程序报告

学号：2310422　　姓名：谢小珂

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

二、设计思想

本次实验为基于特征脸（Eigenface）算法的人脸识别实验。特征脸算法通过主成分分析获取，实验要求在数据准备阶段可自建或直接使用 ORL 人脸库，模型构建时进行数据预处理、完善算法函数并搭建识别与重建模型，最终勾选指定模块提交结果，以实现构建模型完成人脸识别与重建，并对比不同数量特征脸的识别和重建效果的目的。 以下是实验的设计思想及关键步骤：

1. PCA 的本质目标

PCA 的核心是通过线性变换，将原始高维数据（如 112×92=10,304 维的人脸图像）投影到一个低维子空间，这个子空间满足：

- 最大方差准则：保留数据变化最大的方向
- 正交性：各维度（主成分）之间线性无关
- 可逆性：可通过低维表示近似重建原始数据

2. 数学推导（分步骤详解）

步骤 1：数据中心化

- 计算平均脸（所有训练样本的均值）：

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \quad (x_i \in \mathbb{R}^{10304})$$

- 中心化数据：

$$X_c = [x_1 - \mu, x_2 - \mu, ..., x_N - \mu]^T \quad (X_c \in \mathbb{R}^{N \times 10304})$$

物理意义：消除共性的光照/背景影响，突出个体差异

步骤 2：协方差矩阵计算

• 传统方法（不可行）：

$$\Sigma = \frac{1}{N-1} X_c^T X_c \quad (\Sigma \in \mathbb{R}^{10304 \times 10304})$$

直接计算需要处理 10,304×10,304 矩阵，内存约需 8GB（float64）

• 小矩阵技巧（关键创新）：

计算：

$$\Sigma_{small} = \frac{1}{N-1} X_c X_c^T \quad (\Sigma_{small} \in \mathbb{R}^{N \times N})$$

然后通过数学关系得到特征向量：

$$\text{若 } \Sigma_{small} v = \lambda v, \text{ 则 } X_c^T v \text{ 是 } \Sigma \text{ 的特征向量}$$

步骤 3：特征值分解

• 对 Σ_{small} 进行特征分解：

$$\Sigma_{small} = V \Lambda V^T$$

其中 $\Lambda = diag(\lambda_1, ..., \lambda_N)$ 按升序排列

• 获取原始协方差矩阵的特征向量：

$$u_i = X_c^T v_i \quad (i = 1, ..., N)$$

需归一化：$u_i = u_i / \|u_i\|$

步骤 4：选择主成分

• 按特征值降序排序，保留前 k 个：

$$U_k = [u_{i_1}, u_{i_2}, ..., u_{i_k}] \quad (i_j \text{对应第j大的特征值})$$

• 方差解释率计算：

$$\text{解释率} = \frac{\sum_{j=1}^{k} \lambda_{i_j}}{\sum_{m=1}^{N} \lambda_m}$$

3. 在特征脸中的具体表现

<1>特征脸的性质：

基图像特性：每个特征向量 ui 可 reshape 为 112×92 的图像，呈现"幽灵人脸"形态

层级表征：

- 前 5%特征脸：编码全局人脸轮廓
- 中间特征脸：捕获五官特征
- 末尾特征脸：反映噪声/细节

<2>投影与重建过程：

- 投影（降维）：

$$y = U_k^T(x - \mu) \quad (y \in \mathbb{R}^k)$$

- 重建：

$$\hat{x} = U_k y + \mu$$

- 重建误差：

$$\|x - \hat{x}\|_2^2 = \sum_{i=k+1}^{N} (u_i^T x)^2$$

4. 参数选择策略

<1>k 值的确定方法：

- 肘部法则：绘制特征值累计解释率曲线，选择拐点

```
plt.plot(np.cumsum(eigenvalues)/np.sum(eigenvalues))
plt.xlabel('Number of components')
plt.ylabel('Explained variance ratio')
```

- 重构误差法：在测试集上评估不同 k 值的重建 PSNR

<2>典型取值经验：

- ORL 数据集：k=20 可保留 90%以上方差
- 高清人脸：k 可能需要 50-100

三、代码内容

1. 数据预处理

代码模块：load_data(), spilt_data(), plot_gallery()

功能：数据加载、划分和可视化

```
# 加载 ORL 数据集（.npz 格式）
def load_data(file_path):
    data = np.load(file_path)
    return data['arr_0'], data['arr_1']  # 图像数据（4D）和标签（3D）

# 划分训练集和测试集
def spilt_data(nPerson, nPicture, data, label):
    train = data[:nPerson, :nPicture, :, :].reshape(nPerson * nPicture, -1)  # 2D 化
    test = data[:nPerson, nPicture:, :, :].reshape(nPerson * (data.shape[1] - nPicture),
-1)
    return train, train_label, test, test_label

# 可视化图像（如平均脸、特征脸）
```

```
def plot_gallery(images, titles, h=112, w=92):
    plt.imshow(images[i].reshape((h, w)), cmap='gray')  # 1D 转 2D 显示
```

2. 特征人脸算法
代码模块：eigen_train()
功能：通过 PCA 计算特征脸

```
def eigen_train(trainset, k=20):
    avg_img = np.mean(trainset, axis=0)          # 计算平均脸
    norm_img = trainset - avg_img               # 中心化
    cov_matrix = np.dot(norm_img.T, norm_img)   # 小矩阵技巧
    eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)  # 特征分解
    top_k_eigenvectors = eigenvectors[:, np.argsort(eigenvalues)[::-1][:k]]  # 取前 k
个
    return avg_img, top_k_eigenvectors, norm_img
```

3. 人脸识别模型
代码模块：rep_face()
功能：将人脸投影到特征空间

```
def rep_face(image, avg_img, eigenface_vects, numComponents=20):
    centered_image = image - avg_img  # 中心化
    representation = np.dot(centered_image, eigenface_vects[:, :numComponents])  # 投
影
    return representation, numComponents
```

4. 人脸重建模型
代码模块：recFace()
功能：从低维表示重建人脸

```
def recFace(representations, avg_img, eigenVectors, numComponents, sz=(112, 92)):
    face = np.dot(representations, eigenVectors[:, :numComponents].T) + avg_img
    return face.reshape(sz), f'numEigenFaces_{numComponents}'  # 恢复原始尺寸
```

5. 全部代码展示

```
# 在生成 main 文件时，请勾选该模块

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


def load_data(file_path):
    # 加载.npz 格式的数据文件
    # 返回:
    #   arr_0: 图像数据，格式为(n_persons, n_images_per_person, height, width)
    #   arr_1: 对应标签数据，格式为(n_persons, n_images_per_person)
```

```
    data = np.load(file_path)
    return data['arr_0'], data['arr_1']


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

    :param nPerson : 志愿者数量（选择前 nPerson 个志愿者）
    :param nPicture: 各志愿者选入训练集的照片数量
    :param data : 等待分割的 4D 数据集（persons x images x height x width）
    :param label: 对应数据集的 3D 标签（persons x images）
    :return: 训练集(2D)，训练集标签(1D)，测试集(2D)，测试集标签(1D)
    """
    # 获取数据集维度信息
    allPerson, allPicture, rows, cols = data.shape  # 总人数，每人总图片数，图像高度，图像宽度

    # 训练集构造：选择前 nPerson 人，每人前 nPicture 张图片
    # 将 4D 数据转换为 2D 数组（样本数 x 特征数）
    train = data[:nPerson, :nPicture, :, :].reshape(nPerson * nPicture, rows * cols)
    # 对应标签展开为 1D 数组
    train_label = label[:nPerson, :nPicture].reshape(nPerson * nPicture)

    # 测试集构造：选择前 nPerson 人，每人剩余图片
    test = data[:nPerson, nPicture:, :, :].reshape(nPerson * (allPicture - nPicture), rows * cols)
    test_label = label[:nPerson, nPicture:].reshape(nPerson * (allPicture - nPicture))

    return train, train_label, test, test_label


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

    :param images: numpy array 格式的图片集合（样本数 x 特征数）
    :param titles: 图片标题列表
    :param h: 原始图像高度（用于 reshape）
    :param w: 原始图像宽度（用于 reshape）
    :param n_row: 展示行数
    :param n_col: 展示列数
    """
    # 创建画布并调整子图布局
```

```
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))  # 根据行列数动态调整画布大小
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)  # 调整子图间距

    # 遍历所有子图位置绘制图像
    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)  # 将 1D 特征向量 reshape 为 2D 图像
        plt.title(titles[i], size=12)
        plt.xticks(())  # 隐藏坐标轴
        plt.yticks(())
    plt.show()


# 在生成 main 文件时，请勾选该模块

def eigen_train(trainset, k=20):
    """
    特征脸训练过程

    :param trainset: 2D 训练集（样本数 x 特征数）
    :param k: 保留的主成分数量
    :return: 平均脸(1D)，特征向量(2D，每列是一个特征脸)，中心化数据(2D)
    """
    # 计算平均脸：所有训练样本的像素平均值
    avg_img = np.mean(trainset, axis=0)  # 沿样本维度求平均，得到 1D 数组（特征数,）

    # 数据中心化：每个样本减去平均脸
    norm_img = trainset - avg_img  # 广播机制自动扩展 avg_img 到每个样本

    # 计算协方差矩阵（使用小矩阵技巧）
    # 传统协方差矩阵为 (features x features)，但特征数（如 112*92=10304）会导致矩阵过大
    # 因此通过 (X^T X)/(n-1) 计算，其特征值与 (X X^T)/(n-1) 相同
    cov_matrix = np.dot(norm_img.T, norm_img) / (trainset.shape[0] - 1)

    # 特征值分解：cov_matrix = eigenvectors * eigenvalues * eigenvectors.T
    eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)  # 返回的特征值已排序（升序）

    # 选择前 k 个最大特征值对应的特征向量
    # argsort 返回升序排列的索引，[::-1]反转得到降序
    sorted_indices = np.argsort(eigenvalues)[::-1]
    top_k_eigenvectors = eigenvectors[:, sorted_indices[:k]]  # 特征向量按列存储
```

```
    return avg_img, top_k_eigenvectors, norm_img


# 在生成 main 文件时，请勾选该模块

def rep_face(image, avg_img, eigenface_vects, numComponents=20):
    """
    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（1D 数组，特征数）
    :param avg_img: 训练集的平均人脸（1D，特征数）
    :param eigenface_vects: 特征脸向量（2D，特征数 x k）
    :param numComponents: 选用的特征脸数量（k' <= k）
    :return: 投影后的特征向量（1D，k'），实际使用的主成分数
    """
    # 数据预处理：中心化
    centered_image = image - avg_img  # 将输入图像减去平均脸

    # 投影到特征空间：与特征向量矩阵的前 numComponents 列做点积
    representation = np.dot(centered_image, eigenface_vects[:, :numComponents])

    return representation, numComponents


# 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: 低维表示（1D 数组，numComponents）
    :param avg_img: 平均脸（1D，特征数）
    :param eigenVectors: 特征向量矩阵（2D，特征数 x k）
    :param numComponents: 实际使用的主成分数
    :param sz: 原始图像尺寸（高度，宽度）
    :return: 重建后的图像（2D），重建信息字符串
    """
    # 重建过程：低维表示 x 特征向量转置 + 平均脸
    face = np.dot(representations, eigenVectors[:, :numComponents].T) + avg_img

    # 将 1D 数组 reshape 为原始图像尺寸
    face = face.reshape(sz)  # 转换为 2D 图像矩阵

    return face, 'numEigenFaces_{}'.format(numComponents)
```

四、实验结果

测试结果如下图所示：

五、总结

1. 理论知识的实际应用

<1>PCA 降维的数学本质：

通过实验真正理解了 PCA 的四个关键步骤：

- 数据中心化（减去平均脸）
- 协方差矩阵计算（小矩阵技巧优化）
- 特征值分解（获取特征脸）
- 主成分选择（保留最大方差方向）

<2>特征脸的物理意义：

特征向量（特征脸）可视为“人脸基”，前几个主成分编码全局特征（如脸型），后续成分捕获细节（如五官）。通过 plot_gallery()可视化，直观看到了这一现象。

2. 工程能力的提升

<1>高维数据处理的技巧：

学会了用小矩阵技巧解决 10,304 维图像的协方差矩阵计算问题，避免了内存爆炸。

<2>模块化编程思维：

将实验拆分为四个模块：

- 数据预处理（load_data, spilt_data）
- 特征脸训练（eigen_train）
- 人脸识别（rep_face）
- 人脸重建（recFace）

每个函数职责单一，通过 NumPy 数组传递数据，代码可读性和复用性大幅提高。

3. 可视化与结果验证

<1>平均脸与特征脸的可视化：

使用 plot_gallery()展示平均脸和特征脸时，发现前几幅特征脸呈现“模糊人脸”形态，验证了 PCA 的层级表征特性。

<2>重建效果的评估：

通过 recFace()重建人脸时，对比了不同 k 值的重建效果：

- k=10 时，人脸轮廓模糊
- k=30 时，五官细节清晰

这让我理解了降维中“信息保留”与“压缩率”的权衡。

4. 对后续学习的启发

<1>传统方法与深度学习的对比：

特征脸算法虽然简单，但在小样本场景下仍有效。对比现代 CNN，理解了深度学习通过非线性变换能捕获更复杂的特征，但需要更多数据。

<2>优化方向：

- 结合 LDA（线性判别分析）提升分类效果
- 用 Kernel PCA 处理非线性特征
- 在自建人脸库上测试算法鲁棒性