APPENDIX

DNN Implementation Code

```python
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split
import tensorflow as tf
import time
import matplotlib.pyplot as plt
import numpy as np
from tensorflow import keras as k
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from sklearn import preprocessing
from keras.regularizers import l2
from sklearn import metrics
from sklearn.metrics import f1_score

def main():
X,y = fetch_olivetti_faces(return_X_y=True)
print(X.shape, y.shape)
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.10, stratify=y, random_state=42)
train_X = preprocessing.scale(train_X)
test_X = preprocessing.scale(test_X)
print(train_X.shape, train_y.shape)
print(test_X.shape, test_y.shape)

model = Sequential([
Dense(units=200, input_dim=4096, kernel_regularizer=l2(0.0001), activation='relu'),
Dropout(0.2),
Dense(units=200, input_dim=200, kernel_regularizer=l2(0.0001), activation='relu'),
Dropout(0.2),
Dense(units=200, input_dim=200, kernel_regularizer=l2(0.0001), activation='relu'),
Dropout(0.1),
Dense(units=200, input_dim=200, kernel_regularizer=l2(0.0001), activation='relu'),
Dropout(0.1),
Dense(units=40, input_dim=200, activation='softmax'),])
model.summary()
model.compile(loss='sparse_categorical_crossentropy',
optimizer='rmsprop',
metrics=['accuracy'])

print("Starting training ")
start = time.perf_counter()
num_epochs = 100
h = model.fit(train_X, train_y, batch_size=32, epochs=num_epochs, validation_split = 0.1, verbose=1)
finish = time.perf_counter()
print(f"Training finished in {finish - start:0.4f} seconds\n")

```

```
print(h.history.keys()) # dict_keys(['accuracy', 'loss'])

print("Training history: ")
for i in range(num_epochs):
loss = h.history['loss'][i]
accuracy = h.history['accuracy'][i] * 100
print("epoch(s): %5d loss = %0.4f accuracy = %0.2f%%" \
% (i, loss, accuracy))

eval = model.evaluate(test_X, test_y, verbose=0)
print("\nEvaluation on test data: \nloss = %0.4f \
accuracy = %0.2f%%" % (eval[0], eval[1]*100) )
plt.plot(h.history['accuracy'])
plt.plot(h.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.show()

plt.plot(h.history['loss'])
plt.plot(h.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()

test_class = np.argmax(model.predict(test_X), axis=-1)
print("Confussion matrix:\n%s" %
metrics.confusion_matrix(test_y, test_class))
print("Classification report:\n%s" %
metrics.classification_report(test_y, test_class))
print("Classification accuracy: %f" %
metrics.accuracy_score(test_y, test_class))
model.save('dnn_model.h5')
if __name__=="__main__":
main()
```

PCA Implementation Code

```
from matplotlib import pyplot as plt
from matplotlib.image import imread
import numpy as np
import os
import time
from google.colab import drive
from google.colab import files
mount= drive.mount('/content/drive')
dataset_path = 'drive/MyDrive/orl/'
dataset_dir = os.listdir(dataset_path)
print(dataset_dir)
train_images = []
test_images = []
width = 80
height = 70

i=1
for j in range(len(dataset_dir)):
img_name = (str(j+1)+'_'+str(i)+'.jpg')
if (j+1)%10==0:
test_images.append(img_name)
i+=1
else:
train_images.append(img_name)
# print(train_images)
# print(test_images)

training = np.ndarray(shape=(len(train_images), height*width), dtype=np.float64)
for i in range(len(train_images)):
img = plt.imread(dataset_path + train_images[i])
training[i,:] = np.array(img, dtype='float64').flatten()
if i<12:
plt.subplot(3,4,1+i)
plt.subplots_adjust(right=1.5, top=1.5)
plt.imshow(img, cmap='gray')
plt.show()

testing = np.ndarray(shape=(len(test_images), height*width), dtype=np.float64)
# print(len(test_images))
for j in range(len(test_images)):
img = plt.imread(dataset_path + test_images[j])
testing[j,:] = np.array(img, dtype='float64').flatten()
plt.subplot(9,5,1+j)
plt.subplots_adjust(right=1.2, top=1.2)
plt.imshow(img, cmap='gray')
plt.show()
print(training)

#meanface
mean_face = np.zeros((1,height*width))
print(mean_face)

for i in training:
mean_face = np.add(mean_face,i)
```

```
mean_face = np.divide(mean_face,len(train_images)).flatten()

plt.imshow(mean_face.reshape(height, width), cmap='gray')
plt.show()

#normalized
normalised_training = np.ndarray(shape=(len(train_images), height*width))

for i in range(len(train_images)):
normalised_training[i] = np.subtract(training[i],mean_face)

for i in range(len(train_images)):
img = normalised_training[i].reshape(height,width)
if i<12:
plt.subplot(3,4,1+i)
plt.imshow(img, cmap='gray')
plt.show()

#covariance
cov_matrix = np.cov(normalised_training)

#eigen
eigenvalues, eigenvectors, = np.linalg.eig(cov_matrix)

print(eigenvalues)

#sort and pairing
eigen_pairs = [(eigenvalues[index], eigenvectors[:,index]) for index in range(len(eigenvalues))]
eigen_pairs.sort(reverse=True)
sort_eigvalues = [eigen_pairs[index][0] for index in range(len(eigenvalues))]
sort_eigvectors = [eigen_pairs[index][1] for index in range(len(eigenvalues))]

#select k eigenfaces
reduced_data = np.array(sort_eigvectors[:7]).transpose()

proj_data = np.dot(training.transpose(),reduced_data)
proj_data = proj_data.transpose()

for i in range(proj_data.shape[0]):
img = proj_data[i].reshape(height,width)
plt.subplot(2,4,1+i)
plt.imshow(img, cmap='gray')
plt.show()

#find weight
w = np.array([np.dot(proj_data,i) for i in normalised_training])
w

#recognize test images
count = 0
num_images = 0
correct_pred = 0
def recogniser(img, train_images,proj_data,w):
```

```
global count,highest_min,num_images,correct_pred
unknown_face = plt.imread('drive/MyDrive/orl/'+img)
num_images += 1
unknown_face_vector = np.array(unknown_face, dtype='float64').flatten()
normalised_uface_vector = np.subtract(unknown_face_vector,mean_face)
plt.subplot(80,10,1+count)
plt.imshow(unknown_face, cmap='gray')
plt.title('Input:'+'.'.join(img.split('.')[:2]))
plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
count+=1
w_unknown = np.dot(proj_data, normalised_uface_vector)
diff = w - w_unknown
norms = np.linalg.norm(diff, axis=1)
index = np.argmin(norms)
min(norms)
t1 = 99999999999
t0 = 99999999999
if norms[index] < t1:
plt.subplot(80,10,1+count)
if norms[index] < t0: # Face is found
if img.split('_')[1] == train_images[index].split('_')[1]:
plt.title('Matched:'+'.'.join(train_images[index].split('.')[:2]), color='g')
plt.imshow(imread('drive/MyDrive/orl/'+train_images[index]), cmap='gray')
correct_pred += 1
else:
plt.title('Mismatched:'+'.'.join(train_images[index].split('.')[:2]), color='b')
plt.imshow(imread('drive/MyDrive/orl/'+train_images[index]), cmap='gray')
plt.subplots_adjust(right=1.2, top=2.5)
count+=1
fig = plt.figure(figsize=(15, 15))
for i in range(len(test_images)):
recogniser(test_images[i], train_images,proj_data,w)

plt.show()

print('Correct predictions: {}/{} = {}%'.format(correct_pred, num_images, correct_pred/num_images*100.00))
```

PLAGIARISM CHECK.ORG

8.82% PLAGIARISM APPROXIMATELY

Report #14310419

1. CHAPTER 1 INTRODUCTION 1.1. Background Recognizing faces sounds like a simple thing to humans. We can easily recognize someone in person or maybe through pictures or videos. Nevertheless, it is not that easy for computer vision to do it. Back in the day, we could only see the use of face recognition technology on television or maybe in movies, but now facial recognition technology is commonly used in various fields. Our smartphone is one such example, and almost every phone has the face unlock feature nowadays. Because of that, so many algorithms are developed in order to find the most optimum in terms of time, speed, and costs. One of them is Neural Networks. There has been a surge of interest in neural networks, particularly deep and large networks. These networks have exhibited impressive results [1]. However, besides the significant advantages, beginner researchers have one problem: The approach is computationally expensive and requires a high degree of correlation between the test and training