Lab 4

Aim: To build a Convolutional Neural Network and use it to classify faces with images of your own face

```
!unzip /content/Face-Images

import cv2
import numpy as np
import os

# Function to save an image with a given filename and directory
def save_image(image, directory, filename):
    if not os.path.exists(directory):
        os.makedirs(directory)
    cv2.imwrite(os.path.join(directory, filename), image)

# Load the original image
original_image = cv2.imread('/content/Face Images/Final Training
Images/face17/1face17.jpg')
original_image2 = cv2.imread('/content/Face Images/Final Training
Images/face17/2face17.jpg')

# Define the directory to save the generated images
output_directory = '/content/Face Images/Final Training Images/face17'
# Ensure the output directory exists
if not os.path.exists(output_directory):
    os.makedirs(output_directory)

# Save the original image
save_image(original_image, output_directory, '1face17.jpg')
save_image(original_image, output_directory, '2face17.jpg')

# Define transformations
transformations = [
    ("rotate_90_clockwise", cv2.rotate(original_image,
cv2.ROTATE_90_CLOCKWISE)),
    ("rotate_90_counterclockwise", cv2.rotate(original_image,
cv2.ROTATE_90_COUNTERCLOCKWISE)),
    ("horizontal_flip", cv2.flip(original_image, 1)),
    ("vertical_flip", cv2.flip(original_image, 0)),
    ("brightness_increase", cv2.convertScaleAbs(original_image,
alpha=1.2, beta=0)),
    ("brightness_decrease", cv2.convertScaleAbs(original_image,
alpha=0.8, beta=0)),
    ("contrast_increase", cv2.convertScaleAbs(original_image,
alpha=1.0, beta=50)),
    ("contrast_decrease", cv2.convertScaleAbs(original_image,
alpha=1.0, beta=-50)),
    ("rotate_90_clockwise2", cv2.rotate(original_image2,
cv2.ROTATE_90_CLOCKWISE)),
    ("rotate_90_counterclockwise2", cv2.rotate(original_image2,
cv2.ROTATE_90_COUNTERCLOCKWISE)),
    ("horizontal_flip2", cv2.flip(original_image2, 1)),
    ("vertical_flip2", cv2.flip(original_image2, 0)),
    ("brightness_increase2", cv2.convertScaleAbs(original_image2,
alpha=1.2, beta=0)),
    ("brightness_decrease2", cv2.convertScaleAbs(original_image2,
alpha=0.8, beta=0)),
    ("contrast_increase2", cv2.convertScaleAbs(original_image2,
alpha=1.0, beta=50)),
    ("contrast_decrease2", cv2.convertScaleAbs(original_image2,
alpha=1.0, beta=-50))
]

# Save transformed images
for transformation_name, transformed_image in transformations:
    save_image(transformed_image, output_directory,
f'{transformation_name}.jpg')

print("Dataset creation completed.")
```

```
Dataset creation completed.
```

```
import cv2
import numpy as np
import os

# Function to save an image with a given filename and directory
def save_image(image, directory, filename):
    if not os.path.exists(directory):
        os.makedirs(directory)
    cv2.imwrite(os.path.join(directory, filename), image)

# Load the original image
original_image = cv2.imread('/content/Face Images/Final Testing
Images/face17/1face17.jpg')

# Define the directory to save the generated images
output_directory = '/content/Face Images/Final Testing Images/face17'

# Ensure the output directory exists
if not os.path.exists(output_directory):
    os.makedirs(output_directory)

# Save the original image
save_image(original_image, output_directory, 'original.jpg')
```

```
# Define transformations
transformations = [
    ("rotate_90_clockwise", cv2.rotate(original_image,
cv2.ROTATE_90_CLOCKWISE)),
    ("rotate_90_counterclockwise", cv2.rotate(original_image,
cv2.ROTATE_90_COUNTERCLOCKWISE)),
    ("horizontal_flip", cv2.flip(original_image, 1)),
    ("vertical_flip", cv2.flip(original_image, 0)),
    ("brightness_increase", cv2.convertScaleAbs(original_image,
alpha=1.2, beta=0)),
    ("brightness_decrease", cv2.convertScaleAbs(original_image,
alpha=0.8, beta=0)),
    ("contrast_increase", cv2.convertScaleAbs(original_image,
alpha=1.0, beta=50)),
    ("contrast_decrease", cv2.convertScaleAbs(original_image,
alpha=1.0, beta=-50))
]

# Save transformed images
for transformation_name, transformed_image in transformations:
    save_image(transformed_image, output_directory,
f'{transformation_name}.jpg')

print("Dataset creation completed.")
```

```
Dataset creation completed.
```

```
from keras.preprocessing.image import ImageDataGenerator
import warnings
warnings.filterwarnings("ignore")
trainpath='/content/Face Images/Final Training Images'
testpath='/content/Face Images/Final Testing Images'
train_datagen=ImageDataGenerator(shear_range=0.1, zoom_range=0.1,
horizontal_flip=True)

test_datagen = ImageDataGenerator()

training_set = train_datagen.flow_from_directory(trainpath,
target_size=(64, 64), batch_size=32, class_mode='categorical')
#training_set = training_set.repeat()
```

```
Found 262 images belonging to 18 classes.
```

```
test_set = test_datagen.flow_from_directory(testpath, target_size=(64,
64), batch_size=32, class_mode='categorical')
```

```
Found 74 images belonging to 18 classes.
```

```
test_set.class_indices
```

```
{'.ipynb_checkpoints': 0,
 'face1': 1,
 'face10': 2,
 'face11': 3,
 'face12': 4,
 'face13': 5,
 'face14': 6,
 'face15': 7,
 'face16': 8,
 'face17': 9,
 'face2': 10,
 'face3': 11,
 'face4': 12,
 'face5': 13,
 'face6': 14,
 'face7': 15,
 'face8': 16,
 'face9': 17}
```

```
TrainClasses=training_set.class_indices
```

```
ResultMap={}
for faceValue,faceName in
zip(TrainClasses.values(),TrainClasses.keys()):
  ResultMap[faceValue]=faceName
```

```
import pickle
with open("ResultsMap.pkl", 'wb') as fileWriteStream:
  pickle.dump(ResultMap, fileWriteStream)
```

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

```
Mapping of Face and its ID {0: '.ipynb_checkpoints', 1: 'face1', 2:
'face10', 3: 'face11', 4: 'face12', 5: 'face13', 6: 'face14', 7:
'face15', 8: 'face16', 9: 'face17', 10: 'face2', 11: 'face3', 12:
'face4', 13: 'face5', 14: 'face6', 15: 'face7', 16: 'face8', 17:
'face9'}
```

```
OutputNeurons=len(ResultMap)
print('\n The Number of output neurons: ', OutputNeurons)
```

```

 The Number of output neurons:  18
```

```
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=(5, 5), strides=(1, 1),
input_shape=(64,64,3), activation='relu'))
classifier.add(MaxPool2D(pool_size=(2,2)))
classifier.add(Convolution2D(64, kernel_size=(5, 5), strides=(1, 1),
activation='relu'))
classifier.add(MaxPool2D(pool_size=(2,2)))
classifier.add(Flatten())
classifier.add(Dense(64, activation='relu'))
classifier.add(Dense(OutputNeurons, activation='softmax'))
classifier.compile(loss='categorical_crossentropy', optimizer =
'adam', metrics=["accuracy"])

import time
StartTime=time.time()
classifier.fit( training_set, steps_per_epoch=len(training_set),
epochs=20, validation_data=test_set, validation_steps=len(test_set))
EndTime=time.time()
```

```
Epoch 1/20
9/9 [==============================] - 4s 348ms/step - loss: 118.5034
- accuracy: 0.0496 - val_loss: 4.5132 - val_accuracy: 0.0541
Epoch 2/20
9/9 [==============================] - 4s 411ms/step - loss: 2.8358 -
accuracy: 0.1603 - val_loss: 2.4857 - val_accuracy: 0.3108
Epoch 3/20
9/9 [==============================] - 3s 294ms/step - loss: 1.9974 -
accuracy: 0.3931 - val_loss: 1.4749 - val_accuracy: 0.6081
Epoch 4/20
9/9 [==============================] - 3s 305ms/step - loss: 1.3570 -
accuracy: 0.6374 - val_loss: 0.8874 - val_accuracy: 0.8243
Epoch 5/20
9/9 [==============================] - 4s 450ms/step - loss: 0.8857 -
accuracy: 0.7481 - val_loss: 0.8589 - val_accuracy: 0.7973
Epoch 6/20
9/9 [==============================] - 3s 299ms/step - loss: 0.4347 -
accuracy: 0.9008 - val_loss: 0.5785 - val_accuracy: 0.7838
Epoch 7/20
9/9 [==============================] - 3s 294ms/step - loss: 0.2674 -
accuracy: 0.9160 - val_loss: 0.1768 - val_accuracy: 0.9324
Epoch 8/20
9/9 [==============================] - 4s 371ms/step - loss: 0.1745 -
accuracy: 0.9504 - val_loss: 0.1275 - val_accuracy: 0.9459
Epoch 9/20
9/9 [==============================] - 3s 303ms/step - loss: 0.0800 -
accuracy: 0.9847 - val_loss: 0.0865 - val_accuracy: 0.9730
Epoch 10/20
9/9 [==============================] - 3s 365ms/step - loss: 0.2723 -
accuracy: 0.9084 - val_loss: 0.4080 - val_accuracy: 0.8243
Epoch 11/20
9/9 [==============================] - 3s 289ms/step - loss: 0.3730 -
```

```
accuracy: 0.8740 - val_loss: 0.3144 - val_accuracy: 0.9459
Epoch 12/20
9/9 [==============================] - 3s 290ms/step - loss: 0.2623 -
accuracy: 0.9237 - val_loss: 0.4821 - val_accuracy: 0.8919
Epoch 13/20
9/9 [==============================] - 3s 330ms/step - loss: 0.6508 -
accuracy: 0.8244 - val_loss: 0.3124 - val_accuracy: 0.9189
Epoch 14/20
9/9 [==============================] - 4s 434ms/step - loss: 0.3142 -
accuracy: 0.8931 - val_loss: 0.1478 - val_accuracy: 0.9459
Epoch 15/20
9/9 [==============================] - 3s 301ms/step - loss: 0.1295 -
accuracy: 0.9771 - val_loss: 0.1509 - val_accuracy: 0.9595
Epoch 16/20
9/9 [==============================] - 3s 292ms/step - loss: 0.0998 -
accuracy: 0.9771 - val_loss: 0.1307 - val_accuracy: 0.9459
Epoch 17/20
9/9 [==============================] - 3s 289ms/step - loss: 0.0427 -
accuracy: 0.9885 - val_loss: 0.1523 - val_accuracy: 0.9459
Epoch 18/20
9/9 [==============================] - 3s 297ms/step - loss: 0.0455 -
accuracy: 0.9885 - val_loss: 0.1080 - val_accuracy: 0.9595
Epoch 19/20
9/9 [==============================] - 3s 324ms/step - loss: 0.0204 -
accuracy: 0.9962 - val_loss: 0.0754 - val_accuracy: 0.9730
Epoch 20/20
9/9 [==============================] - 3s 296ms/step - loss: 0.0159 -
accuracy: 1.0000 - val_loss: 0.0366 - val_accuracy: 0.9865
```

```
import numpy as np
from keras.preprocessing import image
ImagePath='/content/Face Images/Final Testing
Images/face12/1face12.jpg'
test_image=image.load_img(ImagePath,target_size=(64, 64))
test_image=image.img_to_array(test_image)
test_image=np.expand_dims(test_image,axis=0)
result=classifier.predict(test_image,verbose=0)
print('Prediction is: ',ResultMap[np.argmax(result)])
```

```
Prediction is:  face12
```

```
ImagePath='/content/Face Images/Final Testing
Images/face17/1face17.jpg'
test_image=image.load_img(ImagePath,target_size=(64, 64))
test_image=image.img_to_array(test_image)
test_image=np.expand_dims(test_image,axis=0)
result=classifier.predict(test_image,verbose=0)
print('Prediction is: ',ResultMap[np.argmax(result)])
```

```
Prediction is:  face17
```