程序报告

学号：2213041　　姓名：李雅帆

一、问题重述

1.实验背景与内容

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。

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。

==========

2.实验要求

（1）求解人脸图像的特征值与特征向量构建特征脸模型。

（2） 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。

二、设计思想

1.采用的方法

基于特征脸的人脸识别是一种经典的人脸识别方法，其核心思想是将人脸图像投影到一个低维的特征空间中，然后利用特征空间中的特征向量进行识别。其主要步骤包括数据预处理、特征提取和人脸重建三个部分。

（1）数据预处理：

①加载人脸数据集，包括人脸图像和对应的标签；

②将每张图像转换为统一的大小，以便后续处理。

（2）特征提取：

① 计算训练集中所有人脸图像的平均图像；

② 对每张人脸图像，将其与平均图像相减，得到中心化后的图像；

③ 计算中心化后的训练集的协方差矩阵；

④ 求解协方差矩阵的特征值和特征向量；

⑤ 选取前 k 个特征值对应的特征向量作为特征脸。

（3）识别：

①对测试集中的每张人脸图像，将其与平均图像相减，得到中心化后的图像；

②将中心化后的测试图像投影到特征脸空间中，得到其特征表示；

③计算测试图像与训练集中每张人脸的距离或相似度；

④根据距离或相似度，将测试图像分类到最近的类别，即识别出最相似的人脸。

==========

2.改进与优化方向

（1）数据预处理：

数据预处理主要包括数据集的分割。可以考虑添加更多的数据预处理步骤，例如图像的缩放、裁剪、去噪等，以提高模型的鲁棒性和泛化能力。

（2）特征提取（eigen_train 函数）：

当前采用的方法是基于特征值分解的特征提取，也就是特征脸方法，可能受到数据噪声和维数灾难的影响。可以尝试其他的特征提取方法，例如基于深度学习的方法，如卷积神经网络，或者其他降维技术，如主成分分析。

（3）特征表示（rep_face 函数）：

目前使用的是特征向量表示方法，它可以在一定程度上减少数据的维度。可以尝试局部特征描述子等其他方法，以提高模型对图像变换和噪声的鲁棒性。

（4）人脸重建（recFace 函数）：

目前使用的是简单的线性重建方法，但它可能无法捕捉到图像的复杂结构和纹理信息。可以尝试其他的重建方法，例如基于深度学习的生成对抗网络（GAN）方法，以生成更加逼真的人脸图像。

三、代码内容

1.数据预处理

进行人脸数据集的加载、预处理和准备工作。

首先，通过导入必要的库，定义了数据集分割和图片展示的两个函数。然后，从名为 ORL.npz 的数据集中加载了人脸图像和对应的标签。接着，利用 spilt_data 函数将数据集分割为训练集和测试集，其中训练集包含每个志愿者的前 5 张照片，而测试集包含剩余的照片。最后，对训练集和测试集的图像数据进行了归一化处理，将像素值缩放到 0 到 1 之间，并打印了数据集的形状，以便后续的特征提取和人脸识别任务。

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

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

    print("训练数据集:", train_vectors.shape)
    print("测试数据集:", test_vectors.shape)
```

==========

2.特征人脸算法

这段代码完成了特征脸算法的训练过程，即从训练数据中提取出主要的人脸特征，为后续的人脸识别任务做准备。

首先，通过 eigen_train 函数对训练集进行处理，计算了训练数据的平均人脸，并对训练数据进行了中心化处理；接着，利用奇异值分解（SVD）方法提取特征脸，即主要的人脸特征向量。这些特征向量被保存在 feature 中；然后使用 plot_gallery 函数展示了两张特征人脸图像，以便观察和分析特征脸的效果。

```
def eigen_train(trainset, k=20):

    # 计算训练集的平均人脸
    avg_img = np.mean(trainset, axis=0)
```

```
    # 计算中心化的训练数据
    norm_img = trainset - avg_img

    # 使用奇异值分解（SVD）提取特征脸
    _, _, v = np.linalg.svd(norm_img.T, full_matrices=False)
    feature = v[:k, :]

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

==========

3.人脸识别模型

这段代码完成了基于特征脸算法的人脸识别任务，并输出了识别准确率的结果。

首先，通过 rep_face 函数对输入的人脸图像数据进行投影映射，将其转换为使用特征脸向量表示的数据，并返回该数据的特征向量表示以及使用的特征脸数量；然后，利用训练集中的特征向量表示，计算测试集中每张人脸图像与训练集中所有人脸图像的相似度，采用欧氏距离作为相似度度量；最后，根据相似度结果，判断测试集中的人脸图像所属的类别，并统计正确分类的数量，从而计算出人脸识别的准确率。

```
def rep_face(image, avg_img, eigenface_vects, numComponents=0):
    if numComponents == 0:
        numComponents = eigenface_vects.shape[0]

    # 将输入数据与特征脸向量相乘，得到数据的特征向量表示
    representation = np.dot(image - avg_img, eigenface_vects[:numComponents, :].T)

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

==========

4.人脸重建模型

这段代码完成了特征人脸算法下的人脸重建任务，并通过可视化展示了不同特征脸数量对重建结果的影响。

首先，通过 rep_face 函数将输入的人脸图像数据投影映射到特征空间，得到其特征向量表示，并通过 recFace 函数利用特征脸向量重建原始人脸；然后，根据不同的特征脸数量，对同一张人脸图像进行多次重建，以展示不同数量特征脸对重建效果的影响；最后，通过 plot_gallery 函数展示了重建的人脸图像以及使用的特征脸数量，从而直观地比较了不同数量特征脸下的重建效果。

```
def recFace(representations, avg_img, eigenVectors, numComponents, sz=(112, 92)):
    # 利用特征脸向量重建原始人脸
    face = avg_img + np.dot(representations, eigenVectors[:numComponents, :])

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

==========

5.提交的代码

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

def spilt_data(nPerson, nPicture, data, label):
    # 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 eigen_train(trainset, k=20):
    # 计算平均脸
    avg_img = np.mean(trainset, axis=0)

    # 中心化训练数据
    norm_trainset = trainset - avg_img

    # 计算协方差矩阵
    cov_matrix = np.cov(norm_trainset, rowvar=False)

    # 计算特征值和特征向量
    eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)

    # 对特征向量进行排序
    sorted_indices = np.argsort(eigenvalues)[::-1]
    sorted_eigenvectors = eigenvectors[:, sorted_indices]

    # 选取前 k 个特征向量
    feature = sorted_eigenvectors[:, :k]

    return avg_img, feature, norm_trainset
def rep_face(image, avg_img, eigenface_vects, numComponents=0):
    # 计算输入数据与平均脸的差异
    diff = image - avg_img

    # 如果未指定特征脸数量，则使用全部特征脸
    if numComponents == 0:
        numComponents = eigenface_vects.shape[1]

    # 选取前 numComponents 个特征脸
    selected_eigenfaces = eigenface_vects[:, :numComponents]

    # 计算特征向量表示
    representation = np.dot(diff, selected_eigenfaces)
    return representation, numComponents
def recFace(representations, avg_img, eigenVectors, numComponents, sz=(112,92)):
    # 选取前 numComponents 个特征向量
    selected_eigenfaces = eigenVectors[:, :numComponents]

    # 利用特征脸重建原始人脸
    face = avg_img + np.dot(representations, selected_eigenfaces.T)

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

四、实验结果

系统测试通过截图如下：

五、总结

达到了求解人脸图像的特征值与特征向量构建特征脸模型，并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的预期。

但是测试时长较长，还可以考虑从数据增强、模型优化、并行计算等方面来提升性能。