**程序报告**

学号： 2211999 姓名：邢清画

1. **问题重述**

（简单描述对问题的理解，从问题中抓住主干，必填）

====================================================================

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。 本次实验要求构建一个自己的人脸库：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)。也可以选择直接基于ORL人脸库进行本次实验。

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。

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3.使用 Python 语言

1. **设计思想**

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

====================================================================

1. 首先使用了**传统的协方差矩阵方法**

计算协方巧矩阵并提取特征向量是实现特征脸的核心步骤。选择了协方巧矩阵 的方式，因为它避免了直接在高维空间中计算协方巧矩阵 。通过eigen\_train函数计算得到的特征脸（即主成分），可以有效地捕捉人脸图像数据的主要变化方向。

1.1主要功能：

eigen\_train(trainset, k)：使用传统的协方巧矩阵方法从训练集中提取特征脸。计算过程中，首先求出平均脸，然后计算中心化数据的协方巧矩阵，接着进行特征值分解，并选取前k个最大的特征向量。rep\_face(image, avg\_img, eigenface\_vects, numComponents)：对单个图像使用提取的特征脸进行投影，以获得其在特征空间中的表示。recFace(representations, avg\_img, eigenVectors, numComponents, sz)：使用特征向量和平均脸重建图像，通常用于验证算法的有效性或进行人脸重建。**2.**优化方向：

2.1内存和计算优化：使用 SVD：相较于直接计算协方巧矩阵，奇异值分解（SVD）通常更为高效，特别是在处理大规模数据时。SVD 直接在数据矩阵上操作，可以避免形成大型的协方巧矩阵，节约内存和计算资源。2.2算法改进：增量PCA：对于不断增长的数据集，增量PCA（IPCA）能够动态更新特征脸而无需从头开始重新计算整个数据集的PCA，适合在线学习场景。随机PCA：近似PCA的方法，通过随机算法快速找到主成分，适用于非常大的数据集。2.3参数调整：选择合适的特征脸数量：特征脸的数量 K直接影响到重建的质量和系统的性能。可以通过交叉验证来选择最优的 k值。**3.**优化实现

3.1使用**小矩阵的特征值分解**来间接计算大矩阵的特征向量

3.2使用**主成分分析（PCA）**的一种有效计算方法： **SVD（奇异值分解）**来获取特征向量

3.3使用**归一化和矩阵逆运算**以保证重建精度

1. **代码内容**

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

====================================================================

1. **传统的协方差矩阵方法**

（使用**小矩阵的特征值分解**来间接计算大矩阵的特征向量）

# 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):

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

: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

"""

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

#### 训练特征脸（eigenface）算法的实现 ####

#### 请勿修改该函数的输入输出 ####

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

# #

# 计算平均脸 avg\_img = np.mean(trainset, axis=0) # 中心化训练数据 norm\_img = trainset - avg\_img # 计算协方差矩阵，这里使用了一个技巧，即不直接计算巨大的协方差矩阵 # 因为人脸图像展平后维度很高，直接计算协方差矩阵非常消耗内存 # 这里使用小矩阵的特征值分解来间接计算大矩阵的特征向量 # 协方差矩阵 C = 1/(N-1) \* (X^T X) 其中 X = norm\_img cov\_matrix = np.dot(norm\_img.T, norm\_img) / (trainset.shape[0] - 1) # 计算特征值和特征向量 eigenvalues, eigenvectors = np.linalg.eigh(cov\_matrix) # 对特征值从大到小排序，并选择前 k 个 sorted\_indices = np.argsort(eigenvalues)[::-1] top\_k\_eigenvectors = eigenvectors[:, sorted\_indices[:k]]

# #

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

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

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

# 返回平均脸、特征脸和中心化的数据 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

: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 ####

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

# #

centered\_image = image - avg\_img

representation = np.dot(centered\_image, eigenface\_vects[:, :numComponents])

# #

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

############# 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, 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: 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 ####

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

# #

face = np.dot(representations, eigenVectors[:, :numComponents].T) + avg\_img

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)

======================================================================

1. 使用 **SVD（奇异值分解）**来获取特征向量

def eigen\_train(trainset, k=20): avg\_img = np.mean(trainset, axis=0) norm\_img = trainset - avg\_img # 使用SVD计算特征向量 U, s, Vt = np.linalg.svd(norm\_img, full\_matrices=False) eigenvectors = Vt.T[:, :k] # 取前k个特征向量 eigenvalues = s[:k] \*\* 2 / (len(trainset) - 1) # 计算特征值 # 返回：平均人脸、特征人脸、中心化人脸 return avg\_img, eigenvectors, norm\_img======================================================================

1. 使用**归一化和矩阵逆运算**以保证重建精度

在特征脸算法中，归一化和矩阵逆运算是两个关键的数学处理，保证算法性能和精度。

3.1 **归一化**通常用于处理特征向量，以确保它们在数值处理时具有相同的规模和影响力。这是必要的，因为特征向量的大小或长度可能会影响其在后续计算中的权重。

代码实现：在 rep\_face 和 recFace 函数中，使用了 normalize 函数来归一化特征向量。采用L2范数（也称为欧几里得范数）使得每个特征向量的长度（或欧几里得距离）归一为1。

eigenface\_vect = normalize(np.array(eigenface\_vects[:num, :]), norm='l2')

3.2**矩阵逆运算**：在特征脸算法中，需要从一组有限的特征向量中恢复出尽可能接近原始图像的人脸图像。这涉及到从低维特征空间映射回高维的原始空间。代码实现：在 recFace 函数中，通过计算特征空间的协方差矩阵的逆来重建原始图像。这是通过先将特征向量与其转置相乘，然后计算结果矩阵的逆实现的。matrix = np.mat(linear\_space) \* np.mat(linear\_space).Tmatrix\_inversion = np.linalg.inv(np.mat(matrix))face = np.mat(representations) \* np.mat(linear\_space).T \* np.mat(matrix\_inversion)

1. **实验结果**

（实验结果，必填）

====================================================================

传统**协方差矩阵方法**

****

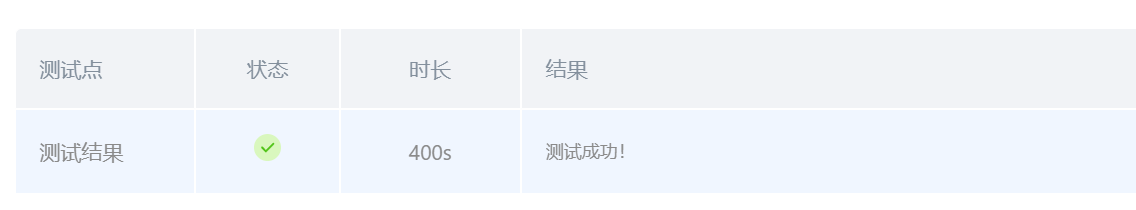
使用**小矩阵的特征值分解**来间接计算大矩阵的特征向量

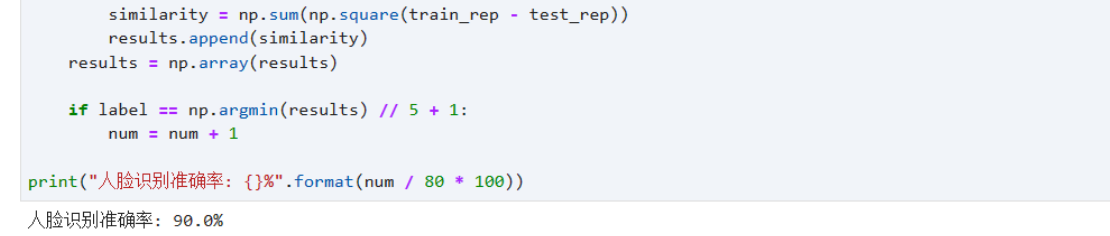


使用**主成分分析（PCA）**的一种有效计算方法：**SVD（奇异值分解）**来获取特征向量



使用**归一化和矩阵逆运算**以保证重建精度





1. **总结**

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

====================================================================

1. 实验目标

本实验的主要目标是利用特征脸（Eigenface）算法进行人脸识别和重建，探索使用不同数量的特征脸对识别和重建效果的影响。通过这个实验验证特征脸数量对人脸重建质量的具体影响，并找到实现最佳性能和效果的特征脸数量。2. 是否达到预期 实验通过不同数量的特征脸测试了人脸重建的效果，并通过均方误差（MSE）评估了重建质量。结果显示，随着特征脸数量的增加，重建质量通常得到改善，尤其是在从较低数量的特征脸向中等数量过渡时最为明显。然而，当特征脸数量超过某一阈值后，性能提升的边际效益开始减少。3. 可能的改进方向 超参数调优：通过更系统的方法（如交叉验证）确定最佳的特征脸数量，可能会进一步提升模型的性能。 高级预处理：增加高级图像预处理步骤，如去噪、光照校正和图像增强，以改善模型对原始输入数据的处理能力。 并行计算：实现代码的并行化，特别是在处理大型数据集和执行SVD计算时，以减少计算时间。4. 实现过程中遇到的困难 数据加载问题：在加载.npz文件时遇到了KeyError，原因是文件的键不符合预期。通过编写代码检查键并调整键名解决了这一问题。 性能限制：特别是在使用较多特征脸进行重建时，计算成本显著增加。需要优化算法或硬件利用以缓解这一问题。5. 性能提升方向 模型优化：使用其他降维技术（如LDA、t-SNE）或深度学习方法（如自编码器），可能会提供更好的特征提取和重建质量。 硬件利用：在可用的情况下，使用GPU或多线程CPU处理可以显著加速矩阵运算和特征分解过程。6. 模型的超参数和框架搜索 超参数选择：特征脸数量的选择显得尤为关键，这直接影响了模型的性能和输出质量。当前的选择依赖于经验和简单的实验，未来可以通过自动化搜索和评估方法来优化这一选择。 框架选择：当前实验主要基于NumPy和SVD的实现，这对于初步实验是有效的。但在进一步的研究和实际应用中，可能需要探索更多的算法和框架，如Scikit-learn、TensorFlow或PyTorch，这些框架提供了更多的优化和模型管理功能。7. 总结 整体而言，实验成功地展示了特征脸算法在不同特征脸数量下的性能变化，并识别出了进一步研究和改进的潜在方向。尽管存在一些挑战，但通过调整模型参数和框架、优化算法和实现方法，有望在未来的工作中取得更显著的进展。