程序报告

学号：2212046　　姓名：王昱

一、问题重述

1.1 实验背景

本实验采用特征脸（Eigenface）算法进行人脸识别。

特征脸（eigenface）是第一种有效的人脸识别方法，通过在一大组描述不同人脸的图像上进行主成分分析（PCA）获得。 本次实验要求大家构建一个自己的人脸库（建议）：大家可以选择基于 ORL 人脸库添加自己搜集到的人脸图像形成一个更大的人脸库，要求人脸库中的每一张图像都只包含一张人脸且眼睛的中心位置对齐(通过裁剪或缩放，使得每张人脸图像大小尺寸一致且人脸眼睛的中心位置对齐)。为了方便同学们操作，大家也可以选择直接基于 ORL 人脸库进行本次实验。

1.2 实验内容

在模型训练过程中，首先要根据测试数据求出平均脸，然后将前 K 个特征脸保存下来，利用这 K 个特征脸对测试人脸进行识别，此外对于任意给定的一张人脸图像，可以使用这 K 个特征脸对原图进行重建。

1.3 实验要求

- 求解人脸图像的特征值与特征向量构建特征脸模型
- 利用特征脸模型进行人脸识别和重建，比较使用不同数量特征脸的识别与重建效果
- 使用 Python 语言。

==========

1.2 对问题的理解：

本实验首先需要收集人脸图像，建立一个包含多个人脸的数据库，确保图像尺寸一致和眼睛位置对齐。随后通过主成分分析（PCA）计算出平均脸，并找出前 K 个特征向量作为特征脸。再进行人脸识别与重建，使用这些特征脸来识别新的人脸图像，并尝试重建原始人脸图像。主要利用 Python 中的科学计算库来处理数据和图像，执行 PCA，以及评估不同特征脸数量对识别和重建效果的影响。

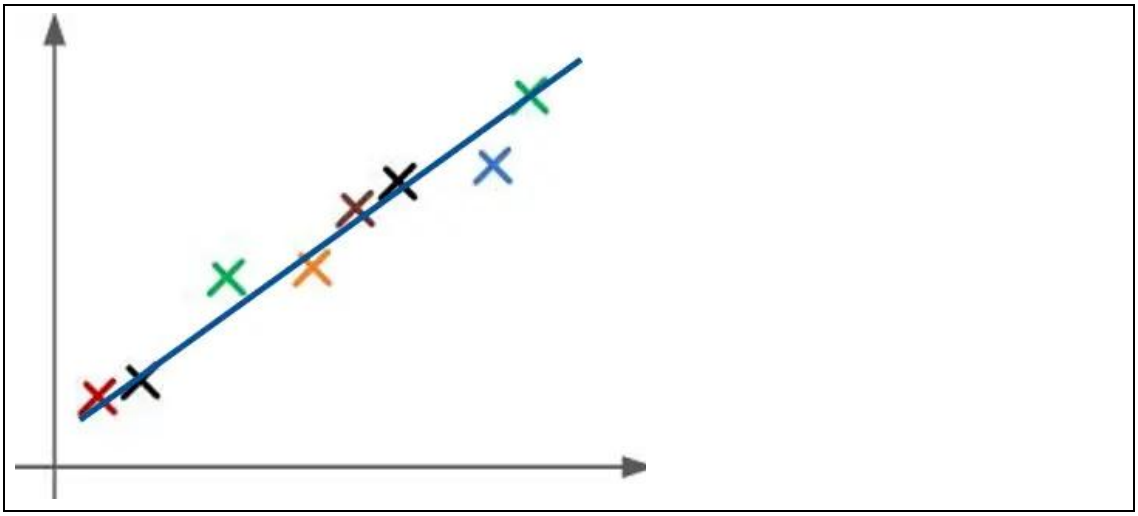
==========

1.3 实验方法 PCA 原理：

1.找到一组新的坐标轴，或者说是一组新的基（basis），用于表示原来的数据，使得在表示数据时不同轴是不相关的（即协方差为 0）。

2.取出其中含有信息较多（即方差较大）的坐标轴（基），构成（span）一个新的空间，舍弃其他维度的信息。

3.由于新空间的维度小于原来的空间，所以把数据投影到新的空间后，可以大大降低数据的复杂度（虽然会损失少量信息）。



二、设计思想

2.1 数据预处理

2.1.1 导入必要的包

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
from PIL import Image
import os
from sklearn.preprocessing import normalize
```

首先代码开始部分导入了进行图像处理和矩阵运算需要的 python 库

2.1.2 划分数据集

```
def spilt_data(nPerson, nPicture, data, label):
```

- 这个函数的作用是将数据集分割为训练集和测试集，在特征脸人脸识别实验中准备数据，确保训练和测试数据的合理划分。
- 参数说明：

1. nPerson：志愿者的数量。
2. nPicture：每个志愿者用于训练的照片数量。
3. data：包含图像数据的数组。
4. label：对应数据集的标签。

```
train = data[:nPerson,:nPicture,:,:].reshape(nPerson*nPicture, rows*cols)
rain_label = label[:nPerson, :nPicture].reshape(nPerson * nPicture)：
```

- 从数据集中选择前 nPerson 个志愿者，并从每个志愿者中选取前 nPicture 张照片作为训练集。从标签数据中选择前 nPerson 个志愿者，并从每个志愿者中选取前 nPicture 个标签作为训练集标签。

```
test = data[:nPerson, nPicture:, :, :].reshape(nPerson*(allPicture - nPicture), rows*cols)
test_label = label[:nPerson, nPicture:].reshape(nPerson * (allPicture - nPicture))
```

- 从数据集中选择前 nPerson 个志愿者，并从每个志愿者中选取剩余的照片作为测试集。从标签数据中选择前 nPerson 个志愿者，并从每个志愿者中选取剩余的标签作为测试集标签。

```
datapath = './ORL.npz'
ORL = np.load(datapath)
```

data = ORL['data']
label = ORL['label']
num_eigenface = 200
train_vectors, train_labels, test_vectors, test_labels = spilt_data(40, 5, data,label)
train_vectors = train_vectors / 255
test_vectors = test_vectors / 255

- 打开了数据路径 ORL.npz ，然后分别读取的数据和标签，将其存储到 train_vectors , train_label , test_vectors , test_label 中，为后面的训练做准备。

==========

2.2 训练特征脸算法实现

2.2.1：对于每个样本 x × i 进行中心化处理：$x_i = x_i - \mu, \mu = \frac{1}{n}\sum \times j = 1^n x_j$

avg_img = np.mean(trainset, axis=0)
norm_img = trainset - avg_img

这部分计算首先训练集中所有人脸图像的平均脸，对平均人脸进行聚类。这个平均脸代表了训练集中所有人脸的一般特征。之后用测试人脸数据减去平均人脸得到差值矩阵。

2.2.2: 计算原始人脸样本数据的协方差矩阵：$\sum = \frac{1}{n-1} X^T X$

covariance_matrix = np.dot(norm_img.T, norm_img)

计算中心化人脸数据的协方差矩阵，找出数据中的主要变化方向。

2.2.3: 进行特征值分解，对所得特征根从到小排序

eigenvalue, featurevector = np.linalg.eig(covariance_matrix)

特征值分解，获得转换之后的特征值、特征向量。特征向量代表了数据变化的主要方向，而特征值表示这些方向的重要性。

2.2.4: 取前 l 个最大特征根所对应特征向量 w1,w2,…,wl 组成映射矩阵 W

eigenvalue, featurevector = np.linalg.eig(covariance_matrix)
k_min = min(featurevector.shape[1], k)
feature_vector = np.mat(norm_img).T * np.mat(featurevector[:,:k_min])

这里，featurevector[:,:k_min] 就是取前（k）个最大特征根对应的特征向量，然后通过 np.mat(norm_img).T * np.mat(featurevector[:,:k_min]) 计算得到映射矩阵（W）。这个映射矩阵（W）用于将高维的人脸数据映射到低维的特征空间中，这样就可以用更少的数据来表示每张人脸，同时保留最重要的特征信息。

2.2.5：将每个人脸图像 xi 按照如下方法降维：$(x_i) \times 1 \times d(W)_{d \times l} = 1 \times l$

feature = np.mat(norm_img.T) * np.mat(norm_img) * np.mat(feature_vector)
feature = np.array(feature).T

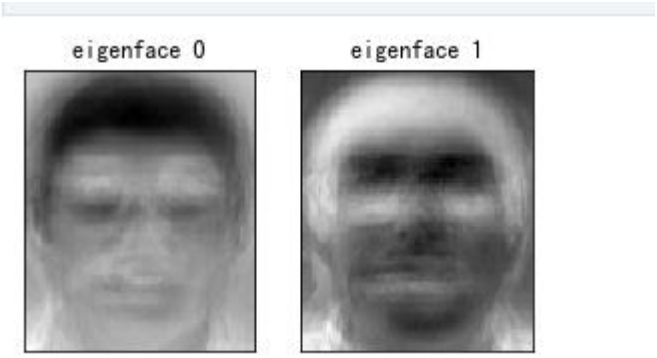
这段代码首先计算了中心化人脸数据的转置 norm_img.T 和中心化人脸数据 norm_img 的乘积，然后再与特征向量 feature_vector 相乘。这个过程实际上是在构建一个新的特征空间，其中的特征向量是从原始数据中提取出来的最重要的特征。

降维是通过将高维的人脸数据投影到由特征向量定义的低维空间来实现的。feature_vector 包含了最重要的特征向量，它们定义了一个新的特征空间，这个空间的维度比原始的像素空间要小得多。

最后，np.array(feature).T 将得到的特征矩阵转置，以便于后续的处理和使用。这样，每一列就代表了一张中心化人脸图像在低维特征空间中的表示，完成了降维过程。这个低维特征空间通常用于人脸识别和分类任务中，因为它能够有效地表示人脸的主要特征，同时去

除了不必要的信息，如背景噪声等。

2.2.6 打印两张特征人脸作为展示



==========

2.3 人脸识别模型

```
def rep_face(image, avg_img, eigenface_vects, numComponents=0):
    difference_image = np.array(image) - np.array(avg_img)
    num = min(eigenface_vects.shape[0], numComponents)
    eigenface_vect = normalize(np.array(eigenface_vects[:num, :]), norm = 'l2')
    linear_space = eigenface_vect.T
    coordinate = np.mat(difference_image) * np.mat(linear_space)
    representation = coordinate
    numEigenFaces = numComponents
    return representation, numEigenFaces
```

这段代码的目的是将一张人脸图像转换成一组数值，这组数值可以用于识别或比较人脸。通过将图像投影到由特征脸定义的较低维度的空间来实现，这种表示方法用于后续的相似度比较任务。此处我们使用特征脸算法对测试集中的人脸照片进行预测，我们在这里定义了 rep_face 函数，其输入是测试数据，训练集的平均人脸数据，特征脸向量，选用的特征脸数量，最后输出我们的人脸识别的准确率，最后我们也是达到了 90.25%的正确率。

```
train_reps = []
for img in train_vectors:
    train_rep, _ = rep_face(img, avg_img, eigenface_vects, num_eigenface)
    train_reps.append(train_rep)

num = 0
for idx, image in enumerate(test_vectors):
    label = test_labels[idx]
    test_rep, _ = rep_face(image, avg_img, eigenface_vects, num_eigenface)

    results = []
    for train_rep in train_reps:
        similarity = np.sum(np.square(train_rep - test_rep))
        results.append(similarity)
    results = np.array(results)

    if label == np.argmin(results) // 5 + 1:
        num = num + 1

print("人脸识别准确率: {}%".format(num / 80 * 100))
```

```
人脸识别准确率: 90.0%
```

==========

2.4 人脸重构

2.4.1 重建图像

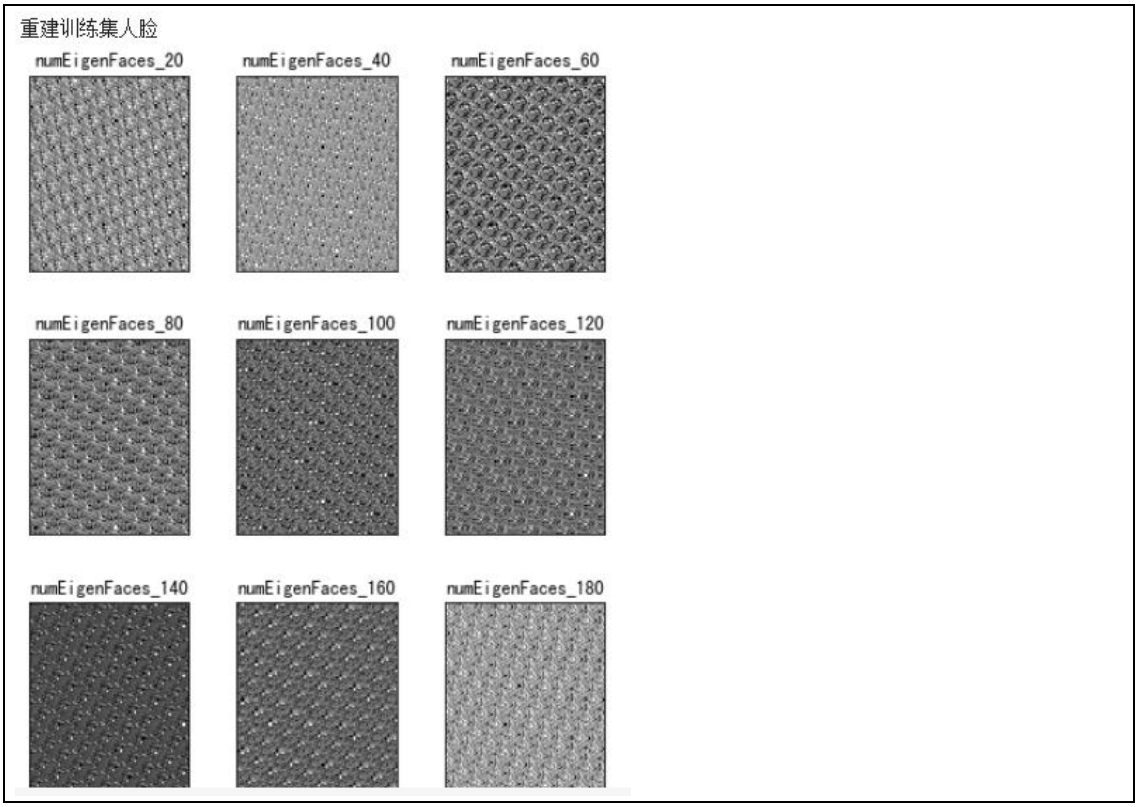
face = np.dot(representations, eigenVectors[0:numComponents,])+avg_img

这部分是矩阵乘法，它将人脸的特征脸表示（representations）与前 numComponents 个特征脸向量（eigenVectors）相乘。这个操作的结果是一个向量，它是原始图像空间中的一个点，代表了重建的人脸图像。由于在创建特征脸表示时减去了平均图像，所以在重建过程中需要加回来，以恢复原始的亮度和对比度。

2.4.2 重建结果

```
print("重建训练集人脸")
image = train_vectors[100]
faces = []
names = []
for i in range(20, 200, 20):
    representations, numEigenFaces = rep_face(image, avg_img, eigenface_vects, i)
    face, name = recFace(representations, avg_img, eigenface_vects, numEigenFaces)
    faces.append(face)
    names.append(name)
plot_gallery(faces, names, n_row=3, n_col=3)
print("-"*55)
print("重建测试集人脸")
image = test_vectors[54]
faces = []
names = []
for i in range(20, 200, 20):
    representations, numEigenFaces = rep_face(image, avg_img, eigenface_vects, i)
    face, name = recFace(representations, avg_img, eigenface_vects, numEigenFaces)
    faces.append(face)
    names.append(name)
plot_gallery(faces, names, n_row=3, n_col=3)
```

此处对其进行重建测试：



三、代码内容

代码内容：

```
# 导入必要的包
import matplotlib.pyplot as plt
import numpy as np
import cv2
from PIL import Image
import os
from sklearn.preprocessing import normalize

def spilt_data(nPerson, nPicture, data, label):
    """
    分割数据集

    :param nPerson :  志愿者数量
    :param nPicture:  各志愿者选入训练集的照片数量
    :param data :  等待分割的数据集
    :param label:  对应数据集的标签
    :return:  训练集,  训练集标签,  测试集,  测试集标签
    """
    # 数据集大小和意义
    allPerson, allPicture, rows, cols = data.shape

    # 划分训练集和测试集
```

```
    train = data[:nPerson,:nPicture,:,:].reshape(nPerson*nPicture, rows*cols)
    train_label = label[:nPerson, :nPicture].reshape(nPerson * nPicture)
    test = data[:nPerson, nPicture:, :, :].reshape(nPerson*(allPicture - nPicture), rows*cols)
    test_label = label[:nPerson, nPicture:].reshape(nPerson * (allPicture - nPicture))

    # 返回: 训练集, 训练集标签, 测试集, 测试集标签
    return train, train_label, test, test_label

def plot_gallery(images, titles, n_row=3, n_col=5, h=112, w=92):   # 3 行 4 列
    """
    展示多张图片

    :param images: numpy array 格式的图片
    :param titles: 图片标题
    :param h: 图像 reshape 的高
    :param w: 图像 reshape 的宽
    :param n_row: 展示行数
    :param n_col: 展示列数
    :return:
    """
    # 展示图片
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())
    plt.show()

# 在生成 main 文件时, 请勾选该模块

def eigen_train(trainset, k=20):
    """
    训练特征脸（eigenface）算法的实现

    :param trainset: 使用 get_images 函数得到的处理好的人脸数据训练集
    :param K: 希望提取的主特征数
    :return: 训练数据的平均脸, 特征脸向量, 中心化训练数据
    """

    # 平均人脸获取
    avg_img = np.mean(trainset, axis=0)
```

```
    # 获得差值矩阵
    norm_img = trainset - avg_img
    norm_img_t = norm_img.T                                    #差值矩阵的转置(10304, 200)
    covariance_matrix = np.mat(norm_img) * np.mat(norm_img_t)
    eigenvalue, featurevector = np.linalg.eig(covariance_matrix)
    k_min = min(featurevector.shape[1], k)
    feature_vector = np.mat(norm_img_t) * np.mat(featurevector[:,:k_min])
    feature = np.mat(norm_img_t) * np.mat(norm_img) * np.mat(feature_vector)
    feature = np.array(feature).T
    return avg_img, feature, norm_img

import numpy as np
from sklearn.preprocessing import normalize

def rep_face(image, avg_img, eigenface_vects, numComponents=0):
    """
    使用特征脸（eigenface）算法对输入图像进行投影映射，得到使用特征脸向量表示的数据。

    :param image: 输入图像（一维数组）
    :param avg_img: 训练集的平均人脸图像（一维数组）
    :param eigenface_vects: 特征脸向量（二维数组）
    :param numComponents: 选用的特征脸数量（如果为 0，则使用所有特征脸）
    :return: 输入图像的特征向量表示，最终使用的特征脸数量
    """
    difference_image = np.array(image) - np.array(avg_img)
    num = min(eigenface_vects.shape[0], numComponents)
    eigenface_vect = normalize(np.array(eigenface_vects[:num, :]), norm = 'l2')
    linear_space = eigenface_vect.T
    coordinate = np.mat(difference_image) * np.mat(linear_space)
    representation = coordinate
    numEigenFaces = numComponents
    return representation, numEigenFaces

# 在生成 main 文件时，请勾选该模块

def recFace(representations, avg_img, eigenVectors, numComponents, sz=(112,92)):
    """
    利用特征人脸重建原始人脸

    :param representations: 表征数据
    :param avg_img: 训练集的平均人脸数据
    :param eigenface_vects: 特征脸向量
```

```
    :param numComponents:  选用的特征脸数量
    :param sz:  原始图片大小
    :return:  重建人脸, str  使用的特征人脸数量
    """
    face = np.dot(representations, eigenVectors[0:numComponents,])+avg_img


    # 返回:  重建人脸, str  使用的特征人脸数量
    return face, 'numEigenFaces_{}'.format(numComponents)
```

四、实验结果

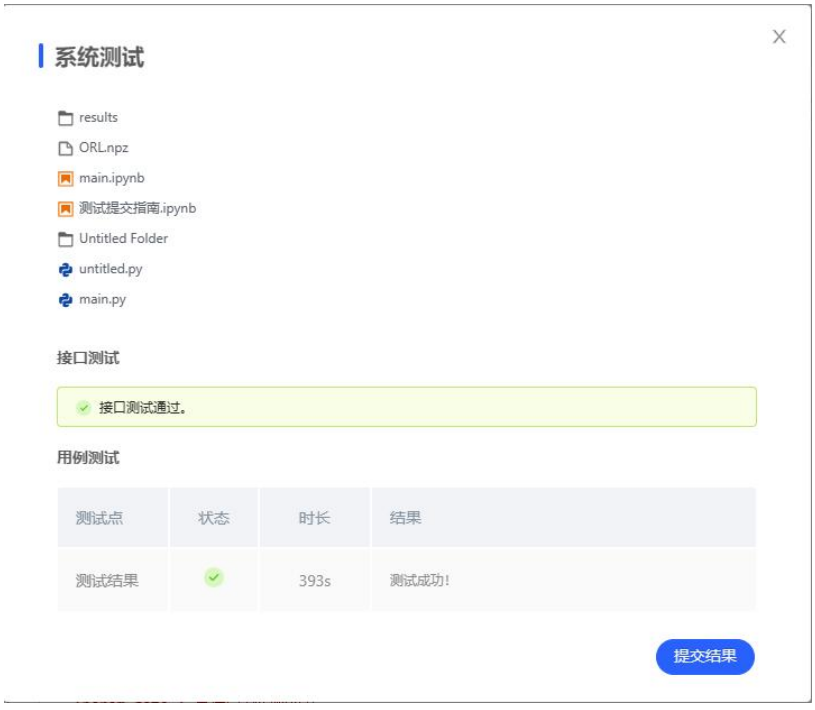
达到了 90%的正确率：

```
train_reps = []
for img in train_vectors:
    train_rep, _ = rep_face(img, avg_img, eigenface_vects, num_eigenface)
    train_reps.append(train_rep)

num = 0
for idx, image in enumerate(test_vectors):
    label = test_labels[idx]
    test_rep, _ = rep_face(image, avg_img, eigenface_vects, num_eigenface)

    results = []
    for train_rep in train_reps:
        similarity = np.sum(np.square(train_rep - test_rep))
        results.append(similarity)
    results = np.array(results)

    if label == np.argmin(results) // 5 + 1:
        num = num + 1

print("人脸识别准确率: {}%".format(num / 80 * 100))
```

```
人脸识别准确率: 90.0%
```

==========

代码在平台上测试成功！



五、总结

特征脸方法是一种强大的人脸识别技术，它通过将高维的图像数据转换为低维的特征表示来简化问题。这种方法不仅可以用于识别和验证人脸，还可以用于图像压缩和重建。本次实验展示了特征脸方法的基本步骤和应用，是人脸识别领域的一个重要实践。本次实验对特征脸识别 PCA 有了初步的认识，通过对 PPT 的学习以及网上资料的查阅成功完成了这次实验，受益良多。