特征人脸识别问题程序报告

学号：2313312　　姓名：杜子妍

一、问题重述

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。该实验旨在使用特征脸（`eigenface`）算法实现人脸识别，涵盖数据准备、模型训练、特征提取、识别以及重建技术的完整流程。实验中需要先对人脸图像进行预处理，包括对齐和尺寸统一，然后通过主成分分析（`PCA`）提取主要特征，选择合适的特征脸数目以优化识别效果。利用这些特征，构建投影模型实现人脸识别，并通过重建原始人脸图像验证特征提取的表达能力。整个过程强调参数调节与模型优化，以提升识别的准确性和效率，同时提供相应的代码框架支持数据加载、裁剪和显示操作。

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。

二、设计思想

1. 数据预处理

构建训练集矩阵：$\text{trainset} \in \mathbb{R}^{N \times D}$
（N：样本数，D：像素数）

2. 计算平均脸

$\text{avg\_img} = \frac{1}{N} \sum_{i=1}^{N} \text{trainset}[i]$

3. 生成特征脸

（1）中心化数据：$\text{norm\_img} = \text{trainset} - \text{avg\_img}$

（2）协方差矩阵：$C = \text{norm\_img} \cdot \text{norm\_img}^T$

（3）特征分解（特征值问题）：$Cv = \lambda v$

（4）映射到高维空间：$u = \text{norm\_img}^T \cdot v$

（5）归一化：$u_{\text{norm}} = \frac{u}{\|u\|}$

4. 人脸重建

（1）投影系数：$\text{representations} = \text{norm\_img} \cdot u_{\text{norm}}$

（2）重建公式：$\text{reconstructed} = \text{avg\_img} + \sum_{i=1}^{k} \text{representations}[i] \cdot u_{\text{norm}}[i]$

三、代码内容

特征人脸算法

按照课本方法进行复现，完成对应代码内容

- 输入：n 个d维样本数据组成的矩阵 X，降维后的维数 l
- 输出：映射矩阵 $W = w_1, w_2, \ldots, w_l$
- 算法步骤：

    1. 对每个样本数据 x_i 进行中心化处理：
       $x_i = x_i - \mu, \quad \mu = \frac{1}{n}\sum_{j=1}^{n} x_j$
    2. 计算原始样本数据的协方差矩阵：
       $\Sigma = \frac{1}{n-1} X^T X$
    3. 对协方差矩阵 Σ 进行特征值分解，特征值按大小排序：
       $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_l$
    4. 取前 l 个最大特征值对应的特征向量 $w_1, w_2, \ldots, w_l$，组成映射矩阵 W
    5. 将每个样本数据 x_i 按照如下方法降维：
       $(x_i) \times 1 \times d(W)_{d \times l} = 1 \times l$

```
# 在生成 main 文件时，请勾选该模块
#trainset代表每一行代表一个展开后的人脸图像
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
    """

 ###############################################################################
    ####                    训练特征脸（eigenface）算法的实现                    ####
    ####                        请勿修改该函数的输入输出                         ####

 ###############################################################################
    #
    #计算平均人脸
    avg_img = np.mean(trainset, axis=0)
    # 数据中心化（每个样本减去平均脸）
    norm_img = trainset - avg_img
    # 计算协方差矩阵（像素数 x 像素数）
    covariance_matrix = np.dot(norm_img.T, norm_img)
    # 特征值分解，得到特征值和特征向量（矩阵是对称的）
    eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)
    # 根据特征值排序，从大到小
    idx = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[idx]
    eigenvectors = eigenvectors[:, idx]
    # 取前k个特征向量（特征脸）
    feature = eigenvectors[:, :min(k, eigenvectors.shape[1])].T  # 转置成每行一个特
征脸
    #
 ###############################################################################
    #############           在生成 main 文件时，请勾选该模块            #############

 ###############################################################################
```

```
    # 返回：平均人脸、特征人脸、中心化人脸
    return avg_img, feature, norm_img
```

直接按照课本方法复现代码如下，直接计算高维空间的协方差矩阵（像素数 x 像素数），利用`np.linalg.eigh()`对其进行特征值分解。这是标准的`PCA`方法。直接在高维像素空间进行特征分解，适合样本数较多、或像素数较少的场景。这里计算成本较高，在计算协方差的部分导致该算法运行速度极慢，故对代码进行优化

```
# 在生成 main 文件时，请勾选该模块
#trainset代表每一行代表一个展开后的人脸图像
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
    """

 ###############################################################################
    ####                    训练特征脸（eigenface）算法的实现                    ####
    ####                        请勿修改该函数的输入输出                         ####

 ###############################################################################
     # 计算平均人脸
    avg_img = np.mean(trainset, axis=0)

    # 数据中心化
    norm_img = trainset - avg_img
    norm_img_t = norm_img.T

    # 协方差矩阵 (样本数，样本数)
    covariance_matrix = np.dot(np.mat(norm_img),np.mat(norm_img_t))

    # 特征分解
    eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

    # 限制特征数
    k = min(eigenvectors.shape[1], k)

    # 映射到高维空间的特征向量
    eigenfaces = np.dot(norm_img_t,eigenvectors[:, :k])

    # 归一化特征向量 (保证单位长度)
    for i in range(eigenfaces.shape[1]):
        eigenfaces[:, i] = eigenfaces[:, i] / np.linalg.norm(eigenfaces[:, i])

    # 转换为 (k, 像素数) 的 numpy 数组
    eigenfaces = np.array(eigenfaces).T
    feature=eigenfaces

 ###############################################################################
    ##############           在生成 main 文件时，请勾选该模块            #############

 ###############################################################################
```

```
    # 返回：平均人脸、特征人脸、中心化人脸
    return avg_img, feature, norm_img
```

采用“降维技巧”，先计算低维空间（样本数 x 样本数）上的协方差矩阵,先在样本空间中得到特征向量（特征脸在样本空间中的投影）,然后映射回高维空间，得到最终的特征脸，特别适合当像素远多于样本数时。这是`Eigenfaces`中常用的特征脸算法优化方法，通过核技巧避免处理高维矩阵，提高效率，并且优化后的效果与未优化后的特征人脸结果是一致的。

人脸识别模型

```
# 在生成 main 文件时，请勾选该模块
from sklearn.preprocessing import normalize
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
    """


 ##########################################################################################
###
    ####  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
####
    ####                                请勿修改该函数的输入输出
####

 ##########################################################################################
###
    #
    #
    # 计算人脸图像与平均人脸的差异图像，用于后续特征提取
    difference_image = np.array(image) - np.array(avg_img)  # 得到差值图像（大小为
10304）

    # 选择用于主成分分析（PCA）的特征向量个数，取最小值
    num = min(eigenface_vects.shape[0], numComponents)

    # 获取前num个特征向量，并进行L2归一化
    eigenface_vect = normalize(np.array(eigenface_vects[:num, :]), norm='l2')  #
归一化后的特征向量矩阵（20, 10304）

    # 转置特征向量矩阵，得到人脸的线性空间（10304, 20）
    linear_space = eigenface_vect.T

    # 计算差值图像在特征空间的坐标（投影向量）
    coordinate = np.dot(difference_image, linear_space)  # （10304,） * （10304,
20） -> （20,）
```

```
    # 使用投影坐标表示人脸的特征向量（特征脸系数）
    representation = coordinate

    # 根据参数决定使用的特征脸个数
    numEigenFaces = numComponents if numComponents != 0 else
eigenface_vects.shape[0]
    #
    #

 ##########################################################################################
###
    ##############              在生成 main 文件时，请勾选该模块
##############

 ##########################################################################################
###

    # 返回：输入数据的特征向量表示，特征脸使用数量
    return representation, numEigenFaces
```

将输入图像与训练集的平均人脸进行差值，得到差异图像。然后，从特征脸矩阵中选择前指定数量的特征脸，并进行`L2`归一化，得到特征脸单位向量。接着，将特征脸矩阵转置，形成特征空间的基础。最后，通过点积计算差异图像在特征空间中的投影系数（即特征向量表示），反映出输入图像在特征脸空间的表达，实现将原始高维图像映射到低维的特征空间。

人脸重构

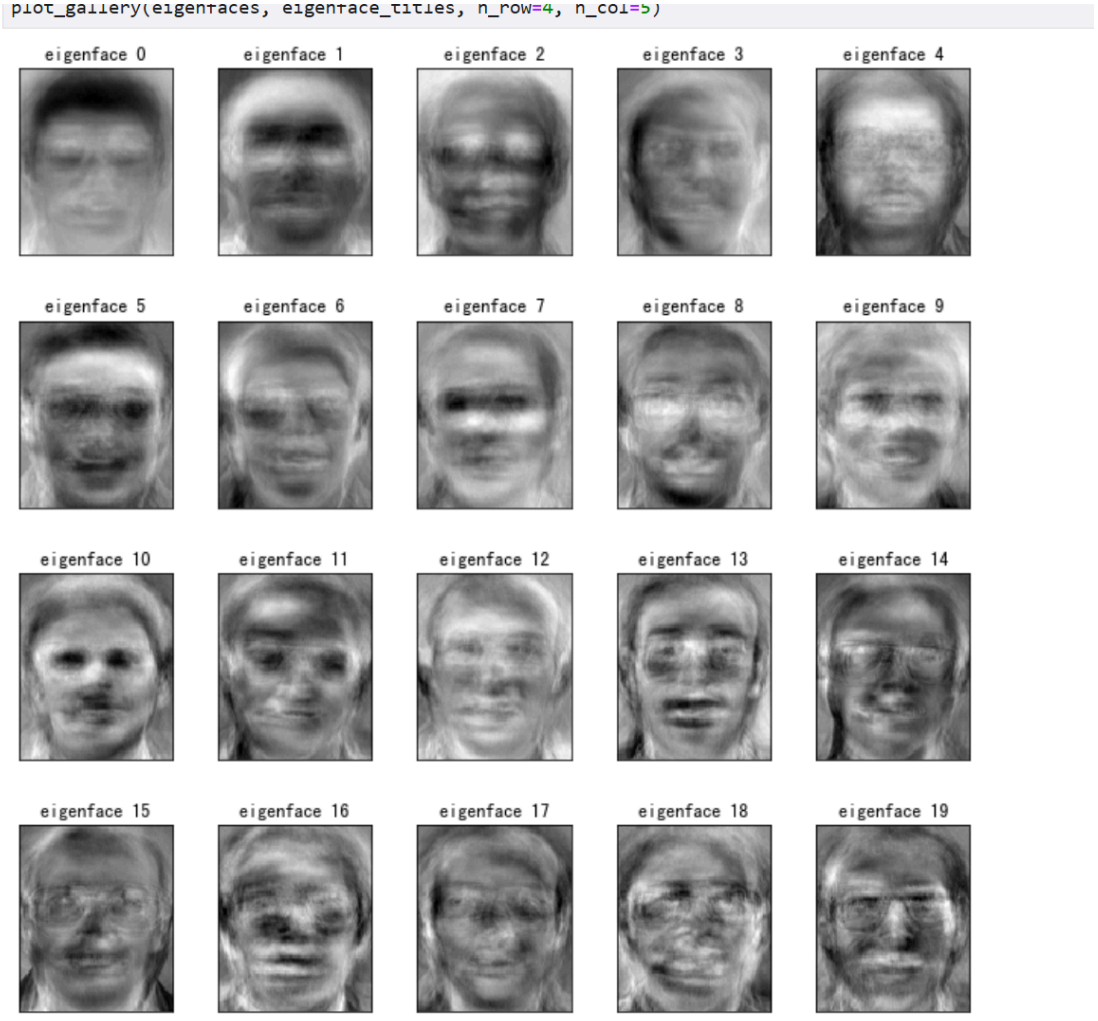
```
# 在生成 main 文件时，请勾选该模块

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
    """


 ##########################################################################################
    ####                              利用特征人脸重建原始人脸                              ####
    ####                              请勿修改该函数的输入输出                              ####

 ##########################################################################################
    #
#
    # 重建过程：重建 = 平均人脸 + 特征值投影
    face = np.dot(representations, eigenVectors[0:numComponents,])+avg_img

    # 转换为要求大小
    face = face.reshape(sz)
```

```
    #
#

##########################################################################################
    ##############            在生成 main 文件时，请勾选该模块            ##############

##########################################################################################

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

利用特征脸（`Eigenface`）的方法进行人脸的重建。它首先用给定的特征系数（`representations`）与前`numComponents`个特征脸向量（`eigenVectors`）进行线性组合（点积），得到一个偏差向量，再加上之前计算的平均脸（`avg_img`），以恢复出原始人脸的像素值。最后，将这个一维的像素数组重塑（`reshape`）为原始图片的尺寸（`sz`）以便显示或保存。

四、实验结果

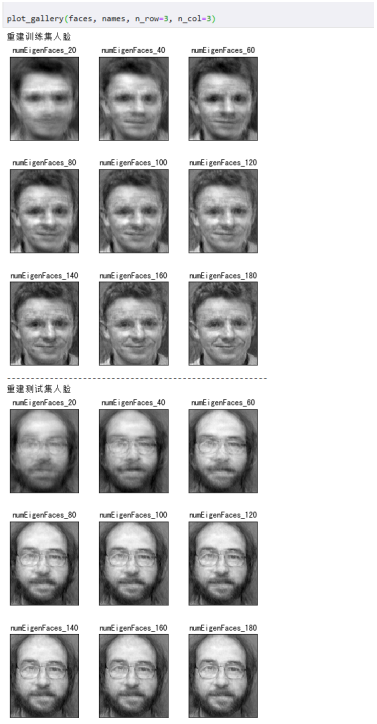
1.特征人脸的生成及准确率



```
    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: 91.25%
```

2.人脸重建效果



五、总结

1. 是否达到目标预期

在本次人脸识别实验中，我基本达到了预期目标。成功实现了`PCA`特征提取、特征脸可视化、特征向量的投影和重建算法流程。通过实验能够较好地区分不同的人脸，准确率为91.25%,并对测试样本作出准确的识别和重建，重建效果较好，人脸识别效果满足实验要求。

2. 可能改进的方向

虽然实验流程较为完整，但人脸识别算法的鲁棒性和泛化能力还有提升空间。可以考虑在以下方面进一步优化：

（1）引入更丰富的数据集，增强算法对不同光照、表情、姿态变化的适应性。

（2）使用深度神经网络（如卷积神经网络），提取高层语义特征。

---

3. 实现过程中遇到的困难

(1)直接使用课堂方法，运算时间过长，需要对主成分分析算法结合对应的训练数据进行优化调整和合理的选取

(2)特征脸矩阵与投影系数的运算维度匹配，经常要仔细检查矩阵的shape，防止形状不匹配导致的报错。

(3)数据归一化和`PCA过程实现时，如何有效避免信息损失。

---

4. 从哪些方面可以提升性能

(1)**算法优化**：优化`PCA`实现，可选用快速`PCA`或增量`PCA`以应对大数据集。

(2)**特征增强**：尝试融合多种特征，或引入深度学习方法，提高表征能力。

(3)**硬件层面**：在`GPU`、并行处理上实现矩阵运算，加速大规模人脸识别。

(4)**参数调整**：系统性地调整特征脸数量和归一化方式，提高识别与重建效果。
