**程序报告**

学号：2212998 姓名：胡博浩

1. **问题重述**

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。

**实验要求：**

1）构建人脸库

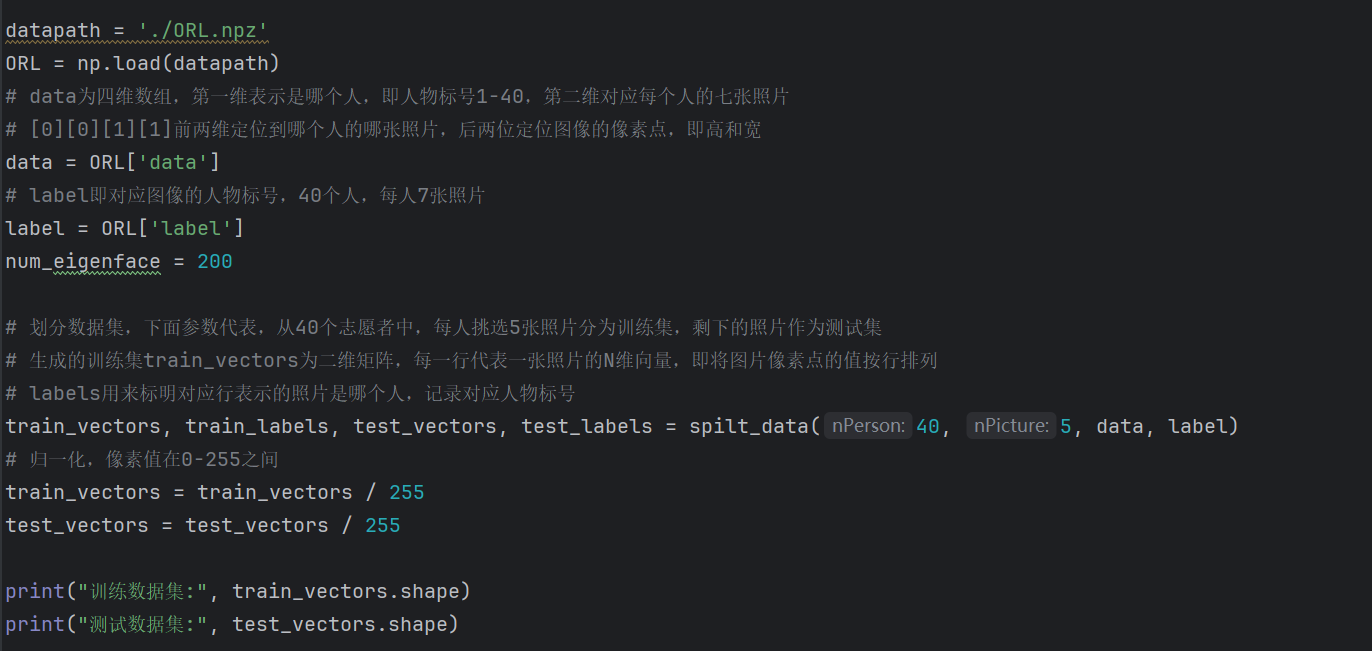
2）求解人脸图像的特征值与特征向量构建特征脸模型

3）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

1. **设计思想**

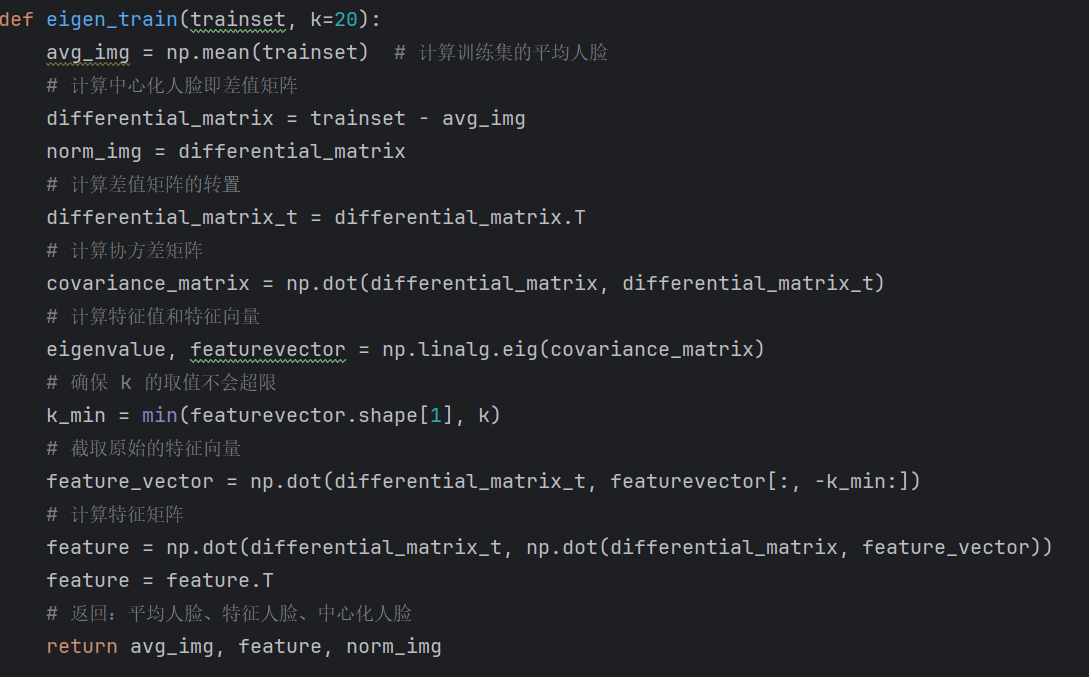
1.数据预处理：

首先加载数据集，并定义特征脸数量。然后划分数据集：针对实验中给出的ORL数据集，我们选用每名志愿者的前五张图片作为训练集，后两张图片作为测试集。



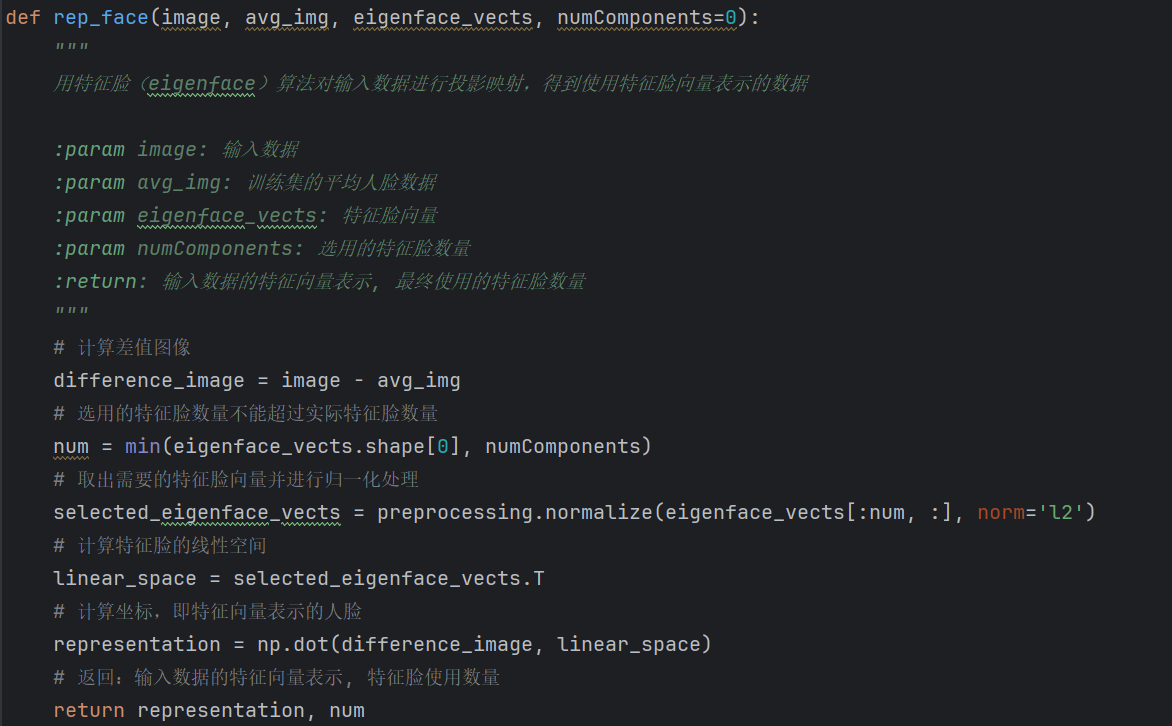
2.人脸特征提取：

算法先根据测试数据求出平均脸，然后计算训练数据里每张脸与平均脸的差异，求差异矩阵的特征值和特征向量，取前 K 个特征向量，计算出 K 张特征脸，然后就可以利用这 K 个特征脸对测试人脸进行识别了。



3.人脸识别：

使用特征脸（Eigenface）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，其输入是策树数据, 训练集的平均人脸数据，特征脸向量, 选用的特征脸数量。通过这个函数，将输入人脸图像投影到特征空间中，以便进行后续的人脸识别或验证任务。



4.人脸重建：

可以理解为识别过程空间转换的逆过程，即利用特征脸的线性空间，将特征空间中的表示重新转换为图像表示，从而重建原始人脸。



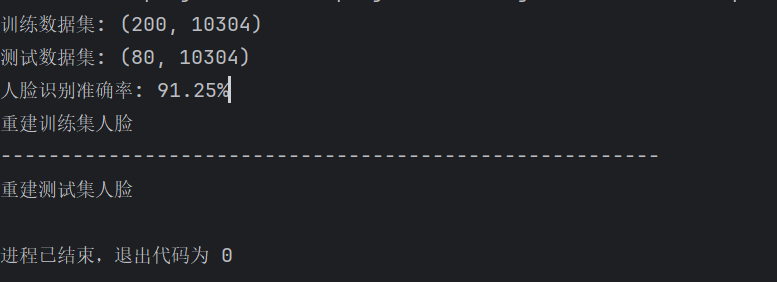
1. **代码内容**

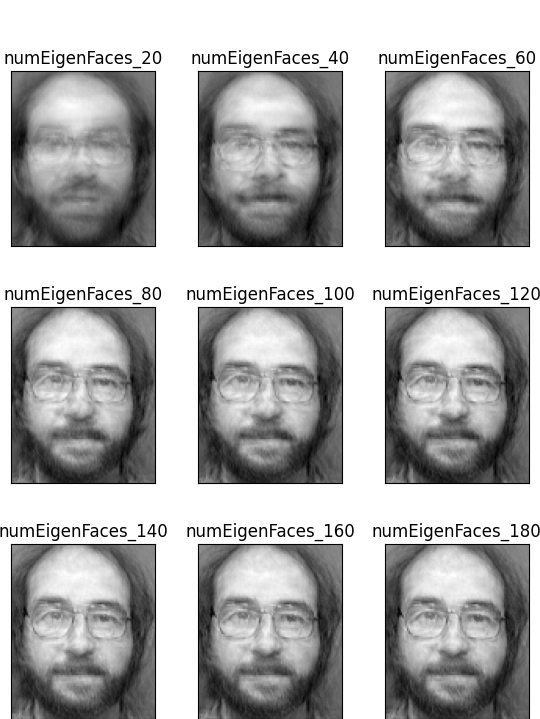
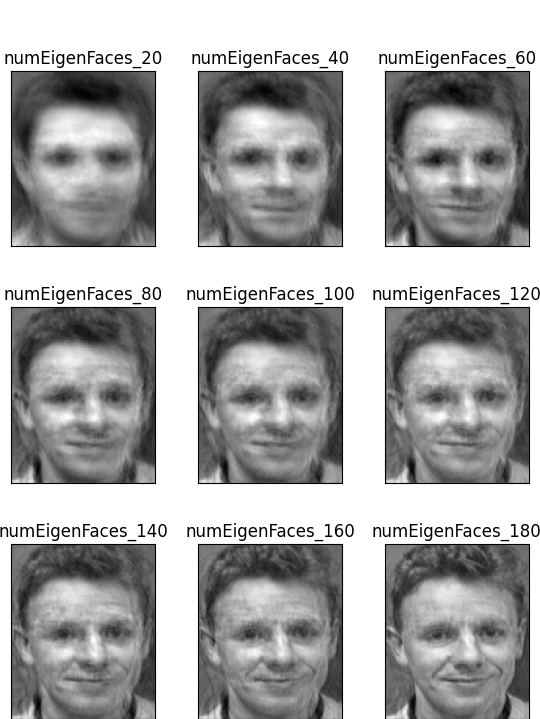
# 导入必要的包  
import matplotlib.pyplot as plt  
import numpy as np  
import cv2  
from sklearn import preprocessing  
from PIL import Image  
import os  
  
  
def spilt\_data(nPerson, nPicture, data, label):  
 *"""  
 分割数据集  
  
 :param nPerson : 志愿者数量  
 :param nPicture: 各志愿者选入训练集的照片数量  
 :param data : 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 show\_img(img, h=112, w=92):  
 *"""  
 展示单张图片  
  
 :param img: numpy array 格式的图片  
 :return:  
 """* # 展示图片  
 plt.imshow(img.reshape(h, w), 'gray')  
 plt.axis('off')  
 plt.show()  
  
  
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()  
  
  
def letterbox\_image(image, size):  
 *"""  
 调整图片尺寸  
 :param image: 用于训练的图片  
 :param size: 需要调整到网络输入的图片尺寸  
 :return: 返回经过调整的图片  
 """* new\_image = cv2.resize(image, size, interpolation=cv2.INTER\_AREA)  
 return new\_image  
  
  
def read\_one\_img(path):  
 *"""  
 根据路径读取一张人脸图片  
  
 :param path: 图片的路径  
 :return:  
 """* # 图片路径  
 # 以灰度模式读取图片  
 img\_sample = Image.open(path).convert('L')  
  
 # 把图片格式转为 numpy array 格式  
 img\_sample = np.array(img\_sample, 'uint8')  
  
 return img\_sample  
  
  
def get\_images(path):  
 *"""  
 读取输入的文件夹路径下的所有照片，读取输入的文件夹路径下的所有照片，将其转为 1 维，  
 统一保存到一个矩阵中，然依据图片名提取标签，最终该函数将输出这个照片矩阵及其中每  
 张照片的标签。  
  
 照片的命名格式请参照"person41\_01.png", 其含义为第41位志愿者的第01张人脸图像。  
  
 :param path: 照片存放的文件夹路径  
 :return: numpy matrix 格式的处理好的图片，及 list 格式的各个图片的标签  
 """* # 首先获取所有人脸图片的路径  
 image\_paths = [os.path.join(path, f) for f in os.listdir(path) if  
 f.endswith('png')]  
  
 # 所有的人脸区域图片都将保存在 images 这个矩阵中  
 images = np.mat(np.zeros((len(image\_paths), 112 \* 92)))  
  
 trainset\_labels = []  
  
 # 对于每一张图片  
 for index, image\_path in enumerate(image\_paths):  
 # 读取图片并将其转为灰度图  
 image\_pil = Image.open(image\_path).convert('L')  
  
 # 把图片转为 numpy array 格式  
 image = np.array(image\_pil, 'uint8')  
 image = letterbox\_image(image=image, size=(112, 92))  
  
 # 把 2 维的平面图像转为 1 维  
 img\_1D = image.flatten()  
  
 # 把处理后的图片保存到 images 中  
 images[index, :] = img\_1D  
  
 # 提取图片名作为图片的标签  
 trainset\_labels.append(int(image\_path.split('.')[-2][-2:]))  
  
 # 得到最终处理好的人脸图片和各个图片的标签  
 trainset\_labels = np.array(trainset\_labels)  
 return images, trainset\_labels  
  
  
def eigen\_train(trainset, k=20):  
 avg\_img = np.mean(trainset) # 计算训练集的平均人脸  
 # 计算中心化人脸即差值矩阵  
 differential\_matrix = trainset - avg\_img  
 norm\_img = differential\_matrix  
 # 计算差值矩阵的转置  
 differential\_matrix\_t = differential\_matrix.T  
 # 计算协方差矩阵  
 covariance\_matrix = np.dot(differential\_matrix, differential\_matrix\_t)  
 # 计算特征值和特征向量  
 eigenvalue, featurevector = np.linalg.eig(covariance\_matrix)  
 # 确保 k 的取值不会超限  
 k\_min = min(featurevector.shape[1], k)  
 # 截取原始的特征向量  
 feature\_vector = np.dot(differential\_matrix\_t, featurevector[:, -k\_min:])  
 # 计算特征矩阵  
 feature = np.dot(differential\_matrix\_t, np.dot(differential\_matrix, feature\_vector))  
 feature = feature.T  
 # 返回：平均人脸、特征人脸、中心化人脸  
 return avg\_img, feature, norm\_img  
  
  
def rep\_face(image, avg\_img, eigenface\_vects, numComponents=0):  
 *"""  
 用特征脸（eigenface）算法对输入数据进行投影映射，得到使用特征脸向量表示的数据  
  
 :param image: 输入数据  
 :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  
 :return: 输入数据的特征向量表示, 最终使用的特征脸数量  
 """* # 计算差值图像  
 difference\_image = image - avg\_img  
 # 选用的特征脸数量不能超过实际特征脸数量  
 num = min(eigenface\_vects.shape[0], numComponents)  
 # 取出需要的特征脸向量并进行归一化处理  
 selected\_eigenface\_vects = preprocessing.normalize(eigenface\_vects[:num, :], norm='l2')  
 # 计算特征脸的线性空间  
 linear\_space = selected\_eigenface\_vects.T  
 # 计算坐标，即特征向量表示的人脸  
 representation = np.dot(difference\_image, linear\_space)  
 # 返回：输入数据的特征向量表示, 特征脸使用数量  
 return representation, num  
  
  
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  
 """* num = min(numComponents, eigenVectors.shape[0]) # 获得符合规范的特征脸数量  
 eigenface\_vects = np.array(eigenVectors[:num, :]) # 根据需要的数量获得对应的矩阵(num,10304)  
 eigenface\_vects = preprocessing.normalize(eigenface\_vects, norm='l2') # 二范式归一化  
 linear\_space = eigenface\_vects.T # 获得坐标转换的线性空间  
 face = np.mat(representations) \* np.mat(linear\_space).T # 获得face的差值图片  
 face = np.array(face) + np.array(avg\_img) # 获得face的原始图片  
  
 # 返回: 重建人脸, str 使用的特征人脸数量  
 return face, 'numEigenFaces\_{}'.format(numComponents)  
  
  
datapath = './ORL.npz'  
ORL = np.load(datapath)  
# data为四维数组，第一维表示是哪个人，即人物标号1-40，第二维对应每个人的七张照片  
# [0][0][1][1]前两维定位到哪个人的哪张照片，后两位定位图像的像素点，即高和宽  
data = ORL['data']  
# label即对应图像的人物标号，40个人，每人7张照片  
label = ORL['label']  
num\_eigenface = 200  
  
# 划分数据集，下面参数代表，从40个志愿者中，每人挑选5张照片分为训练集，剩下的照片作为测试集  
# 生成的训练集train\_vectors为二维矩阵，每一行代表一张照片的N维向量，即将图片像素点的值按行排列  
# labels用来标明对应行表示的照片是哪个人，记录对应人物标号  
train\_vectors, train\_labels, test\_vectors, test\_labels = spilt\_data(40, 5, data, label)  
# 归一化，像素值在0-255之间  
train\_vectors = train\_vectors / 255  
test\_vectors = test\_vectors / 255  
  
print("训练数据集:", train\_vectors.shape)  
print("测试数据集:", test\_vectors.shape)  
  
# 返回平均人脸、特征人脸、中心化人脸  
avg\_img, eigenface\_vects, trainset\_vects = eigen\_train(train\_vectors, num\_eigenface)  
  
# 打印两张特征人脸作为展示  
eigenfaces = eigenface\_vects.reshape((num\_eigenface, 112, 92))  
eigenface\_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]  
plot\_gallery(eigenfaces, eigenface\_titles, n\_row=1, n\_col=2)  
  
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))  
  
print("重建训练集人脸")  
# 读取train数据  
image = train\_vectors[100]  
  
faces = []  
names = []  
# 选用不同数量的特征人脸重建人脸  
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")  
# 读取test数据  
image = test\_vectors[54]  
  
faces = []  
names = []  
# 选用不同数量的特征人脸重建人脸  
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)

1. **实验结果**

1.在自己电脑上运行：

运行代码，人脸识别准确率达到了91.25%；重建人脸效果也不错。



****

2.系统测试：

在Mo环境下进行代码运行测试，测试通过。



1. **总结**

本实验采用特征脸算法进行人脸识别，成功实现了PCA算法，建立了一个比较有效的人脸识别和重建的模型。

通过实验，我深入理解了PCA算法的原理和实现过程，包括协方差矩阵计算、特征向量选择等关键步骤。这让我对数据降维和特征提取有了更深入的理解。

在实现过程中，我遇到了一些细节问题，如协方差矩阵计算和人脸数据预处理等。通过自己的努力，成功解决了这些问题，并实现了更灵活的方法。另外，调整实验结果的过程中比较关键的点，在于如何选取特征向量，这里我也是经过多次尝试和调整，才找到了较为合适的方式。

可能改进的方向主要包括以下几个方面：

a）在人脸数据预处理方面，可以进一步优化图像的对齐和裁剪方法，以减少信息丢失。或者探索更高级的图像处理技术，如人脸检测和对齐算法，以提高数据的一致性和可用性。

b）尝试使用更先进的特征选择技术，如基于深度学习的特征提取方法，以提高特征的表达能力和分类性能。

c）采用更严格的实验设计和评估方法，如交叉验证、网格搜索等，以确保实验结果的可靠性和泛化能力。

总的来说，通过建立人脸识别模型和人脸重建模型，我加深了对视觉处理的认识，学会了将图形转变到另一子空间，再进行比较的办法。同时，自己实现协方差矩阵计算和特征向量选择的过程，也让我熟悉了降维方法的流程和细节，对机器学习和图像处理的原理和应用有了更深入的理解。