_model = tf.keras.models.load\_model(keras\_model\_path)

***ASSIGNMENT:-3***

from google.colab import drive

drive.mount("/content/drive")

import numpy as np

import pandas as pd

import os

import random

import  matplotlib.image as mping

import matplotlib.pyplot as plt

import seaborn as sns

import cv2

import tensorflow

from keras.preprocessing.image import ImageDataGenerator

%matplotlib inline

TrainingImagePath="/content/drive/MyDrive/Image /train"

TestImagePath="/content/drive/MyDrive/Image /test"

ValidationImagePath="/content/drive/MyDrive/Image /valid"

train\_datagen = ImageDataGenerator(

    rescale = 1./255,

    shear\_range=0.1,

    zoom\_range=0.1,

    horizontal\_flip=True

)

test\_datagen = ImageDataGenerator(rescale=1./255)

training\_set = train\_datagen.flow\_from\_directory(

    TrainingImagePath,

    target\_size=(128,128),

    batch\_size=32,

    class\_mode="categorical"

)

test\_set = test\_datagen.flow\_from\_directory(

    TestImagePath,

    target\_size = (128,128),

    batch\_size=32,

    class\_mode="categorical"

)

valid\_set = test\_datagen.flow\_from\_directory(

    ValidationImagePath,

    target\_size=(128,128),

    batch\_size=32,

    class\_mode="categorical"

)

def showImages(class\_name):

  random\_index = random.choice(list(range(1,49)))

  folder\_path = os.path.join(TrainingImagePath, class\_name)

  try:

    image\_path = os.path.join(folder\_path,str(random\_index).zfill(3)+".jpg")

    plt.imshow(mping.imread(image\_path))

  except:

    image\_path = os.path.join(folder\_path,str(random\_index).zfill(2)+".jpg")

    plt.imshow(mping.imread(image\_path))

  plt.title(class\_name)

  plt.axis(False)

plt.figure(figsize = (20,20))

for labels,number in training\_set.class\_indices.items():

  plt.subplot(6,6,number+1)

  showImages(labels)

test\_set.class\_indices

'''#################### Creating lookup table for all balls ##############################'''

# class\_indices have the numeric tag for each balls

TrainClasses=training\_set.class\_indices

# Storing the face and the numeric tag for future reference

ResultMap={}

for ballValue,ballName in zip(TrainClasses.values(),TrainClasses.keys()):

    ResultMap[ballValue]=ballName

# Saving the face map for future reference

import pickle

with open(R"E:\Data Sets\Balls Classification\ResultsMap.pkl", 'wb') as f:

    pickle.dump(ResultMap, f, pickle.HIGHEST\_PROTOCOL)

print("Mapping of Face and its ID",ResultMap)

# The number of neurons for the output layer is equal to the number of faces

OutputNeurons=len(ResultMap)

print('\n The Number of output neurons: ', OutputNeurons)

from keras.models import Sequential

from keras.layers import Convolution2D

from keras.layers import MaxPool2D

from keras.layers import Flatten

from keras.layers import Dense

classifier= Sequential()

classifier.add(Convolution2D(32, kernel\_size=(3, 3), strides=(1, 1), input\_shape=(128,128,3), activation='relu'))

classifier.add(MaxPool2D(pool\_size=(2,2)))

''

classifier.add(Convolution2D(64, kernel\_size=(3, 3), strides=(1, 1), activation='relu'))

classifier.add(MaxPool2D(pool\_size=(2,2)))

classifier.add(Flatten())

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

classifier.add(Dense(OutputNeurons, activation='softmax'))

classifier.compile(loss='categorical\_crossentropy', optimizer = 'rmsprop', metrics=["accuracy"])

classifier.summary()

import time

# Measuring the time taken by the model to train

StartTime=time.time()

# Starting the model training

model\_history=classifier.fit\_generator(

                                        training\_set,

                                        steps\_per\_epoch=len(training\_set),

                                        epochs=20,

                                        validation\_data=valid\_set,

                                        validation\_steps=len(valid\_set),

                                        verbose=1)

EndTime=time.time()

print("############### Total Time Taken: ", round((EndTime-StartTime)/60), 'Minutes #############')

accuracy = model\_history.history['accuracy']

val\_accuracy  = model\_history.history['val\_accuracy']

loss = model\_history.history['loss']

val\_loss = model\_history.history['val\_loss']

plt.figure(figsize=(15,10))

plt.subplot(2, 2, 1)

plt.plot(accuracy, label = "Training accuracy")

plt.plot(val\_accuracy, label="Validation accuracy")

plt.legend()

plt.title("Training vs validation accuracy")

plt.subplot(2,2,2)

plt.plot(loss, label = "Training loss")

plt.plot(val\_loss, label="Validation loss")

plt.legend()

plt.title("Training vs validation loss")

plt.show()

***ASSIGNMENT:-4***

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score

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

from tensorflow.keras import layers, losses

from tensorflow.keras.datasets import fashion\_mnist

from tensorflow.keras.models import Model

(x\_train, \_), (x\_test, \_) = fashion\_mnist.load\_data()

x\_train = x\_train/255.

x\_test = x\_test/255.

print(x\_train.shape)

print(x\_test.shape)

latent\_dim = 64

class Autoencoder(Model):

  def \_\_init\_\_(self, latent\_dim):

    super(Autoencoder, self).\_\_init\_\_()

    self.latent\_dim = latent\_dim

    self.encoder = tf.keras.Sequential([

      layers.Flatten(),

      layers.Dense(latent\_dim, activation='relu'),

    ])

    self.decoder = tf.keras.Sequential([

      layers.Dense(784, activation='sigmoid'),

      layers.Reshape((28, 28))

    ])

  def call(self, x):

    encoded = self.encoder(x)

    decoded = self.decoder(encoded)

    return decoded

autoencoder = Autoencoder(latent\_dim)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())

autoencoder.fit(x\_train, x\_train,

                epochs=10,

                shuffle=True,

                validation\_data=(x\_test, x\_test))

encoded\_imgs = autoencoder.encoder(x\_test).numpy()

decoded\_imgs = autoencoder.decoder(encoded\_imgs).numpy()

n = 10

plt.figure(figsize=(20, 4))

for i in range(n):

  # display original

  ax = plt.subplot(2, n, i + 1)

  plt.imshow(x\_test[i])

  plt.title("original")

  plt.gray()

  ax.get\_xaxis().set\_visible(False)

  ax.get\_yaxis().set\_visible(False)

  # display reconstruction

  ax = plt.subplot(2, n, i + 1 + n)

  plt.imshow(decoded\_imgs[i])

  plt.title("reconstructed")

  plt.gray()

  ax.get\_xaxis().set\_visible(False)

  ax.get\_yaxis().set\_visible(False)

plt.show()