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),epoc
hs=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_valu
e))
```

```
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), in
put_shape=(128,128,3), activation='relu'))


classifier.add(MaxPool2D(pool_size=(2,2)))

''
classifier.add(Convolution2D(64, kernel_size=(3, 3), strides=(1, 1), ac
tivation='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 = 'rmspro
p', 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_se
t),
                                       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_sco
re
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()
```